MSX

GUIA TÉCNICO DE REFERÊNCIA

Inclui: Dicionários dos Comandos
da Linguagem Basic
Sistemas Operacionais e Bios

VOLUME
2

TOSHIYUKI SATO
PAUL MAPSTONE
ISABELLA MURIEL

McGraw-Hill

MSX
Guia do Programador
Guia Técnico de Referência
Incluindo Dicionário dos comandos
da linguagem Basic
Sistema Operacional e Bios
Volume 2

Valorize sua formação profissional,
seu futuro, sua consciência

MSX
Guia do Programador
Guia Técnico de Referência
Incluindo Dicionário dos comandos da linguagem Basic
Sistema Operacional e Bios
Volume 2

TOSHIYUKI SATO PAUL MAPSTONE ISABELLA MURIEL

Tradução:

Flávio Denny Steffen

Lars Gustav Erik Unonius

Revisão Técnica:

Oscar Burd

Daniel Burd

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

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

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

Do original
The Complete MSX – Programmers Guide

EDITOR: MILTON MIRA DE ASSUMPÇÃO FILHO

Coordenadora de revisão: Daisy Pereira Daniel
Supervisor de produção: José Rodrigues

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

Sato, Toshiyuki.
S266g MSX – Guia do programador – Guia técnico de referência – Sistema operacional e bios / Toshiyuki Sato, Paul Mapstone, Isabella Muriel ; tradução Flávio
v.1 e 2 Denny Steffen, Lars Gustav Erik Unonius. – São Paulo : McGraw-Hill, 1988.

1. Microcomputadores – Programação 2. MSX (Computadores) – Programação 3. MSX-BASIC (Linguagem de programação para computadores) I. Mapstone, Paul. II. Muriel, Isabella. III. Título.

CDD-001.6424
88-0192 -001.642

Índices para catálogo sistemático:

1. Microcomputadores : Programação : Processamento de dados 001.642
2. MSX : Computadores : Programação : Processamento de dados 001.642
3. MSX-BASIC : Linguagem de programação : Computadores : Processamento de dados 001.6424

SUMÁRIO

Volume 2

Parte III

GUIA TÉCNICO DE REFERÊNCIA

CAPÍTULO 52

COMANDOS EM BASIC

EXPLANAÇÃO

FORMATO DOS COMANDOS EM BASIC

Este capítulo contém todos os comandos do MSX-BASIC, em ordem alfabética, e tudo o que precisar conhecer a respeito deles. As descrições dos comandos são fornecidos de forma padrão, facilitando o encontro de seus significados. A descrição de cada comando é desdobrada da seguinte forma:

Comando

Seguido algumas vezes por sua derivação.

Descrição

Esta seção explica em termos simples o que o comando faz.

Sintaxe

Enumera cada possibilidade de sintaxe com as correspondentes explicações e as diferenças de cada formato. Serão utilizados os seguintes símbolos:

<const-num>	É um número normal como 100 ou 1.414.
<var-num>	É uma variável numérica como X ou AMOUNT.
<exp-num>	Significa uma expressão que resulta em um número como 2*P*R*COS(Y). <const-num> e <var-num> podem ser considerados como simples expressões e podem ser utilizados sempre que uma expressão for necessária.
<numérico>	Pode ser qualquer um dos acima.
<const-string>	Indica uma string de caracteres envolvidos por aspas, isto é, "MSX COMPUTER 64K".
<var-string>	Significa uma variável string como B$ ou WORD$.
<exp-string>	Significa uma expressão que resulta em uma string, isto é, B$+"FAMÍLIA BURD", "4", STR$(1234).
<variável>	Poderá ser <var-string> ou, ainda, <var-num>.
<condição>	Indica uma condição verdadeira (TRUE) ou falsa (FALSE) de teste como A < 0 ou B$="FAMÍLIA BURD".
<declaração>	Pode ser qualquer declaração BASIC ou um grupo de funções BASIC.
<linha>	Significa o número da linha.
<nome>	Pode ser um nome de programa ou o nome de uma função.

Exemplos

Esta seção inclui vários exemplos de uma linha e programas curtos para ilustrar como o comando pode ser utilizado. Será dada uma breve explicação do exemplo.

Para programas-exemplos curtos serão utilizados os seguintes símbolos:

Qualquer coisa dentro deste
quadro é um programa-exemplo que
pode ser executado (RUN)
em seu computador.

O resultado do programa-exemplo será apresentado neste quadro. Poderá ser dada uma explicação suplementar dentro deste quadro na mesma linha.

Pontos a serem lembrados

Esta seção fornece detalhes sobre o que faz e o que não faz o comando. Uma lista compreensiva de dicas e conselhos será fornecida, bem como qualquer descrição suplementar do comando.

Corrigindo erros

Esta seção fornece possíveis causas de erro associadas ao comando e tem a intenção de ajudá-lo na depuração de seu programa de forma simples.

Comandos associados e outras referências

Esta seção relaciona todos os comandos associados que, em geral, são utilizados juntamente com o comando em questão. Caso sejam dados detalhes, ou um exemplo da utilização do comando, mencionaremos em que parte se encontram os comandos associados.

ABS VALOR ABSOLUTO (ABSOLUTE VALUE)

Descrição

Esta função devolve o valor absoluto, que é a distância entre o número do parênteses e o zero; isto é, os números negativos se transformam em positivos e os positivos se mantêm. Por exemplo, o valor absoluto de –2.6 é 2.6.

Emprega-se também para calcular a diferença positiva entre dois números.

Sintaxe

ABS (<const_num>)
ABS (<var-num>)
ABS (<exp-num->)

Exemplo

```
10 PRINT"VALOR ABSOLUTO DE -1000 E";ABS(-1000)
20 A=1984
30 B=1960
40 PRINT ABS(B-A)
```

```
VALOR ABSOLUTO DE -1000 E 1000
24
```

Pontos a serem lembrados

ABS é uma função com argumento e resultado de tipo numérico. O valor ABS de uma variável indefinida é zero.

Corrigindo erros

A mistura de uma função ABS com uma string de caracteres provocará um erro de incompatibilidade de tipos (type mismatch).

Comandos associados e referências

SGN

Parte I, Capítulo 15 (Vol. 1): *Funções*.

AND — OPERADOR LÓGICO AND

Descrição

AND é um dos operadores lógicos que executam um teste de condição múltipla utilizando álgebra booleana. A álgebra booleana é uma parte muito importante da ciência de computação. Para maiores detalhes, verifique a seção sobre Álgebra Booleana.

AND é freqüentemente utilizado com a declaração IF...THEN...ELSE para testar mais que duas condições antes que uma ação resultante seja tomada. Por exemplo:

```
IF A=5 AND B$="GODZILLA" THEN PRINT "AAARGH!" ELSE
PRINT "OK!"
```

A operação algébrica de AND pode ser relacionada em uma tabela da verdade como a seguir:

AND	X	Y	X AND Y
	0	0	0
	1	0	0
	0	1	0
	1	1	1

Portanto, 100 AND 50 é calculado da seguinte forma:

	X	100	0000000001100100
AND	Y	50	0000000000110010
		32	0000000000100000

Sintaxe

IF <condição> AND <condição> ...THEN...ELSE...
<var-num> = <const-num> AND <const-num>
<var-num> = <var-num> AND <exp-num>

ou outras combinações numéricas.

Exemplos

Exemplo do teste de condição IF THEN ELSE:

```
10 T=12:FOME$="MUITA"
20 INPUT "VOCE QUER ALMOCAR (S/N)";R$
30 IF T=12 AND FOME$="MUITA" AND R$="S"OR R$="s" THEN
PRINT "COMA ALGUNS SANDUICHES":END
40 PRINT"OK.ENTAO COMA MAIS TARDE"
```

Processe este programa duas vezes. Veja o que acontece quando você digita sim (S) e quando você digita não (N).

```
RUN
VOCE QUER ALMOCAR (S/N)? S
COMA ALGUNS SANDUICHES
```

Exemplo da álgebra booleana:

```
10 CLS
20 A%=185: B%=85
30 C%=A%ANDB%
40 PRINT A%;" AND ";B%;"=";C%
50 PRINT "O RESULTADO VEZES 100 = ";C%*100

 185  AND  85 = 17
O RESULTADO VEZES 100 =  1700
```

Pontos a serem lembrados

O AND lógico pode ser utilizado para testar os bits de um determinado número. Por exemplo, se $Y=2^n$, onde n é o bit de teste de valor entre 0 e 7, e se X AND Y = Y, então o bit de teste n é verdadeiro (vale 1). Se X AND Y = 0, então o bit de teste n é falso (vale 0).

Faça o teste com o 5º bit (n = 5) para X = 117.

```
Y=2^5=32
        X     117     0000000001110101
AND     Y      32     0000000000100000
-------------------------------------
               32     0000000000100000
```

portanto verdadeiro.

Os operandos booleanos tem de estar na faixa inteira de –32768 até 32767. Números reais são truncados e tornam-se inteiros.

Corrigindo erros

Um erro de extravasamento (overflow) ocorre quando um dos operandos externo da faixa de inteiros.

Ocorre um erro de incompatibilidade de tipos (type mismatch) quando um dos operandos é uma string.

Comandos associados e referências

IF
THEN
ELSE
OR
EQV
XOR
NOT
IMP

Parte 1, Capítulo 6 (Vol. 1): "O Uso de Condições".
Parte 2, Capítulo 34 (Vol. 1): "Álgebra Booleana I : Expressões Lógicas".
Parte 3, Capítulo 35 (Vol. 1): "Álgebra Boolena II: O IF/THEN3ELSE".

ASC — CÓDIGO ASCII

Descrição

Cada carctere tem um número de código correspondente denominado código ASCII. (ASCII significa *Código Padrão Americano Para Troca de Informação.*) O computador converte todas as strings em números porque trabalha internamente só com números. O padrão ASCII foi estabelecido de modo que todos os computadores que o utilizam tenham o mesmo código numérico para seus caracteres alfanuméricos. Isto torna a comunicação entre computadores mais fácil.

Algumas vezes é necessário encontrar o código ASCII durante a programação; ASC converte um determinado caractere ou o primeiro caractere de uma string no código ASCII correspondente.

Sintaxe

ASC("<string>")	Fornece o código ASCII do primeiro caractere da <string> <string> pode ter mais de um caractere.
ASC(<var-string>)	Fornece o código ASCII do primeiro caractere da variável string.

Exemplos

```
PRINT ASC ("A")
```

dará como resultado o código de "A", que é 65.

```
10 CLS
20 A$="FESTA"
30 B=ASC(A$)
40 PRINT "O VALOR ASCII DA PRIMEIRA LETRA DE  ";A$;" E
";B
```

O programa acima fornecerá:

```
O VALOR ASCII DA PRIMEIRA LETRA DE  FES
TA E 70
```

Pontos a serem lembrados

Todos os caracteres, incluindo os códigos de controle como <RETURN> e os caracteres gráficos, têm seus próprios códigos ASCII que podem ser calculados utilizando ASC.

Apenas o primeiro caractere de uma string tem significado em ASC. Os demais caracteres são ignorados.

O processo reverso, ou seja, a geração de um caractere a partir de um número ASCII pode ser conseguida utilizando o comando CHR$, mas não se pode produzir mais que um caractere por vez.

Corrigindo erros

Se uma variável string não tem nenhum conteúdo resultará em um erro do tipo "Illegal Function Call" (executou-se um comando de forma ilegal.)

Por exemplo: 10 PRINT ASC("") resultará "Illegal Function Call" (chamada de função ilegal) em 10.

Ocorre um erro de incompatibilidade de tipo ("type mismatch") se o argumento é numérico.

Comandos associados e referências

CHR$

Parte 1, Capítulo 20 (Vol. 1): "O Código ASCII".

ATN ARCO-TANGENTE

Descrição

Esta função retorna o arco-tangente do argumento em radianos entre –PI/2 e PI/2.

Sintaxe

ATN(<const-num>)
ATN(<var-num>)
ATN(<exp-num>)
ATN fornece o arco-tangente em radianos com precisão dupla.

Exemplos

```
PRINT ATN(1.5)
    .98279372324731
PRINT ATN(3/6)
    .46364760900081
```

PI=4*ATN(1) ... 4*ATN(1) calcula o valor de PI até 14 casas decimais.

Pontos a serem lembrados

ATN sempre fornece um número com precisão dupla.

O argumento pode ser qualquer tipo de variável numérica.

Corrigindo erros

Como se trata de uma função trigonométrica, obviamente não deveria ser misturada com strings. Caso o seja, resultará em um erro de incompatibilidade de tipo ("type mysmatch").

Comandos associados e referências

TAN
SIN
COS

Parte 1, Capítulo 19 (Vol. 1): "Funções Matemáticas".

AUTO NUMERAÇÃO AUTOMÁTICA DE LINHAS

Descrição

Programas em BASIC necessitam de um número de linha para cada linha do programa. Ao digitar um programa muito longo, o comando AUTO vai colocá-lo no modo de numeração automático de linhas, que vai oferecer-lhe um novo número de linha toda vez que a tecla <RETURN> for pressionada.

Isto facilita muito as coisas para o programador. Os números das linhas terão um incremento de tamanho constante.

Sintaxe

AUTO	Fornece os números de linha a partir do 10 com incremento 10.
AUTO <const-num>	Fornece números de linha a partir do número especificado com incremento de 10.
AUTO <const-num> , <const-num>	Fornece números de linha a partir do número especificado, com incremento no valor do segundo número.
AUTO, <const-num>	Fornece números de linha a partir do 0 com o incremento especificado em <const-num>.
AUTO <const-num>	Fornece números de linha a partir do número de linha especificado com o mesmo incremento anteriormente utilizado.

Exemplos

AUTO fornecerá números de linha 10, 20, 30, 40, e assim por diante. AUTO 1000,5 fornecerá os números de linha 1000, 1005, 1010, 1015, e assim por diante.

Pontos a serem lembrados

Toda vez que <RETURN> é pressionado, o computador fornece um novo número de linha.

Existem duas maneiras de sair do modo AUTO:

1. Pressione <CTRL> e <STOP> ao mesmo tempo.
2. Pressione <CTRL> e <C> ao mesmo tempo.

A faixa de números de linha que você pode ter vai de 0 até 65529, e você pode selecionar o incremento de 1 até 65529. Quando o computador chega na linha 65529, automaticamente pára de colocar números de linha, voltando ao modo direto.

Caso haja um número de linha existente no programa que você estiver editando, o computador vai fornecer-lhe um sinal de aviso na forma de um sinal "*" após o número da linha. Caso você não queira mudar aquela linha, pressione a tecla de retorno; mas, caso esteja substituindo aquela linha, ignore o aviso "*".

A tecla de função (F-2) é inicializada em "AUTO" quando o computador é ligado, sendo que você pode colocar no modo AUTO ao toque de uma tecla.

Os números de linha têm de ser inteiros.

Corrigindo erros

Caso o número da linha ou o tamanho do incremento especificado não sejam inteiros, isso resultará em um erro de Sintaxe ("Syntax Error").

É muito fácil cancelar involuntariamente parte do programa com o modo AUTO de modo que você deve prestar atenção nos avisos "*". AUTO pode ser incluído em um programa, mas não tem muito sentido. Não o recomendamos.

Comandos associados e referências

RENUM

Parte 1, Capítulo 7 (Vol. 1): "Comandos Úteis e Dicas para Escrever Programas".

Parte 2, Capítulo 29 (Vol. 1): "Edição Avançada de Programas".

BASE

Descrição

Esta variável de sistema fornece a posição do endereço inicial atual das diversas tabelas do modo de tela na RAM de vídeo BASE pode ser avaliada ou usada como uma variável comum, mas deveria ser utilizada apenas em programas gráficos avançados. BASE é um número inteiro de 16 bits.

Sintaxe

BASE(<número>)

BASE(0)	Endereço inicial da tabela de nomes do modo de texto 0.
BASE(1)	Não utilizado.
BASE(2)	Endereço inicial do gerador de padrões para o modo de texto 0.
BASE(3)	Não utilizado.
BASE(4)	Não utilizado.
BASE(5)	Endereço inicial da tabela de nomes do modo de texto 1.
BASE(6)	Endereço inicial da tabela de cores do modo de texto 1.
BASE(7)	Endereço inicial do gerador de padrões para modo de texto 1.
BASE(8)	Endereço inicial dos atributos de sprite para o modo de texto 1.
BASE(9)	Endereço inicial do padrão de sprites para o modo de texto 1.
BASE(10)	Endereço inicial da tabela de nomes dos gráficos de alta resolução no modo de tela 2.
BASE(11)	Endereço inicial da tabela de cores dos gráficos de alta resolução no modo de tela 2.
BASE(12)	Endereço inicial do gerador de padrões dos gráficos de alta resolução no modo de tela 2.
BASE(13)	Endereço inicial da tabela de atributos de sprites dos gráficos de alta resolução no modo de tela 2.
BASE(14)	Endereço inicial da tabela de padrões de sprites dos gráficos de alta resolução no modo de tela 2.
BASE(15)	Endereço inicial da tabela de nomes de gráficos policromáticos no modo de tela 3.
BASE(16)	Não utilizado.

BASE(17) Endereço inicial da tabela de geração de padrão de gráficos policromáticos no modo de tela 3.

BASE(18) Endereço inicial da tabela de atributos de sprite de gráficos policromáticos no modo de tela 3.

BASE(19) Endereço inicial da tabela de padrões de sprite de gráficos policromáticos no modo de tela 3.

Exemplos

```
PRINT BASE(0)
0
```

Corrigindo erros

O argumento deve estar entre 0 e 19 e deve ser um inteiro. Nenhum outro número é reconhecido e resultará uma chamada para uma função ilegal (Illegal Function Call).

Comandos associados e referências

Parte 2, Capítulo 45 (Vol. 1): "Gráficos Avançados VI".

BEEP BEEP DO ALTO-FALANTE

Descrição

BEEP fornece exatamente o mesmo som que PRINT CHR$(7) durante 0,04 segundos.

Sintaxe

BEEP

Exemplos

BEEP

Pontos a serem lembrados

BEEP é igual a CHR$(7).

Comandos associados e referências

SOUND
PLAY

Parte 1, Capítulo 28 (Vol. 1): "A Macrolinguagem Musical".

BIN$ STRING BINÁRIA

Descrição

BIN$ devolve o equivalente binário de um argumento decimal na forma de uma string. Tem o alcance de –32768 até 65535 e o argumento tem de ser um inteiro, ou uma expressão numérica que resultará em um inteiro.

Sintaxe

BIN$(<const-num>)
BIN$(<var-num>)
BIN$(<exp-num>)

Exemplos

```
10 PRINT BIN$(255)
20 PRINT BIN$(9999)
```

```
RUN
11111111
10011100001111
```

Pontos a serem lembrados

Caso o argumento seja negativo, então será utilizada a forma complemento de dois, isto é, BIN$(–n)=BIN$(65536–n).

Para converter um número binário em decimal, utilize o prefixo &B para representar um número binário e o iguale a uma variável decimal como abaixo:

```
A%=&B00001111.
```

Corrigindo erros

O argumento não deve ser uma string ou um número real. Caso seja uma string ou um número real, resultará um erro "Type mismatch" (no HOT-BIT, a mensagem de erro será "Tipo Desigual").

Ocorrem erros de overflow quando o inteiro estiver fora da faixa de –32678 a 65535.

Comandos associados e referências

Parte 2, Capítulo 33 (Vol. 1): "Introdução aos Sistemas de Numéricos Usados no MSX".

BLOAD CARREGAR BYTES

Descrição

BLOAD é utilizado para carregar programas em linguagem de máquina e dados de um dispositivo especificado como um cassete, por exemplo. Tem a opção de executar automaticamente o programa em linguagem de máquina carregado, e também de carregar o programa em um endereço diferente daquele do qual foi gravado.

Sintaxe

BLOAD "<nome-dispositivo>:"

Para carregar arquivo com nome desconhecido.

BLOAD "<nome-dispositivo>:<nome>"

Para carregar um arquivo de nome conhecido.

BLOAD "<nome-dispositivo>:<nome>",R

Isto executa automaticamente o programa em linguagem de máquina recém-carregado. R representa RUN.

BLOAD "<nome-dispositivo>:<nome>",<const-num>

Quando a <const-num> é especificada, todos endereços especificados em BSAVE são deslocados neste valor.

BLOAD "<nome-dispositivo>:<nome>",R,<const-num>

Carrega do desvio e depois executa.

Exemplos

```
BLOAD "CAS:GAME",R
BLOAD "CAS:ADVENT",&HOOFF
```

Pontos a serem lembrados

Caso o endereço de execução seja omitido quando o código for gravado, o computador assume que o endereço de execução é o endereço inicial de quando o programa é carregado com a opção R.

A maioria dos jogos em linguagem de máquina gravada em cassete é auto-executada utilizando este comando.

O nome do arquivo deve ter no máximo 6 caracteres.

Corrigindo erros

Ocorre um erro de dispositivo de E/S ("Device I/O Error"), se o arquivo estiver mal gravado na fita.

Se o nome do arquivo estiver errado, obviamente não vai ser carregado no computador. Este é um erro comum. Se não estiver certo quanto ao nome do arquivo então omita-o, uma vez que o computador vai carregar os primeiros bytes em linguagem de máquina encontrados.

Comandos associados e referências

BSAVE

Parte 2, Capítulo 39 (Vol. 1): "Gravando no e Carregando do Cassete".

BSAVE GRAVAR BYTES

Descrição

Este comando grava um pedaço de memória em um cassete ou disco. Você tem de especificar o endereço inicial e o endereço final. Este comando é utilizado para gravar programas em linguagem de máquina e dados na forma de bytes. Você pode especificar também o endereço de execução, que será utilizado ao executar automaticamente o programa em linguagem de máquina ao carregar com BLOAD.

Sintaxe

```
BSAVE "<nome-dispositivo>:<nome>",<end.inicial>,<end.final>
BSAVE "<nome-dispositivo>:<nome>",<end.inicial>,<end.final>,<end.execução>
```

Exemplos

```
BSAVE "CAS:GAME",&HA100,&HA200,&HA1FF
BSAVE "CAS:ADVENT",&HC000,&HCFFF
```

Pontos a serem lembrados

O nome do arquivo deve ter no máximo 6 caracteres. Caso o endereço de execução seja omitido, então o computador presume que o endereço inicial é o endereço de execução quando o programa for carregado com a opção R.

Corrigindo erros

Uma gravação correta de memória somente será conseguida se o dispositivo de gravação for confiável. Se não conseguir gravar adequadamente, verifique o seu cassete.

Um nome de arquivo com mais de 6 dígitos provoca um Syntax Error (no HOT-BIT: Erro Sintaxe).

Comandos associados e referências

BLOAD

Parte 2, Capítulo 39 (Vol. 1): "Gravando no e Carregando do Cassete".

CALL CHAMA DECLARAÇÃO EXPANDIDA

Descrição

Este comando chama uma rotina contida em um cartucho ROM de expansão. A abreviação de call é "_", a marca de sublinhamento. CALL é dependente do cartucho ROM e portanto, ao ser utilizado em um programa BASIC, o programa perde a compatibilidade MSX.

Sintaxe

CALL <nome> (<lista de argumentos>)
_<nome> (<lista de argumentos>)

A lista de argumentos pode ser variada. Realmente depende do que fazem as declarações expandidas.

Exemplos

```
CALL SPEECH(altura%,voz%,"HELLO")
```

Pontos a serem lembrados

Alguns computadores utilizam CALL para executar rotinas em linguagem de máquina, mas este não é o caso do MSX. Para fazer isso você precisa utilizar USR.

O tamanho máximo do comando expandido é 15 caracteres. A não ser que você tenha um cartucho ROM de expansão em um dos slots da máquina, este comando não tem utilidade, pois o que CALL faz é totalmente dependente de um cartucho.

Corrigindo erros

Não utilize este comando para software comercial, para evitar que o programa perca a compatibilidade MSX.

Um nome de chamada inexistente resultará em um Syntax Error (no HOT-BIT: Erro Sintaxe).

Comandos associados e referências

Parte 2, Capítulo 50 (Vol. 1): "Gerenciamento de Memória do MSX e o Mecanismo dos Cartuchos".

CDBL CONVERTER EM UM NÚMERO DE PRECISÃO DUPLA

Descrição

A função CDBL converte números inteiros e de precisão simples, variáveis e expressões, em números de precisão dupla.

Sintaxe

```
CDBL(<const-num>)
CDBL(<var-num>)
CDBL(<exp-num>)
```

Exemplos

```
B#=CDBL(A!)
PRINT CDBL(NO%)
```

Pontos a serem lembrados

Os números de precisão dupla ocupam 8 bytes de memória e têm precisão de até 14 casas decimais.

Corrigindo erros

Ocorrerá um "Type mismatch" (no HOT-BIT: Tipo Desigual), caso seja utilizado um argumento string, ou se CDBL for equiparado a uma variável string.

Comandos associados e referências

CINT
CSNG

Parte 2, Capítulo 31 (Vol. 1): "Conversão de Tipos de Variáveis".

CHR$ STRING DE CARACTERES

Descrição

CHR$ gera um caractere a partir do número dado de acordo com o código ASCII. Isto é complementar à função ASC.

Uma vez que alguns códigos ASCII correspondem a caracteres de controle, como limpar o modo de tela, você pode ativá-los utilizando PRINT CHR$.

Sintaxe

CHR$(<const-num>)
CHR$(<var-num>)
CHR$(<exp-num>)

Todos estes devolvem uma string de caracteres.

Exemplos

PRINT CHR$(66) imprimirá um "B" no modo de tela.

```
10 CLS
20 C1=88: REM CODIGO ASCII DE X=88
30 C2=83: REM CODIGO ASCII DE S=83
40 C3=77: REM CODIGO ASCII DE M=77
50 PRINT CHR$(C3);CHR$(C2);CHR$(C1)
60 PRINT CHR$(C3+32);CHR$(C2+32);CHR$(C1+32)
```

Caso você processe o programa acima, verá que acrescentar 32 ao código ASCII das letras maiúsculas as transforma em letras minúsculas.

```
MSX
msx
```

Pontos a serem lembrados

Quando o número no interior do parênteses estiver entre 0 e 31, CHR$ vai agir de acordo com o código de controle conforme definido pelo padrão ASCII. Por exemplo, PRINT CHR$(7) resulta em um som do alto-falante. Verifique a tabela de códigos de controle. Verá que pode embutir códigos de controle em uma string.

Quando o número no interior do parênteses estiver entre 32 e 255, CHR$ vai dar-lhe o caractere ASCII correspondente.

Corrigindo erros

Caso o número no interior do parênteses for menor que 0 ou maior que 255, resultará um "Illegal function call" (no HOT-BIT: "Função Ilegal").

Lembre-se sempre de que CHR$ é uma string, e portanto não pode ser comparado a um número. Caso contrário, ocorrerá um "Type mismatch" (no HOT-BIT: "Tipo Desigual").

Comandos associados e referências

Veja o apêndice da tabela de códigos de caracteres ASCII.

Parte 1, Capítulo 20 (Vol. 1): "O Código ASCII".

CINT CONVERSÃO EM INTEIRO

Descrição

Esta função converte números de precisão simples e dupla, variáveis e expressões, em números inteiros. Tem um alcance de –32768 até 32767. Qualquer valor fora desta faixa criará problemas.

As partes fracionárias de um número real serão truncadas.

Sintaxe

```
CINT(<const-num>)
CINT(<var-num>)
CINT(<exp-num>)
```

Exemplos

```
A%=CINT(B!*C#)
PRINT CINT(1234.56789) fornecerá 1234
```

Pontos a serem lembrados

Os números inteiros ocupam 2 bytes de memória.

Corrigindo erros

O argumento tem de estar rigorosamente entre –32678 e 32767. Caso não esteja, resultará um "Overflow". Ocorrerá um "Type mismatch" (no HOT-BIT: "Tipo Desigual"), caso seja utilizado um argumento em string.

Comandos associados e referências

CDBL
CSNG
INT
FIX

Parte 2, Capítulo 31 (Vol. 1): "Conversão de Tipos de Variáveis".

CIRCLE DESENHA UM CÍRCULO

Descrição

CIRCLE desenha um círculo inteiro, ou parte de um círculo ou elipse no modo gráfico. Você pode especificar a posição do centro do círculo, a cor, e mesmo quanto do círculo você quer que o computador desenhe.

Sintaxe

CIRCLE <espec-coorden>,<raio>,<cor>,<ângulo inicial>,<ângulo final>,
<relação entre os raios>

Todos podem ser <const-num>,<var-num> ou <exp-num> mas nas suas faixas permitidas (veja a seguir).

Explicação de <espec-coorden>.

Existem duas maneiras de especificar as coordenadas do centro, uma relativa e outra absoluta.

<espec-coorden> pode ser um dos seguintes:

1. (<coordenada-x>,<coordenada-y>)
   Estas coordenadas especificam a posição absoluta do centro do círculo no modo de tela.

2. STEP(<deslocamento-x>,<deslocamento-y>)
   Estas coordenadas, <deslocamento-x>,<deslocamento-y>), são relativas ao último ponto referido pelo computador. Caso o deslocamento resultante estiver fora do modo de tela, o computador desenhará até o limite do modo de tela.

<cor>,<ângulo inicial>,<ângulo final>, e <relação entre os raios> são opcionais.

Alcance:

<coordenada-x>	–32768 até 32767: para colocar o centro no modo de tela utilize a faixa de 0 até 255.
<coordenada-y>	–32768 até 32767: para colocar o centro no modo de tela utilize a faixa de 0 até 191.
<raio>	0 até 32767 mas o círculo não cabe no modo de tela se <raio> for muito grande.
<cor>	0 até 15. (Veja a seção de código de cores em COLOR.)
<ângulo inicial>	0 até 2*PI em radianos.
<ângulo final>	0 até 2*PI em radianos.

Um prefixo "_" para o <ângulo inicial> e o <ângulo final> especifica uma linha que deve ser desenhada do centro até dada extremidade.

<relação entre os raios>	0 até 32767.
<relação entre os raios>	=(raio vertical)/(raio horizontal)

Caso a proporção seja maior que 1, a elipse será alongada verticalmente.

Caso a proporção seja menor que 1, a elipse será alongada horizontalmente.

Quando a proporção não for especificada, o computador assume o valor 1 e desenha um círculo.

Caso <proporção> seja muito grande, o máximo que você consegue será uma linha vertical.

Se 0, obterá uma linha horizontal.

Exemplos

```
10 SCREEN 2
20 CIRCLE (100,100),50,8
30 FOR I=1 TO 5
40 CIRCLE STEP (I*5,0),50,8
50 NEXT
60 GOTO 60:REM ISTO MANTEM O SCREEN 2
```

Pontos a serem lembrados

Este comando funciona apenas no modo gráfico, de modo que esteja certo de ter selecionado SCREEN 2 ou SCREEN 3.

Caso a cor não esteja especificada, o computador desenhará na cor atual do texto.

Os ângulos inicial e final estão em radianos.

Corrigindo erros

Ocorre um erro do tipo "Overflow" quando o valor do <espec-coordenada> ou <raio> estiverem fora da faixa.

Ocorre um erro do tipo "Illegal function call" (no HOT-BIT: "Função Ilegal"), caso o número do código de cor estiver errado ou

Um erro do tipo "Illegal function call" (no HOT-BIT: "Função Ilegal"), quando <ângulo inicial> ou <ângulo final> estiver fora da faixa.

Comandos Associados e Referências

COLOR
SCREEN

Parte 1, Capítulo 25 (Vol. 1): "Desenhando Círculos e Elipses".

CLEAR LIMPAR ÁREA DE MEMÓRIA

Descrição

CLEAR informa ao computador para que apague todas as variáveis atualmente residentes na memória. Este comando limpa o espaço de memória reservado para variáveis anulando todas as variáveis string e numéricas.

Com este comando, você pode também reservar espaço de memória para variáveis string e linguagem de máquina na área de trabalho do sistema.

Você pode reservar espaço de memória pela alteração da posição de memória mais elevada disponível para ser utilizada em BASIC. A posição de memória mais elevada em default é &HF380. Você pode abaixar isto utilizando CLEAR,<mem-alta>. O espaço de memória criado pode ser utilizado para armazenar programas em linguagem de máquina e dados.

Sintaxe

CLEAR<espaço string>	Estabelece o tamanho do espaço string.
CLEAR,<mem-alta>	Estabelece a posição mais alta de memória disponível ao BASIC.
CLEAR<espaço string>,<mem-alta>	Estabelece o tamanho do espaço string, bem como a posição mais alta de memória disponível ao BASIC.

Exemplos

CLEAR	
CLEAR 255	Expande a área de trabalho string até 255 bytes.
CLEAR,&HF300	Cria espaço para o programa em código de máquina do usuário entre &HF300 e &HF380.

Pontos a serem lembrados

O espaço de string inicial é 200 bytes. Isto limita o comprimento total de variáveis string em 200 caracteres. Se você quer que o computador trate com variáveis string maiores que 200 caracteres, primeiro terá de executar CLEAR 255, por exemplo.

A área de memória entre &HFFFF e &HF380 será utilizada pela ROM básica como espaço de trabalho do sistema.

Comandos associados e referências

Parte 1, Capítulo 1 (Vol. 1): "Comandos Úteis e Dicas para Escrever Programas".

Parte 2, Capítulo 30 (Vol. 1): "Constantes e Variáveis".

Parte 2, Capítulo 48 (Vol. 1): "Mapa da Memória".

CLOAD/CLOAD? CARREGAR DO CASSETE

Descrição

Utilize CLOAD para carregar no computador um programa BASIC de uma fita cassete. O computador vai informar-lhe se encontrou um programa e se está pulando o programa ou carregando-o. Você não precisa conhecer o nome do programa pois, caso este não seja especificado, o CLOAD vai carregar o primeiro programa BASIC encontrado.

CLOAD? compara o programa BASIC carregado do cassete com aquele na memória do computador. O computador vai dar um OK, ou um erro de verificação se o programa atual for diferente daquele em fita.

Sintaxe

```
CLOAD
CLOAD "<nome>"
CLOAD?
CLOAD? "<nome>"
```

Exemplos

Vamos dizer que há dois programas denominados "SNAKE" e "LADDERS" no cassete, se você digitar:

```
CLOAD <return>
```

```
CLOAD
Found : SNAKE
OK
```

No HOT-BIT, em vez de FOUND, a mensagem é ACHEI.

Tente CLOAD "LADDERS" <return>

```
CLOAD LADDERS
Skip  : SNAKE
Found : LADDERS
OK
```

No HOT-BIT em vez de SKIP a mensagem é PULEI.

Pontos a serem lembrados

O nome do programa não deve ter mais que 6 caracteres. Quando um programa é carregado com CLOAD, qualquer programa anterior da memória será apagado.

A tecla de função F7 é CLOAD".

A freqüência em BAUD de armazenamento pode ser tanto 1200 BAUD ou 2400 BAUD, mas CLOAD automaticamente vai perceber em que razão a gravação foi feita e fazer o carregamento de acordo.

Caso haja um arquivo aberto (utilizando a função OPEN), então CLOAD automaticamente fechará aquele arquivo e prosseguirá.

Corrigindo erros

Ocorre um "Device I/O error" (no HOT-BIT: Erro/Periférico) quando você tentar carregar um programa mal gravado. Utilize sempre um cassete confiável.

Caso o nome do arquivo esteja errado, obviamente nada será carregado. Cuidado ao soletrar o nome.

Comandos associados e referências

CSAVE

Parte 1, Capítulo 11 (Vol. 1): "Gravando o seu Programa em uma Fita".

CLOSE FECHAR O CANAL

Descrição

Este comando é utilizado para fechar um canal aberto e é o oposto ao comando OPEN. O buffer associado com o canal especificado também é liberado.

Sintaxe

CLOSE ... fechar todos os canais.
CLOSE # <número do arquivo>,<número do arquivo>

Exemplos

```
CLOSE# 2
```

Corrigindo erros

Resultará um erro do tipo "Bad file number" (no HOT-BIT: Nº de Arquivo) (Número de Arquivo Errado) quando você coloca um número de arquivo não-existente no argumento.

Comandos associados e referências

OPEN

Parte 2, Capítulo 47 (Vol. 1): "Como Usar Arquivos".

CLS — LIMPAR O MODO DE TELA

Descrição

CLS limpa qualquer modo de tela.

No modo gráfico, a tela será limpa à cor de fundo atual.

Sintaxe

```
CLS
```

Exemplos

O que segue limpa o modo de tela para vermelho-claro:

```
10 SCREEN 2
20 COLOR ,9
30 CLS
40 GOTO 40
```

LINHA 10 ESCOLHE MODO GRÁFICO DE ALTA RESOLUÇÃO.
LINHA 20 COLOCA A COR DO FUNDO EM 9 (VERMELHO-CLARO).
LINHA 30 LIMPA O MODO DE TELA.
LINHA 40 FICA NESTE MODO DE TELA.

>Resultado: Um fundo de tela em vermelho-claro.

Pontos a serem lembrados

O cursor de texto será recolocado no canto superior esquerdo. Quando as teclas de função estiverem no modo KEY ON, então o modo de tela de texto continuará apresentando a lista das teclas de função na parte inferior do modo de tela depois que ele for limpo com CLS. Para nos livrarmos da lista de teclas teremos de executar KEY OFF.

CONTROL L: <CTRL> <L> tem a mesma função que CLS.
PRINT CHR$(12) tem a mesma função que CLS, isto é, você pode embutir um CLS em uma string.

Comandos associados e referências

COLOR

Parte 1, Capítulo 3 (Vol. 1): "Escrevendo um Programa".

COLOR COR

Descrição

Este comando estabelece as cores do fundo, do texto e da margem. Existem 15 cores e o transparente.

Sintaxe

COLOR <cor do texto>,<cor do fundo>,<cor da margem>

Todos podem ser <exp-num> mas têm de estar entre 0 e 15. Também podem ser opcionais de modo que alguns argumentos poderão estar faltando.

Código de Cores

0 transparente	8 vermelho-médio
1 preto	9 vermelho-claro
2 verde-médio	10 amarelo-escuro
3 verde-claro	11 amarelo-claro
4 azul-escuro	12 verde-escuro
5 azul-médio	13 roxo
6 vermelho-escuro	14 cinza
7 azul-claro	15 branco

Exemplos

COLOR 11	Estabelece para o texto um amarelo-claro.
COLOR 15..15	Estabelece a cor branca para o texto e para a margem.
COLOR 1.5.1	Estabelece um azul-claro para o fundo e preto para o texto e margem.

Pontos a serem lembrados

Quando o computador é ligado, COLOR é estabelecido em COLOR 15,3,7, isto é, um texto branco, um fundo azul e margens azul-claro.

A tecla de função 1 é predefinida como o comando "COLOR" quando o computador é ligado. Também F6 é "COLOR 15,4,7", que é a cor default.

A cor de fundo não se altera a não ser que o comando CLS seja executado.

Caso a cor não seja especificada nem em PSET, ou LINE, ou CIRCLE, então estes comandos gráficos desenham de acordo com COLOR.

Corrigindo erros

Os argumentos têm de estar entre 0 e 15; qualquer outro valor provocará um erro do tipo "Illegal function call", ou erro de chamada ilegal.

Comandos associados e referências

DRAW
PRESET
PSET
LINE
CIRCLE
CL$

Parte 1, Capítulo 22 (Vol. 1): "Cores".
Parte 2, Capítulo 41 (Vol. 1): "Gráficos Avançados II"

CONT — CONTINUE

Descrição

Este comando informa ao computador para continuar a execução de um programa a partir do ponto em que parou devido a um STOP ou <CTRL> <STOP>. O

computador lembra a última linha executada antes do <CTRL> <STOP> e continua a partir do próximo comando. Caso a interrupção tenha ocorrido no meio de um comando INPUT, o computador continua a partir do INPUT.

Sintaxe

```
CONT
```

Exemplos

```
10 PRINT "DE-ME O DIA,MES E ANO DE SEU NASCIMENTO"
20 INPUT DIA,MES,ANO
30 PRINT "OBRIGADO"

DE-ME O DIA,MES E ANO DE SEU NASCIMENTO

? 23,06,61
OBRIGADO
```

Pontos a serem lembrados

CONT operará nos seguintes casos:

1. Depois que o programa for interrompido por um STOP. O programa continuará a partir da próxima linha.
2. Depois que o programa for interrompido por um END. O programa continuará a partir da próxima linha.
3. Depois que o programa for interrompido por uma combinação <CTRL> <STOP>. O programa continuará a partir da próxima linha.

Corrigindo erros

CONT não funciona nos seguintes casos:

1. Depois que o programa tiver sido editado.
2. Depois de parar a impressora.
3. Depois de interromper os comandos de entrada/saída do cassete como INPUT#.

A mensagem de erro resultante dos casos acima será "Cant' Continue" (no HOT-BIT: Não Contínuo!).

Comandos associados e referências

Parte 1, Capítulo 7 (Vol. 1): "Comandos Úteis e Dicas para Escrever Programas".

COS COSSENO

Descrição

COS devolve o cosseno de um ângulo.

COS é calculado com precisão dupla.

O argumento, isto é, o ângulo, tem de ser em radianos. O MSX não reconhece graus.

Sintaxe

```
COS(<const-num>)
COS(<var-num>)
COS(<exp-num>)
```

Exemplo

```
PRINT COS(1.4445432)
.1259179846524
```

Pontos a serem lembrados

COS retorna com precisão dupla.

O argumento tem de ser em radianos.

Corrigindo erros

Esta é uma função trigonométrica, e portanto não se mistura com strings.

Um erro do tipo "Type mismatch" (no HOT-BIT: Tipo Desigual) ocorre se você utilizar, por exemplo, COS(A$).

Comandos associados e referências

SIN
TAN
ATN

Parte 1, Capítulo 19 (Vol. 1): "Funções Matemáticas".

CSAVE GRAVAR EM CASSETE

Descrição

CSAVE é utilizado para gravar um programa em BASIC, que está na memória do computador, em uma fita cassete. O programa gravado precisa ter nome de arquivo de até 6 caracteres. CSAVE grava o programa em BASIC em códigos em vez de um arquivo ASCII, para ganhar tempo. Pode ser especificado um valor em baud de 1200 ou 2400. Caso o valor em baud não seja especificado, o computador gravará em 1200 baud.

Sintaxe

CSAVE"<nome>"
CSAVE"<nome>,<valor em baud>
<valor em baud>
1= 1200 baud.
2= 2400 baud.

Exemplos

```
CSAVE "ADVENT"
CSAVE "games", 2
```

Pontos a serem lembrados

O nome do programa tem de ter no máximo 6 caracteres.

O valor em baud pode ser colocado também com o comando SCREEN. Qualquer programa gravado com CSAVE não pode ser "juntado" com o comando MERGE, porque o programa é gravado em códigos. Para fundir programas você tem de gravar o programa utilizando o comando SAVE que grava os programas BASIC em um arquivo ASCII.

Corrigindo erros

O sucesso de CSAVE realmente depende da fita cassete e da confiabilidade do cassete.

Uma gravação mal feita tem como resultado um erro do tipo "DEVICE I/O" (no Hot-Bit: Erro Periférico) ao tentar carregar o programa utilizando a declaração CLOAD.

Comandos associados e referências

CLOAD
SAVE

Parte 1, Capítulo 11 (Vol. 1): "Gravando o seu Programa em uma Fita".

CSNG CONVERTER PARA UM NÚMERO DE PRECISÃO SIMPLES

Descrição

Esta função converte o argumento em um número de precisão simples.

Sintaxe

CSNG(<const-num>)
CSNG(<var-num>)
CSNG(<exp-num>)

Exemplos

```
PRINT COS(0.7656)
    .72096670541357
PRINT CSNG(COS(0.7656))
    .720967
```

Pontos a serem lembrados

CSNG arredonda na sexta casa decimal mais próxima.

Corrigindo erros

Resulta um erro do tipo "Type mismatch" (no Hot-Bit: Tipo Desigual) se o argumento é uma string.

Comandos associados e referências

CDBL
CINT

Parte 2, Capítulo 31 (Vol. 1): "Conversão de Tipos de Variáveis".

CSRLIN POSIÇÃO DO CURSOR NA LINHA

Descrição

Esta variável de sistema devolve a posição do cursor de texto.

Sintaxe

CSRLIN

Exemplos

PRINT CSRLIN

Pontos a serem lembrados

CRSLIN é 0 para a linha superior.

Corrigindo erros

Você não pode estabelecer esta variável de sistema. Caso você tente, ocorrerá um "Syntax Error" (Erro de Sintaxe).

Comandos associados e referências

POS(0)

Parte 2, Capítulo 40 (Vol. 1): "Gráficos Avançados I".

DATA

Descrição

Nos comandos DATA você pode armazenar os dados requeridos no seu programa BASIC, que depois serão acessados por meio de READ. Você pode armazenar

qualquer número de strings ou números que caibam em uma linha de programa. Estes itens de dados têm de ser separados por vírgulas. A posição dos comandos DATA não é importante, uma vez que são não-executáveis, isto é, o computador ignora um comando DATA ao encontrar um; portanto, podem ser colocados em qualquer ponto do programa.

O comando READ é utilizado para acessar, seqüencialmente, os itens mantidos no comando DATA. O computador se lembra quais os itens nos comandos DATA que já foram utilizados. O próximo comando READ vai começar a partir do próximo item não lido no comando.

Os itens contidos em um comando DATA podem ser números ou strings. O comprimento da lista, porém, tem de ser menor que 255 caracteres para caber em uma linha. As constantes numéricas podem ser inteiros, número de precisão simples ou dupla. A constante string não precisa ser envolvida por " ", a não ser que a string contenha vírgulas, dois-pontos, ponto-e-vírgula etc. Não é possível armazenar variáveis ou expressões em um comando DATA.

A variável correspondente no comando READ tem de ser do mesmo tipo que o item de dados, isto é, uma variável string para um dado string.

Sintaxe

DATA <lista de constantes>

Exemplos

```
10 CLS
20 PRINT "LISTA DE TELEFONES"
30 PRINT
40 FOR I=1 TO 4
50 READ NOME$,TEL$
60 PRINT NOME$,TEL$
70 NEXT I
80 DATA TOM,123456,PAULO,777777
90 DATA BELLO,987654,ZE,100001
```

```
LISTA DE TELEFONES

TOM             123456
PAULO           777777
BELLO           987654
ZE              100001
```

Pontos a serem lembrados

Aspas em torno de dados string são requeridos apenas nos seguintes casos:

1. Se os dados string contêm vírgulas.
2. Se os dados string contêm dois-pontos.
3. Se os dados string contêm espaços antes ou depois dos caracteres.

Para apontar para um determinado comando DATA, utilize o comando RESTORE para colocar o computador em READ dos dados na linha referida pelo RESTORE.

Corrigindo erros

Se o tipo de caractere na variável READ e no item DATA estão discordantes, resultará um erro do tipo "Type mismatch" (no Hot-Bit: Tipo Desigual). Portanto, variáveis string têm de ler dados string e variáveis numéricas, números.

Caso o computador fique sem DATA para o READ, ocorrerá um erro do tipo "Out of DATA" (no Hot-Bit: Sem DATA).

Comandos associados e referências

READ
RESTORE

Parte 1, Capítulo 12 (Vol. 1): "Lendo Dados em Matrizes".

DEFDBL DEFINIR PRECISÃO DUPLA

Descrição

O comando DEFDBL declara que qualquer nome de variável começando com a faixa especificada de caracteres deve ser tratado como um número com precisão dupla.

Sintaxe

DEFDBL <faixa de letras>

Exemplos

5 DEFDBL A, B, C, ... precisão dupla para variáveis iniciando com A, B e C.

Pontos a serem lembrados

O comando do tipo de variável com #,$,% e ! tem prioridade sobre DEFDBL.

Comandos associados e referências

DEFINT
DEFSNG
DEFSTR

Parte 2, Capítulo 30 (Vol. 1): "Constantes e Variáveis".

DEF FN DEFINE UMA FUNÇÃO

Descrição

Utilizando DEF FN você pode definir a sua própria função, que pode ser chamada pelo nome em qualquer lugar do programa. É destinado a economizar memória com expressões que aparecem mais que uma vez em um programa.

Você pode dar à sua função o nome que desejar, desde que não seja uma palavra reservada.

O comando DEF EN tem de ser executado para que o computador saiba que a função existe, antes dela ser chamada.

Você pode colocar parâmetros na função para serem processados. Qualquer variável no interior da função é local, isto é, o seu valor permanece inalterado fora da função.

Para chamar uma função, você tem de colocar FN em frente ao nome da função.

O comando DEF FN está limitado a uma linha.

Sintaxe

DEF FN <nome> = <expressão>
DEF FN <nome> (<parâmetros>) = <expressão>

<nome> tem de ter um sufixo "$" para uma função string. Outros tipos de especificadores podem ser utilizados, se necessário.

Exemplos

```
10 CLS
20 DEF FNCUBO(X)=X^3
30 INPUT "DIGITE UM NUMERO";A%
40 PRINT "O CUBO DE ";A%;"E";FNCUBO(A%)
50 B%=FNCUBO(A%+1)
60 PRINT "O CUBO DE ";A%+1;"E";B%
70 C%=FNCUBO(FNCUBO(A%))
80 PRINT "O CUBO DE ";FNCUBO(A%);"E";C%
```

```
DIGITE UM NUMERO? 2
O CUBO DE  2 E 8
O CUBO DE  3 E 27
O CUBO DE  8 E 512
```

Pontos a serem lembrados

Os itens na lista de parâmetros, ao definir a função, poderão ser numéricos, string, expressões ou variáveis, mas têm de ser separados por vírgulas.

Você pode incluir também variáveis externas à função. Você pode estabelecer uma função de nível pela utilização de uma função no parâmetro como no exemplo anterior.

Corrigindo erros

A maioria dos erros ocorre com DEF FN quando você chama a função. Aqui estão alguns deles:

1. Função de usuário não definida ("Undefined user function"): se uma função é chamada antes que DEF FN tenha sido executado.
2. Caso o tipo de variáveis não confira não haverá concordância entre a variável estabelecida para a função e o resultado da função, resultando um "Type Mismatch" (Tipo Desigual), por exemplo, A$=FNCUBO(2).
3. Caso não haja um número suficiente de parâmetros ao chamar a função, resultará um erro.
4. "Type Mismatch" (Tipo Desigual) resultará caso os tipos de parâmetros não coincidam.

Comandos associados e referências

Parte 1, Capítulo 16 (Vol. 1): "Definindo as suas Próprias Funções".

DEFINT DEFINIR UM INTEIRO

Descrição

O comando DEFINT declara que qualquer nome de variável iniciando entre a faixa especificada de caracteres deve ser tratado como inteiro.

Sintaxe

DEFINT <faixa(s) de letras>

Exemplo

10 DEFINT I, J, K. As variáveis iniciando com I, J e K devem ser inteiros.

Pontos a serem lembrados

O comando do tipo de variável com #,%,$ e ! tem prioridade sobre DEFINT.

Comandos associados e referências

DEFDBL
DEFSNG
DEFSTR

Parte 2, Capítulo 30 (Vol. 1): "Constantes e Variáveis".

DEFSNG DEFINIR PRECISÃO SIMPLES

Descrição

O comando DEFSNG declara que quaisquer nomes de variáveis iniciando com os caracteres na faixa especificada devem ser tratados como números de precisão simples.

Sintaxe

DEFSNG <faixa(s) de letras>

Exemplo

10 DEFSNG X, Y, Z ... Precisão simples para as variáveis iniciando com X, Y ou Z.

Pontos a serem lembrados

O comando do tipo de variável com #,$,% e ! tem prioridade sobre DEFSNG.

Comandos de associados e referências

DEFDBL
DEFINT
DEFSTR

Parte 2, Capítulo 30 (Vol. 1): "Constantes e Variáveis".

DEFSTR DEFINIR UMA STRING

Descrição

O comando DEFSTR afirma que quaisquer nomes de variáveis iniciando com os caracteres na faixa especificada devem ser tratados como variáveis string.

Sintaxe

DEFSTR <faixa(s) de letras>

Exemplos

DEFSTR E,R,T

Pontos a serem lembrados

A declaração do tipo de variável com #,$,% e ! tem prioridade sobre DEFSTR.

Comandos associados e referências

DEFDBL
DEFINT
DEFSNG

Parte 2, Capítulo 30 (Vol. 1): "Constantes e Variáveis".

DEF USR DEFINIR O ENDEREÇO DE ROTINA EM CÓDIGO DE MÁQUINA DO USUÁRIO

Descrição

Para utilizar sub-rotinas em linguagem de máquina a partir do BASIC, utilize a função USR. A função USR chama um programa em linguagem de máquina no endereço especificado do comando DEF USR. Você pode definir até 10 funções USR. Cada USR deve ser seguido por um número, que é 5 no exemplo a seguir:

```
DEF USR5=&HFF80
```

Quando você quer chamar a sub-rotina em linguagem de máquina definida por DEFUSR, utilize a função USR, isto é:

```
PRINT USR5("parâmetro")
```

onde "parâmetro" entre parênteses é o parâmetro que deve ser passado do BASIC ao programa em linguagem de máquina.

Sintaxe

DEFUSR=<endereço inicial> ... quando [dígito] = 0
DEFUSR [dígito] = <endereço inicial>
<endereço inicial> poderá ser <const-num>,<var-num>, ou <exp-num> mas tem de ser um inteiro.
[dígito] tem de estar entre 0 e 9.

Exemplos

```
DEFUSR5=&HF300
DEFUSR=&HF300+255
Para chamar USR5
10  DUMMY=USR5(9):
```

(9) é o parâmetro que deve ser passado para o programa em linguagem de máquina.

Pontos a serem lembrados

Você pode ter até dez chamadas USR ao mesmo tempo, mas elas podem ser redefinidas de modo que você pode ter quantas rotinas em linguagem de máquinas quantas sejam necessárias.

DEFUSR é o mesmo que DEFUSRO.

Você tem de conhecer o código de máquina do Z80 relativamente bem para utilizar DEFUSR e USR.

Corrigindo erros

O endereço inicial tem de estar na RAM e tem de ser um inteiro. Qualquer outra coisa causará um erro.

Comandos associados e referências

Parte 2, Capítulo 49 (Vol. 1): "O Comando USR e a Linguagem de Máquina".

DELETE

Descrição

Este é um comando de edição que apaga parte do programa.

Sintaxe

DELETE <linha>
DELETE <linha> – <linha>
DELETE – <linha>

Exemplos

DELETE 20	Apaga a linha 20.
DELETE 20 – 100	Apaga as linhas 20 até 100 inclusive.
DELETE – 100	Apaga até a linha 100 inclusive.

Pontos a serem lembrados

Ao apagar apenas uma linha, é mais fácil introduzir o número da linha e depois <RETURN> do que utilizar o comando DELETE.

O comando DELETE não pode ser utilizado em uma linha de programa.

Corrigindo erros

Ocorrerá um erro caso você aponte para um número de linha não-existente.

Caso você apague parte de um programa por engano, evidentemente o estragará. Assegure-se de não apagar linhas necessárias.

Comandos associados e referências

Parte 1, Capítulo 7 (Vol. 1): "Comandos Úteis e Dicas para Escrever Programas".

Parte 2, Capítulo 29 (Vol. 1): "Edição Avançada de Programas".

DIM DIMENSIONANDO MATRIZES

Descrição

O comando DIM é utilizado para definir matrizes. Uma matriz é um grupo inteiro de variáveis que tem todos o mesmo nome, mas diferem entre si por um parâmetro numérico.

Sintaxe

DIM <variável> (<const-num>)
DIM <variável> (<const-num> , <const-num>...)

<variável> é o nome da matriz definida e pode ser um inteiro, precisão simples, precisão dupla ou uma string. <const-num> é o tamanho da matriz e deve ser um inteiro positivo. Para definir uma matriz multidimensional, coloque mais que uma <const-num> entre chaves.

Exemplos

DIM A(5, 5, 5) é uma matriz tridimensional.
DIM B$(100), C$(100, 2), D%(100), E!(100), F#(100).

Você pode definir qualquer tipo de matriz e quantas desejar em um comando DIM.

Pontos a serem lembrados

Você não precisa definir nenhuma matriz unidimensional que tenha um número de elementos menor que dez, isto é, DIMA%(5) não será necessário!

O valor mínimo de elemento de uma matriz é o 0. Portanto, uma matriz começa pelo elemento de ordem zero.

DIMA(5) fornece A(0), A(1), A(2), A(3), A(4), A(5), isto é, 6 elementos.

Quando uma matriz é inicialmente definida por um comando DIM, todos os elementos são colocados em zero.

Corrigindo erros

Ocorrerá um erro, caso você não fique nos limites definidos no comando DIM.

Comandos associados e referências

Parte 1, Capítulo 12 (Vol. 1): "Lendo Dados em Matrizes".
Parte 1, Capítulo 13 (Vol. 1): "Manuseio de Dados e sua Ordenação".
Parte 2, Capítulo 30 (Vol. 1): "Constantes e Variáveis".

DRAW

Descrição

Este comando permite que você faça desenhos de acordo com a Macrolinguagem Gráfica (MLG). A macrolinguagem gráfica é escrita na forma de strings, e os comandos DRAW informam ao computador para desenhar de acordo com esta sublinguagem.

Você deve se lembrar de que há um cursor gráfico que pode se movimentar em qualquer lugar da tela. A macrolinguagem gráfica controla os movimentos do cursor gráfico e suas ações de acordo com a maneira que você programa.

Sintaxe

DRAW "<string>"
DRAW <var-string>
DRAW <exp-string>

Os comandos da macrolinguagem gráfica: para cima, para baixo, esquerda e direita.

U<n> ↑ : desenha para cima.
D<n> ↓ : desenha para baixo.
L<n> ← : desenha para a esquerda.
R<n> → : desenha para a direita.

<n>, em cada caso, é a distância que deve ser movimentada. Isto entretanto depende do fator de escala S (veja em seguida).

Como desenhar diagonalmente:

E<n> ↗ : desenha diagonalmente para cima e para a direita.
F<n> ↘ : desenha diagonalmente para baixo e para a direita.
G<n> ↙ : desenha diagonalmente para baixo e para a esquerda.
H<n> ↖ : desenha diagonalmente para cima e para a esquerda.

<n>, neste caso, é ligeiramente diferente do que para o desenho horizontal e vertical. E <n>, por exemplo, desenha para cima <n> ponto e para direita <n> coordenadas pontos, e não um comprimento <n> diagonalmente.

Como desenhar coordenadas específicas:

Para desenhar as coordenadas específicas no modo de tela, utilize o comando M que movimenta o cursor gráfico de sua última posição às coordenadas x e y enquanto desenha.

M<x> , <y> : desenha até as coordenadas x e y.

Você pode fazer um desenho relativo ao último ponto desenhado pela utilização dos prefixos + ou – às coordenadas x e y.

M+<x> , <y> : M – <x> , <y>
M+<x> , – <y> : M – <x> , – <y>

Como se movimentar sem desenhar (comando vazio):

Existem momentos em que você não quer que o computador faça uma linha, como quando estiver movendo o cursor gráfico até uma posição específica. Utilize o prefixo B que representa vazio (Blank). Ele vai interromper o desenho do computador enquanto move o cursor gráfico.

Prefixo B para todos comandos gráficos anteriores:

B : movimentação sem desenhar.

As seguintes combinações podem ser utilizadas:

BU<n> : move para cima.
BD<n> : move para baixo.
BL<n> : move para a esquerda.
BR<n> : move para a direita.
BE<n> : move diagonalmente para cima e para a direita.
BF<n> : move diagonalmente para baixo e para a direita.
BG<n> : move diagonalmente para baixo e para a esquerda.
BH<n> : move diagonalmente para cima e para a esquerda.
BM<x> , <y> : move de forma absoluta para apontar as coordenadas x e y.
BM+<x> , <y> : BM – <x> , <y> : move relativamente.
BM+<x> , – <y> : BM – <x> , – <y> : com passo x e y.

Como voltar à posição de partida, depois de desenhar uma linha:

Caso queira voltar ao ponto de partida depois de desenhar uma linha, utilize o prefixo N. Este vai colocá-lo de volta às coordenadas em que começou.

Prefixo N : desenha mas retorna o cursor à posição inicial.

Podem ser utilizadas as seguintes combinações:

NU<n>	: desenha para cima, depois volta.
ND<n>	: desenha para baixo, depois volta.
NL<n>	: desenha para a esquerda, depois volta.
NR<n>	: desenha para a direita, depois volta.
NE<n>	: desenha diagonalmente para cima e para a direita, depois volta.
NF<n>	: desenha diagonalmente para baixo e para a direita, depois volta.
NG<n>	: desenha diagonalmente para baixo e para a esquerda, depois volta.
NH<n>	: desenha diagonalmente para cima e para a esquerda, depois volta.
NM<x> , <y>	: desenha de forma absoluta ao ponto de coordenadas x, y, depois volta.
NM+ <x> , <y>	: NM – <x>, <y>: movimento relativo em passo x e y.
NM+ <x> , <y>	: NM <x> , – <y> : depois volta.

Como girar a direção do movimento relativo:

Utilizando o comando de ângulo A, você pode girar o eixo dos comandos de movimento relativo como U, D, L, R, F, E, G e H.

Comando de ângulo A:

A<n>	: onde <n> pode ser 0, 1, 2 ou 3.
A0	: em 0 graus.
A1	: gira 90 graus em sentido anti-horário.
A2	: gira 180 graus em sentido anti-horário.
A3	: gira 270 graus em sentido anti-horário.

Como colocar cores na Macrolinguagem Gráfica:

Você pode utilizar o comando C para alterar a cor na MLG. Você pode alterar a cor do desenho com a freqüência que desejar com uma única declaração DRAW.

O prefixo C do comando colorido:

C<n> : estabelece a cor durante o desenho.
<n> : é um inteiro entre 0 e 15.

C0	transparente	C8	vermelho-médio
C1	preto	C9	vermelho-claro
C2	verde-médio	C10	amarelo-escuro
C3	verde-claro	C11	amarelo-claro
C4	azul-escuro	C12	verde-escuro
C5	azul-médio	C13	roxo
C6	vermelho-escuro	C14	cinza
C7	azul-claro	C15	branco

Como alterar a escala do desenho:

Prefixo do comando de escala S.

S<n> : onde <n> pode ser um inteiro entre 0 e 255.
fator de escala = <n> / 4.

Portanto, S1 desenha 1/4 do comprimento especificado por U, D, L, R etc.

S4 e S0 são iguais e têm como resultado um desenho sem escala.

S8 vai estabelecer a escala ao dobro do valor default (S4).

Como utilizar substrings:

X <var-string>;

significa executar o que estiver dentro da variável string. Na macrolinguagem gráfica você pode ter uma string, que contém o comando de desenhar dentro de uma string. O ponto-e-vírgula será sempre necessário.

DRAW "X <var-string> ;"

Como utilizar variáveis numéricas na MLG:

Em todos os comandos da MLG, <n>, <x> e <y> podem ser variáveis, mas têm de ser prefixados com um sinal "=" e requerem um ponto-e-vírgula no final, isto é:

```
= <var-num>      = <n> ;
                 = <x> ;
                 = <y> ;
```

Exemplo da variável numérica G%

```
DRAW "U=G%;"
```

Exemplos

```
10 SCREEN 2
20 A$="R50U50L50D50"
30 DRAW "BM100,150"
40 DRAW "S0XA$;"
50 IF INKEY$="" THEN 50
60 FOR I=1 TO 10
70 DRAW "S=I;XA$;"
80 NEXT I
90 GOTO 90
```

LINHA 10 SELECIONA MODO DE TELA DE ALTA RESOLUÇÃO.
LINHA 20 A$ CONTÉM COMANDOS DA MLG. PROCURE ENTENDÊ-LOS.
LINHA 30 POSICIONA O CURSOR GRÁFICO EM 100, 150 SEM DESENHAR.
LINHA 40 DESENHA UM QUADRADO A$ DE TAMANHO NORMAL.
LINHA 50 ESPERA QUE SE PRESSIONE UMA TECLA.
LINHA 70 DESENHA UM QUADRADO A$ DE TAMANHO I.
LINHA 90 RETÉM AO MODO GRÁFICO DE TELA.

Observe que A$ se lê "direita 50 pontos, para cima 50 pontos, esquerda 50 pontos, para baixo 50 pontos".

Pontos a serem lembrados

O comando X é muito útil, pois você pode definir parte de um desenho que pode ser incorporado em qualquer lugar do programa. O conceito é como ter uma sub-rotina gráfica dentro do comando gráfico.

O comando X vai permitir-lhe fazer um desenho que exceda os 255 caracteres se for colocado explicitamente no único comando DRAW.

Comando S: se você alterar o valor de S (4 é o default), o computador se lembrará do novo valor e desenhará tudo depois dele naquela escala.

Caso você queira voltar à escala normal, tem de colocar S4 em um comando DRAW.

BM move o cursor gráfico a um determinado ponto no modo de tela sem fazer nenhum desenho. Esta é a melhor maneira de posicionar um objeto, apesar de haver muitas outras maneiras de se fazê-lo.

Caso o ponto de origem não seja especificado pelo BM, então o computador começará a partir do último ponto especificado, isto é, a partir de onde o cursor gráfico terminou depois do último comando DRAW.

Você pode estabelecer a origem utilizando PSET ou o comando M. Assim que um comando de ângulo A é executado, todo desenho seguinte vai ser girado, a não ser que você o devolva à direção original com outro comando A. O ângulo não é retornado ao valor default no final do comando DRAW ou mesmo no final do programa.

A cor atual de texto permanecerá a mesma depois da execução do comando DRAW que contém um comando C.

Corrigindo erros

Você pode utilizar DRAW apenas no modo gráfico (SCREENS 2 e 3).

Caso você erre a sintaxe da MLG, resultará um erro de chamada ilegal de função (Illegal function call).

Comandos associados e referências

COLOR
SCREEN

Parte 1, Capítulo 26 (Vol. 1): "A Macrolinguagem Gráfica".

ELSE

Descrição

ELSE é parte da estrutura IF/THEN/ELSE. ELSE informa ao computador que se a <condição> em IF não for satisfeita ele deverá pular o comando após THEN e executar o comando após ELSE.

ELSE GOTO <linha> pode ser abreviado para ELSE <linha>.

Podem ser estabelecidos múltiplos comandos IF/THEN/ELSE, bem como comandos IF/THEN/ELSE em níveis. São limitados no comprimento apenas pelo número máximo de caracteres permitidos em uma linha (que é 255).

Sintaxe

IF <condição> THEN <comando> ELSE <comandos>
IF <condição> THEN <comando> ELSE <linha>

Exemplos

Uma estrutura IF/THEN/ELSE simples.

```
10 INPUT D$
20 IF D$="NORTE" THEN PRINT "VOCE ENCONTRA UM MONSTRO
HORRIVEL!!" ELSE PRINT "VOCE ESTA NO ";D$
30 GOTO 10
```

```
? LESTE
VOCE ESTA NO LESTE
? SUL
VOCE ESTA NO SUL
? norte
VOCE ESTA NO norte
? NORTE
VOCE ENCONTRA UM MONSTRO HORRIVEL!!
```

Pontos a serem lembrados

Caso haja menos ELSE do que THEN em uma estrutura IF/THEN/ELSE em níveis, cada ELSE sempre combinará com o THEN mais próximo.

Por exemplo:

```
IF THEN(IF THEN(IF THEN ELSE)ELSE)
```

Os comandos IF/THEN/ELSE têm a tendência de ficar muito longos para caber em uma única linha, porque você pode ter múltiplos comandos depois do THEN e ELSE. Se este é o caso, então é aconselhável utilizar sub-rotinas, utilizando o comando GOSUB.

IF/THEN/ELSE poderá criar um programa "espaguete" se não tomar cuidado.

Corrigindo erros

Há a tendência de encontrar erros na seção <condição> de IF/THEN/ELSE.

Comandos associados e referências

AND
OR
NOT
IF
THEN

Parte 1, Capítulo 6 (Vol. 1): "O Uso de Condições".
Parte 2, Capítulo 35 (Vol. 1): "Álgebra Booleana II: o IF/THEN/ELSE".

END

Descrição

END informa ao computador que ele chegou ao fim do programa e o coloca de volta ao modo direto.

Você pode utilizar o comando END em qualquer parte do programa e com a freqüência necessária.

Você poderá necessitar um comando END antes de uma sub-rotina para impedir que o programa entre nela acidentalmente.

Você não requer um END exatamente no final do programa, porque o computador presume que é o fim do programa quando ficam sem linhas para executar.

Quando o computador encontra um comando END, fecha todos os arquivos abertos durante o programa. Isto é diferente do STOP que, apesar de interromper a execução do programa, não fecha os arquivos abertos durante o programa.

Sintaxe

```
END
```

Exemplos

```
9999 END
```

Pontos a serem lembrados

A diferença entre STOP e END é que STOP envia a mensagem "Break", mas END não o faz. STOP não fecha nenhum arquivo aberto.

Corrigindo erros

Vamos declarar o óbvio: se você colocar um END onde não deseja que o programa termine, então o programa termina prematuramente. Certo!

Comandos associados e referências

STOP

Parte 1, Capítulo 7 (Vol. 1): "Comandos Úteis e Dicas para Escrever Programas".

EOF FIM DE ARQUIVO

Descrição

EOF informa-lhe se foi atingido um fim de arquivo. Devolve –1 (verdadeiro) se atingiu; em caso negativo, entrega 0 (falso). Esta é a parte vital da manipulação de arquivos e é utilizado com OPEN etc.

Sintaxe

EOF(<número do arquivo>)

Exemplos

```
IF EOF(4)= – 1 THEN PRINT "FIM DE ARQUIVO":END
```

Corrigindo erros

Se você tentar escrever em um arquivo que já está completo sem testar a condição EOF, resultará um erro "Input Past end" (no Hot-Bit: Fim de Arquivo). Se você se referir a um arquivo não aberto em <número do arquivo> resultará um erro de "File not open" (no Hot-Bit: Falta "Open").

Comandos associados e referências

OPEN
CLOSE
INPUT

Parte 2, Capítulo 47 (Vol. 1): "Como Usar Arquivos".

EQV EQUIVALENTE

Descrição

EQV é um dos operadores lógicos booleanos utilizados na manipulação binária. Sua operação pode ser relacionada na seguinte tabela da verdade:

EQV	X	Y	E EQV Y
	0	0	1
	1	0	0
	0	1	0
	1	1	1

Exemplo:	51	EQV 74
	51	0000000000110011
EQV	74	0000000001001010
	−122	1111111110000110

Sintaxe

<numérico> EQV <numérico>

<numérico> tem de estar na faixa de inteiros entre –32768 e 32767. Todas as partes fracionárias dos números decimais são arredondadas.

Exemplos

```
10 A=51
20 B=74
30 PRINT A;" EQV ";B
40 PRINT A EQV B,BIN$(A EQV B)


 51  EQV  74
-122          1111111110000110
```

Pontos a serem lembrados

As seguintes relações são verdadeiras:

X EQV X = –1

–X EQV X = 1

Corrigindo erros

Ocorre um erro do tipo "Overflow" quando o argumento estiver fora da faixa dos inteiros (–32768 até 32767).

Comandos associados e referências

AND
OR
XOR
IMP
NOT

Parte 2, Capítulo 34 (Vol. 1): "Álgebra Booleana I: Expressões Lógicas".

ERASE APAGUE UMA MATRIZ

Descrição

Este comando permite que você apague matrizes criadas com o comando DIM.

Isto permite então que você redimensione a matriz apagada.

Sintaxe

ERASE<nome da matriz>
ERASE<nome da matriz> , <nome da matriz> , ...

Exemplos

```
10  A%=99
20  DIM A%(150)
30  FOR I=1 TO 150
40  A%(I)=I
50  NEXT I
60  ERASE A%
70  DIM A%(1000)
80  FOR I=1 TO 1000
90  A%(I)=I
100 NEXT I
110 PRINT A%

RUN
99
```

Pontos a serem lembrados

Uma variável com o mesmo nome que a matriz não será afetada pelo comando, conforme demonstrado no programa anterior.

Apenas o nome da matriz que vai ser apagada será necessário no comando ERASE. Significa que você não precisa escrever ERASE A%(150); basta ERASE A%.

Corrigindo erros

Apagar uma matriz inexistente causará um erro do tipo "illegal function error".

Caso você redimensione uma matriz sem antes utilizar ERASE, obterá um erro "Redimensioned array" (matriz redimensionada).

Comandos associados e referências

DIM

Parte 1, Capítulo 12 (Vol. 1): "Lendo Dados em Matrizes".

ERL LINHA DE ERRO

Descrição

ERL é uma variável reservada que contém o número de linha da linha que contém o erro encontrado no programa. É utilizada na rotina para localização de erros.

Sintaxe

<var-num> =ERL
PRINT ERL

Exemplos

```
10 ON ERROR GOTO 1000
20 PRINT "Z80"
30 PRINT "TMS 9918"
40 PRONT "FALHA"
50 END
1000 E%=ERL
1010 PRINT"Uma pequena falha na linha";E%
1020 END

Z80
TMS 9918
Uma pequena falha na linha 40
```

Pontos a serem lembrados

Quando ocorre um erro ao estarmos no modo direto, ERL conterá 65535.

ERL é uma variável reservada, de modo que não é possível estabelecer um número para ela.

Corrigindo erros

Utilize ERL em uma rotina para localização de erros para encontrar a posição exata dos mesmos.

Comandos associados e referências

ON ERROR GOTO
ERROR
ERR

Parte 2, Capítulo 38 (Vol. 1): "Tratamento de Erros".

ERR NÚMERO DE CÓDIGO DE ERRO

Descrição

Esta é uma variável reservada que contém o número de código do erro que o computador detectou no programa. É utilizada principalmente em sub-rotinas para detecção de erro.

Tem um valor entre 1 e 255 e pode ser utilizada em conjunto com mensagens de erro definidas pelo usuário.

ERR é normalmente utilizada na criação de novos erros utilizando o comando ERROR (veja o exemplo a seguir).

Sintaxe

<var-num> =ERR

Exemplos

Nesta rotina as strings com MATE serão tratadas como um novo erro (Nº 255) e

estes são tratados na rotina de localização de erros utilizando a função ERR.

```
10 ON ERROR GOTO 1000
20 INPUT"MEU MESTRE,QUAL O SEU DESEJO";A$
30 IF INSTR(A$,"MATE") THEN ERROR 255
40 PRINT"OK":END
50 END
100 IF ERR=255 THEN PRINT "VOCE NAO PODE MATAR NADA NE
STA AVENTURA":RESUME 20
110 END
1000 IF ERR=255 THEN PRINT "VOCE NAO PODE MATAR NADA N
ESTA AVENTURA:FIM DO PROGRAMA":END
1010 END

MEU MESTRE,QUAL O SEU DESEJO? MATE
VOCE NAO PODE MATAR NADA NESTA AVENTU
RA:FIM DO PROGRAMA
```

Pontos a serem lembrados

ERR é uma variável reservada; portanto, não se pode estabelecer um valor para ela.

Corrigindo erros

Utilize esta variável para encontrar erros em seus programas.

Comando associados e referências

ERL
ERROR
ON ERROR
RESUME

Parte 2, Capítulo 38: "Tratamento de Erros".

ERROR

Descrição

Existem basicamente duas maneiras de se utilizar o comando ERROR:

1. Para simular a ocorrência de um erro.
2. Para criar erros definidos pelo usuários utilizando comandos ON ERROR GOTO.

Um comando ERROR com um argumento inteiro vai levar o computador a pensar que há um erro, vai terminar o programa e imprimir a mensagem de erro com o número de linha.

Um comando ERROR, juntamente com a utilização de um ON ERROR GOTO, para detectar o erro, permitirá que você crie seus próprios erros sob medida.

Existem cerca de 36 erros, entre os códigos de números de erro de 1 até 60, na versão atual do MSX-BASIC. Os números de código 61 até 255 podem ser utilizados pelo programador.

Para programar seu próprio erro, primeiramente você terá de colocar ON ERROR GOTO <linha> no começo do programa. Depois, se estiver no meio do programa e surgir uma condição tal que você queira chamar a rotina de encontro de erro, poderá fazê-lo da seguinte forma:

```
IF <condição> THEN ERROR <código de erro>
```

ON ERROR GOTO será ativado e o computador imediatamente começará a executar a sub-rotina de encontro de erro. Na sub-rotina você deverá ter uma linha dizendo:

```
IF ERR= <código de erro> THEN PRINT "mensagem de erro"
```

A rotina de encontrar erros poderá terminar o programa pela utilização do comando END ou reassumir operação em qualquer linha especificada.

Sintaxe

ERROR <código de erro>
onde <código de erro> = 0 até 255.

Exemplos

1. Nesta rotina todos os comandos colocados com MATE serão tratados como um novo erro (nº 255) e serão tratados na rotina de localização de erros utilizando a função ERR.

```
10 ON ERROR GOTO 1000
20 INPUT"MEU MESTRE,QUAL O SEU DESEJO";A$
30 IF INSTR(A$,"MATE") THEN ERROR 255
40 PRINT"OK":END
50 END
1000 IF ERR=255 THEN PRINT "NESTA AVENTURA VOCE NAO PO
DE MATAR":RESUME 20
1010 END
```

```
MEU MESTRE,QUAL O SEU DESEJO? MATE
NESTA AVENTURA VOCE NAO PODE MATAR
MEU MESTRE,QUAL O SEU DESEJO? SORRIR
OK
```

Pontos a serem lembrados

É de prática normal no MSX-BASIC, ao definir seus próprios códigos de erro, ir descendo a partir do 255 de modo a evitar choques com qualquer melhoramento futuro às mensagens de erro feitas pelos fabricantes dos computadores MSX.

Se o código de erro não tiver nenhuma mensagem de erro anteriormente definida, quando o computador encontrar este erro no programa imprimirá "Unprintable Error" (no HOT-BIT: Erro Indefinido) e interromperá o programa.

Nenhum comando do tipo ON ERROR GOTO é executado durante a execução de uma rotina para encontro de erro. Caso haja um erro no interior da rotina de encontro de erro, fornecerá a mensagem de erro e interromperá o programa.

Comandos associados e referências

ON ERROR GOTO
ERL
ERR
RESUME

Parte 2, Capítulo 38 (Vol. 1): "Tratamento de Erros".

EXP EXPOENTE

Descrição

Esta função matemática fornece o valor exponencial do argumento dado, X.

e^X

Sintaxe

EXP(<const-num>)
EXP(<var-num>)
EXP(<exp-num>)

Exemplos

```
PRINT EXP(1)
  2.7182818284588
```

Ponto a serem lembrados

Esta função calcula em precisão dupla (até 14 dígitos).

Corrigindo erros

Caso a exponencial seja muito grande para que o computador possa cuidar dela, resultará um erro do tipo "Overflow" (Extravasamento).

Comandos associados e referências

LOG

Parte 1, Capítulo 19 (Vol. 1): "Funções Matemáticas".

FIX DEVOLVER PARTE INTEIRA

Descrição

Esta função devolve a parte inteira do argumento. As frações são truncadas.

FIX é equivalente à SGN(x)*INT(ABS(x)).

FIX fornece o inteiro mais próximo inferior para argumentos positivos e o inteiro mais próximo superior para argumentos negativos.

```
FIX(1.87) = 1
FIX(-12.88) = -12
```

Sintaxe

FIX(<const-num>)
FIX(<var-num>)
FIX(<exp-num>)

Exemplos

PRINT FIX(–1.2209)
IMPRIMIRÁ –1
E
A%=FIX(10.99)
FARÁ A%=10

Pontos a serem lembrados

A função INT fornece o número inferior mais próximo com um argumento, seja ele negativo ou positivo. Por exemplo: INT(– 1.3) = –2. De modo que há uma diferença entre INT e FIX.

Corrigindo erros

Esta é uma função numérica, portanto não a misture com strings ou ocorrerá um erro do tipo "Type mismatch" (no HOT-BIT: Tipo Desigual).

Comandos associados e referências

INT

Parte 1, Capítulo 15 (Vol. 1): "Funções".

FOR LOOP FOR/TO/NEXT

Descrição

FOR é um dos comandos utilizados para criar loops. Define-se os limites inferiores e superiores do contador do loop e também a quantidade com que o contador é incrementado cada vez em que o loop for completado.

Por exemplo:

```
10 FOR I=1 TO 3
20 PRINT I
30 NEXT I
```

fornecerá os números 1, 2, 3.

O contador I inicialmente toma o valor 1 e executa o comando entre FOR e NEXT, isto é, linha 20 PRINT I. Quando chega à linha 30 NEXT I, o computador volta à linha 10, incrementa o contador I em um e passa novamente pelo loop, até que I seja igual ao valor limite de 3.

Até aqui vimos o formato FOR/TO/NEXT, mas podemos introduzir STEP na linha que contém o comando FOR a fim de controlar o tamanho do incremento. Portanto, podemos modificar o programa anterior da seguinte forma:

```
10 FOR I=1 TO 3 STEP 0.5
20 PRINT I
30 NEXT I
```

imprimirá 1, 1.5, 2, 2.5, 3.

É possível ter um loop em níveis, isto é, um loop dentro de outro loop, como a seguir:

```
10 FOR I=1 TO 5
20 FOR J =1 TO 5
30 PRINT I,J
40 NEXT J
50 NEXT I
```

que vai imprimir 1, 1.2, 1.3 etc.

Há determinadas regras para loops em nível:

1. Cada loop tem de ter uma variável diferente como contador.
2. O NEXT para o loop interno tem de ser executado antes daquele do loop externo.
3. Você pode ter loops em níveis com a profundidade que quiser. O único fator de limitação é a memória.
4. Um único NEXT com uma lista de contadores pode substituir uma série inteira de comandos NEXT.

Sintaxe

FOR<var-num> = <numérico> TO <numérico>
FOR<var-num> = <numérico> TO <numérico> STEP <numérico>

Exemplos

Você pode ter um incremento negativo como indicado a seguir:

```
10 FOR F=10 TO 5 STEP -1
20 PRINT F
30 NEXT F
Run
10
9
8
7
6
5
```

Pontos a serem lembrados

Deixar a execução de um comando FOR/TO/NEXT inacabada, isto é, sair no meio de um LOOP é muito desaconselhável em termos de programação.

Corrigindo erros

Você tem de ter um comando NEXT correspondente para cada comando FOR, caso contrário um NEXT sem FOR causará um erro.

Comandos associados e referências

NEXT
STEP
TO

Parte 1, Capítulo 5 (Vol. 1): "O Uso de Loops".

FRE QUANTIDADE DE MEMÓRIA LIVRE

Descrição

Esta variável de sistema informa quanta memória lhe sobra na área de programação BASIC e na área de armazenamento de string.

Sintaxe

FRE(0)	fornece a memória que sobra na área de programação BASIC.
FRE ("")	fornece a memória que sobra na área de string.

Exemplos

```
PRINT FRE(0)
28815
PRINT FRE("")
200
```

Pontos a serem lembrados

FRE(0) é equivalente à área de memória indicado por FREE no mapa de memória.

Corrigindo erros

Você precisa do argumento (0) ou (" ") para utilizar a função FREE.

Comando associados e referências

Parte 1, Capítulo 7 (Vol. 1): "Comandos Úteis e Dicas para Escrever Programas".

Parte 2, Capítulo 48 (Vol. 1): "Mapa da Memória".

GOSUB VÁ À SUB-ROTINA

Descrição

Em programação BASIC, freqüentemente você necessita usar a mesma rotina em lugares diferentes. Você pode colocar uma sub-rotina em um local diferente daquele em que vai ser necessário, e pode utilizar GOSUB para chamar a sub-rotina no número de linha especificado quantas vezes precisar.

Quando o BASIC encontra o comando GOSUB, imediatamente passa para o número de linha especificado e continua a execução a partir desta linha. Quando o computador chega ao comando RETURN, volta à rota principal original que deixou para ir à linha da sub-rotina.

Você pode ainda chamar uma sub-rotina a partir de uma sub-rotina. De fato, você pode ter sub-rotinas em níveis com a profundidade desejada, desde que a memória não se esgote. É possível ter uma sub-rotina recursiva que chama a si própria.

Sintaxe

GOSUB<linha>

Exemplos

Um exemplo de um GOSUB recursivo. A sub-rotina chama a si mesma na linha 110 até que a condição seja satisfeita.

```
10 PRINT "INICIO"
20 A=1
30 GOSUB 100
40 PRINT A
50 END
90 REM SUB-ROTINA
100 A=A+1
110 IF A<>100 THEN GOSUB 100
120 RETURN
```

Pontos a serem lembrados

Não é de boa prática pular para uma linha com um comando REM porque poderá haver necessidade de cancelar os comandos REM para poupar memória posteriormente.

Não é possível ter uma expressão de variável para o número de <linha>, isto é, GOSUB A%*10 está errado.

É comum na programação de computador, tornar a sub-rotina facilmente distinguível do programa principal e preferivelmente anotada com REM.

Caso a lógica assim dite, é perfeitamente possível ter mais que um RETURN para uma sub-rotina.

Assegure-se, ao fazer a programação, de que não está entrando na sub-rotina por acidente; utilize END antes das sub-rotinas.

GOSUB é muito utilizado também nos comandos para controlar eventos como ON SPRITE etc.

Corrigindo erros

Caso GOSUB <linha> aponte para um número de linha inexistente, resultará um erro do tipo "Undefined Line Number" (no HOT-BIT: Nº Linha Inexistente).

Comandos associados e referências

RETURN
ON...GOSUB
ON INTERVAL GOSUB
ON KEY...GOSUB
ON SPRITE GOSUB
ON STOP GOSUB
ON STRING SOGUB

Parte 1, "Capítulo 17 (Vol. 1): "Estruturando o seu Programa".

GOTO VÁ PARA UM NÚMERO DE LINHA

Descrição

GOTO informa ao computador para pular até o número de linha especificado, e depois continuar a execução a partir desta linha.

Em um comando do tipo IF/THEN/ELSE, você pode substituir THEN GOTO apenas com GOTO ou THEN. Seu computador vai compreender o que você quer dizer.

Sintaxe

GOTO<linha>

Exemplos

```
10 PRINT "BEM-VINDO AO MSX"
20 GOTO 10

BEM-VINDO AO MSX
BEM-VINDO AO MSX
BEM-VINDO AO MSX
BEM-VINDO AO MSX
BEM-VINDO AO MSX
BEM-VINDO AO MSX
BEM-VINDO AO MSX
BEM-VINDO AO MSX
BEM-VINDO AO MSX
BEM-VINDO AO MSX
```

Continua a imprimir "BEM-VINDO AO MSX" até que você opere <CTRL> <STOP>.

Pontos a serem lembrados

Você pode ir até a mesma linha que contenha o GOTO. Isto é utilizado para retestar determinadas condições e reter modos gráficos de tela.

Por exemplo:

```
100 IF INKEY$< >"A"GOTO100
```

Corrigindo erros

Caso a <linha> de GOTO aponte um número de linha inexistente, ocorrerá um erro do tipo "Undefined Line Number" (no HOT-BIT: Nº Linha Inexistente).

Comandos associados e referências

ON GOTO
ON ERROR GOTO

Parte 1, Capítulo 3 (Vol. 1): "Escrevendo um Programa".

HEX$ UMA STRING HEX

Descrição

HEX$ fornece o valor hexadecimal de um número decimal inteiro na forma de uma string. Por exemplo: HEX$ (255) fornece a string FF.

Sintaxe

HEX$(<número-inteiro>)

Pode ser utilizado na faixa de –32768 até 65535.

Para inteiros negativos, é utilizado o complemento de dois, isto é, HEX$ (65535 – x) = HEX$ (–x).

Exemplos

```
PRINT HEX$ (255)
AS=HEX$(NUM%)
```

Pontos a serem lembrados

&H <número> fornece o valor decimal de um número hexadecimal.

Corrigindo erros

O argumento deve ser um inteiro, ao passo que o resultado é uma string.

Comandos associados e referências

OCT$
BIN$

Parte 2, Capítulo 33 (Vol. 1): "Introdução aos Sistemas Numéricos Usados no MSX".

IF

Descrição

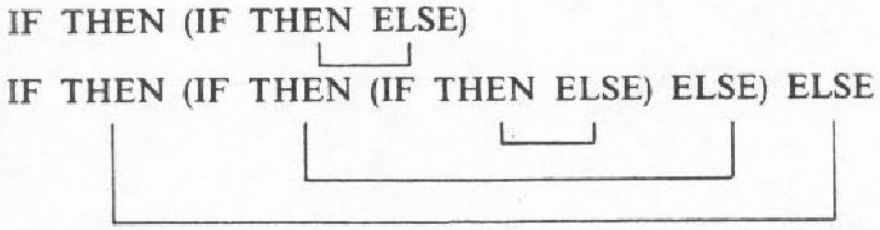
O comando IF testa uma condição, ou uma combinação de condições, reagindo de acordo.

Os comandos de teste de condição envolvendo IF têm os seguintes formatos:

1. IF (se) a condição é verdadeira THEN (então) faça tudo depois da palavra THEN.
2. IF (se) a condição é falsa THEN (então) não execute os comandos depois do THEN, mas continue a partir da próxima linha.

ELSE é parte da estrutura IF/THEN/ELSE. ELSE informa ao computador que se a <condição> não é satisfeita (isto é, é falsa), então pule o que houver depois do THEN e execute os comandos depois do ELSE. Diversos comandos IF/THEN/ELSE podem ser utilizados, bem como comandos IF/THEN/ELSE em níveis. Estão limitados apenas pelo tamanho da linha que tem 255 caracteres.

Caso haja um número menor de ELSE que de THEN, cada ELSE é combinado com o THEN mais próximo.



O formato da <linha> IF...THEN GOTO pode ser abreviado para um dos seguintes formatos:

1. IF...THEN <linha>
2. IF...GOTO <linha>

ELSE GOTO <linha> pode ser abreviado para ELSE <linha>.

Sintaxe

IF<condição> THEN <declarações>
IF<condição> THEN <linha>
GOTO
IF<condição>THEN<comando ou linha> ELSE <comando ou linha> GOTO

Exemplos

A linha 50 é uma estrutura IF THEN simples.

A linha 60 é uma estrutura IF/THEN/ELSE IF/THEN/ELSE.

```
10 PRINT "INTRODUZA TRES NUMEROS DIFERENTES"
20 INPUT"A";A%
30 INPUT"B";B%
40 INPUT"C";C%
50 IF A%=B% OR A%=C% OR B%=C% THEN 10
60 IF A%<B% AND A%<C% THEN PRINT "A" ELSE IF B%<A% AND
 B%<C% THEN PRINT "B" ELSE PRINT "C"
70 PRINT " E O MENOR NUMERO"
```

```
INTRODUZA TRES NUMEROS DIFERENTES
A? 8
B? 4
C? 3
C
 E O MENOR NUMERO
```

Pontos a serem lembrados

Os comandos IF/THEN/ELSE tendem a ficar muito longos para caberem em uma linha, porque você pode ter diversos comandos depois do THEN e ELSE. Se este for o caso, então será conveniente utilizar sub-rotinas com o comando GOSUB.

É aconselhável evitar comandos IF/THEN/ELSE em níveis, uma vez que isto poderá criar uma certa confusão. Se não tomar cuidado, acabará criando um programa "espaguete".

Corrigindo erros

Você tende a encontrar erros na seção <condição> de IF/THEN/ELSE.

Comandos associados e referências

THEN
ELSE
AND
OR
NOT

Parte 1, Capítulo 6 (Vol. 1): "O Uso de Condições".

Parte 2, Capítulo 35 (Vol. 1): "Álgebra Booleana II: o IF/THEN/ELSE".

IMP IMPLICAÇÃO

Descrição

Um dos operadores lógicos booleanos.

A operação algébrica booleana pode ser relacionada na tabela da verdade IMP:

IMP	X	Y	X IMP Y
	0	0	1
	1	0	0
	0	1	1
	1	1	1

100 IMP 50 pode ser calculado como segue:

	X	100	0000000001100100
IMP	Y	50	0000000000110010
		–69	1111111110111011

Sintaxe

<var-num> = <const-num> IMP <const-num>
<var-num> = <var-num> IMP <exp-num>

e outras combinações numéricas.

Exemplos

```
10 S=99
20 PRINT S IMP S
30 PRINT (-S IMP S)
40 PRINT (S IMP -S)
```

```
RUN
-1
 99
-99
```

Pontos a serem lembrados

As seguintes relações são verdadeiras:

```
X IMP X = –1
–X IMP X = X
X IMP –X = –X
```

IMP é uma operação de 16 bits.

Corrigindo erros

Resultará um erro do tipo "Overflow" (extravasamento) se um dos argumentos estiver fora da faixa dos inteiros.

Comandos associados e referências

AND
OR
NOT
XOR
EQV

Parte 2, Capítulo 34 (Vol. 1): "Álgebra Booleana I: Expressões Lógicas".

INKEY$

Descrição

Esta função testa o teclado para ver se alguma tecla foi pressionada, e, em caso afirmativo, retorna o caractere digitado. Caso não haja nenhuma tecla sendo pressionada retornará uma string nula.

Sintaxe

<var-string> = INKEY$

Exemplos

Este programa demonstra como fazer o computador esperar até que uma tecla seja pressionada. A linha 20 testa para ver se uma tecla foi pressionada e o caractere armazenado em A$. Se nenhuma tecla foi pressionada, informa ao computador para que volte ao início da mesma linha e para que tente novamente até que uma seja pressionada.

```
10 PRINT "PRESSIONE QUALQUER TECLA PARA SAIR DO  PROGR
AMA"
20 A$=INKEY$:IF A$="" THEN 20 ELSE STOP

PRESSIONE QUALQUER TECLA PARA SAIR DO
  PROGRAMA
Break in 20
```

Pontos a serem lembrados

Existem muitas diferenças entre INKEY$ e INPUT:

1. INKEY$ não fornecerá um ponto de interrogação "?" no modo de tela.
2. INKEY$ não espera você pressionar uma tecla. Ou você tem uma tecla pressionada enquanto o computador executa INKEY$ ou receberá de volta uma string nula.
3. INKEY$, ao contrário de INPUT, não apresenta o caractere devolvido no modo de tela.
4. INKEY$ reconhece as teclas TAB e DEL.
5. Ao contrário de INPUT, INKEY$ não coloca você no modo texto ao ser executado no modo gráfico.
6. INKEY$ devolve apenas um caractere ao passo que INPUT devolve uma linha inteira de caractere.

Corrigindo erros

Ocorre um erro do tipo "Type Mismatch" (no HOT-BIT: Tipo Desigual), caso INKEY$ esteja igualada a uma variável numérica.

Comandos associados e referências

INPUT$

Parte 1, Capítulo 10 (Vol. 1): "Programação Interativa".

INP ENTRADA DE UMA PORTA

Descrição

Devolve o byte lido na porta especificada. O número da porta tem de estar na faixa de 0 até 255. Qualquer valor acima de 255 não é utilizado pelo MSX.

Não utilize este comando para software comercial. Utilize o BIOS – é mais fácil.

Sintaxe

<var-num> = INP(<número da porta>)

Exemplo

```
PRINT INP(1)
```

Pontos a serem lembrados

INP é o oposto de OUT.

Corrigindo erros

A não ser que saiba exatamente o que estiver fazendo procure não utilizá-lo.

Comandos associados

OUT
WAIT

INPUT

Descrição

INPUT permite que você coloque números e strings em variáveis durante o processamento do programa, de modo a fazer com que o programa seja interativo com o usuário.

O INPUT interrompe o programa e espera que os dados necessários sejam dígitados pelo teclado.

É possível incluir um texto antes do ponto de interrogação que aparece na tela.

Qualquer número de variáveis numéricas ou string pode ser colocado um após o outro no mesmo comando INPUT, desde que sejam separados por vírgulas. Entretanto, o tipo de dado digitado e o tipo de variável especificado no comando INPUT têm de

concordar, caso contrário teremos:

? Redo from start

e você terá de redigitar todas as entradas desde a primeira variável solicitada.

Sintaxe

INPUT<lista de variáveis numéricas ou strings separadas por vírgulas>
INPUT "mensagem";<lista de variáveis string ou numéricas>

Exemplos

```
10 INPUT "SEU NOME";N$
20 INPUT "SUA IDADE E RELIGIAO";I%,R$
30 PRINT N$
40 PRINT I%
50 PRINT R$

SEU NOME? JOAO DA SILVA
SUA IDADE E RELIGIAO? 80
?? TODAS
JOAO DA SILVA
 80
TODAS
```

Pontos a serem lembrados

No modo gráfico, INPUT obriga você a voltar ao modo texto.

O comando INPUT responde de três maneiras quando a entrada digitada não estiver como deveria:

1. Um erro de desencontro de tipo causará:

?Redo from start

e você terá de redigitar tudo a partir do início.

2. Entradas em excesso provocarão:

?Extra ignored

depois a execução do programa continua.

3. Uma quantidade insuficiente de entradas causará:

??

dois pontos de interrogação, e o computador vai aguardar até que a(s) variável(eis) faltante(s) esteja(m) digitada(s).

A única maneira de escapar do INPUT é <CTRL> <C> ou <CTRL> <STOP>.

É possível voltar ao programa depois de um erro deste tipo, utilizando o comando CONT, mas você vai ter de redigitar as respostas a partir do começo.

Se apenas a tecla RETURN for operada, a variável de entrada não será alterada, (isto é, ainda conterá seu valor anterior).

Cada variável tem de ser separada da próxima por uma vírgula na lista de variáveis. Os itens digitados na resposta também têm de ser separados por vírgulas.

Corrigindo erros

Erros de digitação durante a entrada serão tratados pelo computador conforme acima.

Caso o comprimento de uma string de entrada exceda a 200 caracteres isso provocará um erro de falta de espaço para string, a não ser que você aumente o espaço de trabalho para strings utilizando o comando CLEAR.

Comandos associados e referências

INKEY$
INPUT#
INPUT$
LINE INPUT

Parte 1, Capítulo 3 (Vol. 1): "Escrevendo um Programa".

Parte 1, Capítulo 10 (Vol. 1): "Programação Interativa".

INPUT# ENTRADA DE UM ARQUIVO

Descrição

Este comando permite que você leia dados de outros dispositivos que não sejam o teclado, como de um cassete, por exemplo.

Antes que este comando seja executado, um canal específico de E/S deve ser aberto utilizando o comando OPEN e um número de arquivo designado.

O formato de entrada tem de ser exatamente o mesmo que seria aceito para o teclado.

Durante a entrada para variáveis numéricas, tanto espaços anteriores, como retornos de carro e alimentação de linha são ignorados. O mesmo se aplica às variáveis string, onde aspas também são ignoradas.

Sintaxe

INPUT# <número do arquivo> , <lista de variáveis numéricas ou strings>

Exemplos

```
INPUT#1, A$
```

Pontos a serem lembrados

Para abrir um arquivo ao cassete, utilize:

```
OPEN "CAS:" FOR INPUT AS # 1.
```

Corrigindo erros

Ocorrerá um erro "Bad file number" se o número de arquivo referido não for aberto antes.

Comandos associados e referências

OPEN
CLOSE
INPUT
INPUT$

Parte 2, Capítulo 47 (Vol. 1): "Como Usar Arquivos".

INPUT$

Descrição

Este comando devolve uma string com um dado número de caracteres, lida do teclado. Nenhum dos caracteres será apresentado na tela. <CTRL><C> encerra a encerra a execução da função INPUT$.

Sintaxe

```
<var-string> = INPUT$(<const-num>)
```

Exemplo

```
A$=INPUT$(5)
```

Pontos a serem lembrados

Ao contrário da função INKEY$, INPUT$ espera pelo caractere a ser digitado. Também pode tratar mais de um caractere por vez.

Comandos associados e referências

INKEY$
INPUT$(#)
INPUT

Parte 1, Capítulo 10 (Vol. 1): "Programação Interativa".

INPUT$(#)

Descrição

Este comando devolve uma string com um dado número de caracteres, lidos de um dispositivo diferente do que o teclado. Nenhum caractere será apresentado no modo de tela. O número do arquivo tem de ser especificado, sendo que o mesmo deve ter sido aberto utilizando o comando OPEN.

Sintaxe

<var-string> = INPUT$(<const-num>,<número do arquivo>)
<var-string> = INPUT$(<const-num>,#<número do arquivo>)

Exemplos

```
W$=INPUT$(5,#1)
```

Pontos a serem lembrados

Para abrir um arquivo ao cassete, utilize:

```
OPEN "CAS: "FOR INPUT AS #1
```

Corrigindo erros

Ocorrerá um erro de "Número de arquivo errado" (Bad hile name), caso o arquivo referido não tenha sido aberto antes.

Comandos associados e referências

OPEN
CLOSE
INPUT
INPUT#
INPUT$

Parte 2, Capítulo 47 (Vol. 1): "Como Usar Arquivos".

INSTR

Descrição

INSTR procura por uma determinada string dentro de outra string, e depois devolve a posição da string especificada se a encontrar.

INSTR (A$,B$)... A$ é a string principal e B$ é a string que deve ser procurada.

Caso a string não seja encontrada, INSTR devolve 0.

A procura inicia no começo da string, mas você também pode especificar o ponto de partida da procura. O número da posição deve variar entre 1 e 255 inclusive.

Sintaxe

<numérico>=INSTR(<var-string>,"<string>")
<numérico>=INSTR(<var-string>,<var-string>
<numérico>=INSTR(<numérico>,<var-string>,<var-string>

Exemplos

```
10 PRINT "A PRINCIPAL STRING E A SEGUINTE:"
20 PRINT "PROCURA-SE CHAPEUZINHO VERMELHO-LOBAO"
30 A$="PROCURA-SE CHAPEUZINHO VERMELHO-LOBAO"
40 INPUT "DIGITE O CARACTERE A SER PROCURADO";B$
50 C%=0
60 C%=INSTR(C%+1,A$,B$)
70 IF C%=0 THEN END
80 PRINT "CARACTERE";C%;"E ";B$
90 GOTO 60

A PRINCIPAL STRING E A SEGUINTE:
PROCURA-SE CHAPEUZINHO VERMELHO-LOBAO

DIGITE O CARACTERE A SER PROCURADO? C
HA
CARACTERE 12 E CHA
```

Pontos a serem lembrados

Para INSTR(Z, X$, Y$), você receberá de volta um 0 nos seguintes casos:

1. Se Y$ não é encontrado em X$.
2. Se Z for maior que o comprimento de X$.
3. Se X$ for uma string nula, isto é, X$".
4. Se X$ e Y$ são strings nulas.

Caso você tente procurar uma string nula, " ", obterá 1 de INSTR.

Corrigindo erros

Se a posição inicial da procura é 0 ou mais que 255, resultará um erro de chamada ilegal de função (Illegal function CALL).

Comandos associados e referências

LEFT$
RIGHT$
MID$
LEN

Parte 1, Capítulo 14 (Vol. 1): "Manipulando Strings".

INT INTEIRO

Descrição

Esta função devolve a parte inteira de um número real. Arredonda o número real ao número inteiro menor mais próximo, tanto para números positivos como negativos.

Sintaxe

INT(<const-num>)
INT(<var-num>)
INT(<exp-num>)

Exemplo

```
PRINT INT(2.76)
2
PRINT INT(10.1111)
10
PRINT INT(-1.234)
-2
```

Pontos a serem lembrados

Observe que INT de um número negativo devolve o número inteiro menor mais próximo do número fornecido. Portanto, você consegue com que INT(–0. 2) devolva –1.

INT é diferente de FIX. FIX fornece o inteiro inferior mais próximo para um argumento positivo e o inteiro superior mais próximo para um argumento negativo.

Corrigindo erros

Ocorrerá um erro de desencontro de tipo (Type mismatch error) se INT for usado com strings.

Comandos associados e referências

FIX

Parte 1, Capítulo 4 (Vol. 1): "Mais sobre Aritmética".

Parte 1, Capítulo 15 (Vol. 1): "Funções".

INTERVAL ON/OFF/STOP

Descrição

O MSX-BASIC pode ser temporizado, ou seja, em determinados intervalos de tempo ele pode executar uma determinada rotina utilizando o comando ON INTERVAL (veja ON INTERVAL).

Os comandos INTERVAL ON/OFF/STOP ativam e desativam o mecanismo para interrupções por tempo no ON INTERVAL.

Para utilizar ON INTERVAL GOSUB, você tem de ativá-lo com INTERVAL ON. Depois de INTERVAL ON, o computador verifica a cada vez em que executa uma linha nova, se chegou o momento certo para chamar a rotina especificada no ON INTERVAL GOSUB.

INTERVAL OFF informa ao computador para que deixe de verificar o intervalo de tempo decorrido e desativa ON INTERVAL GOSUB.

INTERVAL STOP informa ao computador para não executar ON INTERVAL GOSUB mas o lembra de continuar a cronometrar o tempo de modo a poder executar o desvio quando o comando INTERVAL ON for executado.

Sintaxe

```
INTERVAL ON
INTERVAL OFF
INTERVAL STOP
```

Exemplos

No programa a seguir, tecle <s> para começar a cronometrar o intervalo de tempo. Tecle <n> para interromper o intervalo de tempo. Você escutará som quando a interrupção do intervalo for ativada.

```
10 ON INTERVAL=60 GOSUB 60
20 IF INKEY$="s" THEN INTERVAL ON
30 IF INKEY$="n" THEN INTERVAL OFF
40 GOTO 20
50 REM SUB-ROTINA DE INTERVALO
60 BEEP
70 RETURN
```

Observação: O teclado deve estar na modalidade minúscula para a execução do programa acima.

LINHA 10 COLOCA O INTERVALO EM 1 SEG, SUB-ROTINA NA LINHA 50.
LINHA 20 ATIVA A INTERRUPÇÃO SE VOCÊ DIGITOU "S".
LINHA 30 DESATIVA A INTERRUPÇÃO SE VOCÊ DIGITOU "N".
LINHA 40 UM LOOP INFINITO PARA A LINHA 20.
LINHA 60 VOLTA AO PONTO EM QUE DEIXOU A ROTINA PRINCIPAL.

Pontos a serem lembrados

O intervalo de tempo é colocado em unidades de 1/60 do segundo.

As interrupções temporizadas são desativadas no modo direto, isto é, quando o programa não estiver sendo processado e durante as rotinas para detecção de erros.

Comandos associados e referências

ON INTERVAL GOSUB

Parte 2, Capítulo 37 (Vol. 1): "Tratamento de Intervalos e Interrupções pelo BASIC".

KEY

Descrição

Este comando é utilizado para definir a ação de uma tecla de função. Todos os computadores MSX têm cinco teclas de função, cada qual representando duas funções: uma acessada pela operação da tecla de função e a outra pela operação de <SHIFT> + <tecla de função>, simultaneamente.

Quando você liga o computador, a teclas de função já estão inicializadas da seguinte forma:

F1 = color[s]	Cor
F2 = auto[s]	Auto
F3 = goto[s]	Goto
F4 = list[s]	List
F5 = run[r]	Processa o programa.
F6 = color 15, 1, 1, [r]	Coloca a cor naquela default.
F7 = cload"	Carrega do cassete.
F8 = cont[r]	Continua o programa.
F9 = list.[r][u][u]	Lista a última linha referida e move o cursor ao começo daquela linha.
F10 = [c1s] run[r]	Limpa o modo de tela depois processa o programa

[s] = espaço
[r] = RETURN
[c1s] = limpa o modo de tela
[u] = subir o cursor uma linha

Você pode redefinir qualquer uma das teclas de função utilizando a função KEY. A string, entretanto, tem de ter menos que 15 caracteres de tamanho.

Se você quiser definir uma função de modo que seja executada imediatamente depois de operar a tecla de função, acrescente um caractere de retorno de carro CHR$(13) à string. Você pode também incluir o comando CLS e os códigos de outros caracteres de controle à string de função.

Sintaxe

KEY<const-num> , <string>
KEY<const-num> , <exp-string>
onde a <const-num> tem de estar entre 1 e 10.

Exemplos

```
KEY2, "PRINT FRE(0)" + CHR$(13)...    Imprime o número de bytes livres ao BASIC.
KEY4, "RENUM" CHR$(13)                Renumera.
a$ = "KEY LIST"
KEY5, a$ + CHR$(13)                   Devolve "KEY LIST"[R].
```

Nota: CHR$(13) é o caractere de controle da tecla <RETURN>, ou seja, equivale à tecla <RETURN>.

Pontos a serem lembrados

Quando o micro é ligado, os conteúdos das teclas de função são apresentados na linha inferior do modo de tela. Se você quiser apagar esta apresentação utilize o comando KEY OFF.

Lembre-se de que os caracteres de controle podem ser incorporados na tecla de função, de modo que você pode fazer com que as teclas de função limpem o modo de tela, emitam um som etc.

Corrigindo erros

Não se esqueça das aspas como delimitadores da string.

Comandos associados e referências

KEY ON/OFF
KEY LIST

Parte 1, Capítulo 8 (Vol. 1): "Teclas de Função".

KEY LIST

Descrição

Este comando lista o conteúdo de todas teclas de função sem executá-las.

Todos os caracteres de controle são convertidos em espaços ao serem listados.

Sintaxe

KEY LIST

Exemplo

```
color
auto
goto
list
run
color 15,1,1
cload"
cont
list
run
```

Pontos a serem lembrados

Veja o comando KEY para saber como alterar o conteúdo das teclas de função.

Comandos associados e referências

KEY
KEY ON/OFF

Parte 1, Capítulo 8 (Vol. 1): "Teclas de Função".

KEY ON/OFF

Descrição

Quando o computador é ligado, a 24ª linha apresenta o conteúdo das teclas de função. Isto tem a finalidade de lembrá-lo sobre o que faz cada tecla de função mas, algumas vezes, quando estiver processando um programa, a lista de teclas atrapalha. KEY ON/OFF permitirá que você ligue e desligue a lista das teclas de função da 24ª linha.

Sintaxe

```
KEY ON
KEY OFF
```

Exemplos

```
KEY ON
KEY OFF
```

Pontos a serem lembrados

KEY LIST vai fornecer-lhe uma lista do conteúdo das teclas de função.

Comandos associados e referências

```
KEY
KEY LIST
```

Parte 1, Capítulo 8 (Vol. 1): "Teclas de Função".

KEY ( ) ON/OFF/STOP

Descrição

O comando KEY ( ) ON/OFF/STOP ativa/desativa a detecção de uma determinada tecla de função. Assim que a detecção da tecla de função estiver ativada, o computador vai pular até a sub-rotina especificada pelo comando ON KEY GOSUB quando aquela tecla de função for pressionada.

Teclas de função individuais podem ser ativadas e desativadas separadamente, Teclas de função não especificadas em KEY ( ) ON são consideradas desativadas.

Caso KEY ( ) STOP tenha sido, executado, então o computador não desviará até a sub-rotina especificada quando a tecla de função especificada for operada, mas o computador pulará imediatamente à sub-rotina assim que KEY ( ) ON for executado para aquela tecla de função.

KEY ( ) OFF desativa completamente a inerrupção para aquela tecla de função.

Sintaxe

KEY(<const-num>)ON
KEY(<const-num>)OFF
KEY(<const-num>)STOP

Exemplos

Se você pressionar a tecla de função F0 o computador emitirá um som duas vezes; se pressionar F1 emitirá apenas um som.

```
10 ON KEY GOSUB 50,60
20 KEY (1) ON
30 KEY (2) ON
40 GOTO 40
50 BEEP
60 BEEP
70 RETURN
```

```
LINHA 10    ESTABELECE SUB-ROTINA PARA F0 E F1 NA LINHA 50
LINHA 20    ATIVA A TECLA DE FUNÇÃO F0.
LINHA 30    ATIVA A TECLA DE FUNÇÃO F1.
LINHA 40    LOOP INFINITO PARA ESPERAR POR UMA TECLA DE FUNÇÃO.
LINHA 50    SOM (PARA F0) E VOLTA PARA A LINHA 40.
LINHA 60    SOM (PARA F1).
LINHA 70    VOLTA AO LOCAL DE ONDE VEIO: NESTE CASO LINHA 40.
```

Observe que o programa anterior não faz nada quando outra tecla de função for pressionada.

Pontos a serem lembrados

A interrupção da tecla de função é desativada durante as rotinas de detecção de erros.

Corrigindo erros

Não se esqueça de executar primeiro o comando ON KEY GOSUB. Este comando é totalmente diferente de KEY ON/OFF.

Comandos associados e referências

ON KEY GOSUB

Parte 2, Capítulo 37 (Vol. 1): "Tratamento de Intervalos e Interrupções pelo BASIC".

LEFT$

Descrição

LEFT$ é uma função de manipulação de string, que retorna os n caracteres da esquerda de uma determinada string. Caso a string fonte seja menor que o valor dado, toda a string será retornada.

Sintaxe

LEFT$(<var-string> , <const-num>
LEFT$(<var-string> , <var-num>)
LEFT$(<var-string> , <exp-num>)

O número tem de estar na faixa de 0 até 255.

Exemplo

```
10 A$="Ze das Couves"
20 P=INSTR(A$," ")
30 B$=LEFT$(A$,P-1)
40 PRINT B$
```

LINHA 20 ENCONTRA A POSIÇÃO DO ESPAÇO EM A$.
LINHA 30 OBTÉM OS P-1 CARACTERES DA ESQUERDA DE A$ (O PRIMEIRO NOME).

```
Ze
```

Pontos a serem lembrados

Caso o número do argumento seja 0, então será devolvida uma string nula.

Para obtermos o sobrenome no programa anterior, você teria de utilizar MID$ da seguinte forma:

```
PRINT MID$(A$,P+1)
```

Corrigindo erros

O número do argumento tem de ser um inteiro entre 0 e 255, caso contrário ocorrerá um erro de desencontro de tipo ("Type mismatch error"). Ocorre uma chamada de função ilegal ("Illegal function call") quando uma variável não-existente é usada.

Comandos associados e referências

INSTR
RIGHT$
MID$
LEN

Parte 1, Capítulo 14 (Vol. 1): "Manipulando Strings".

LEN COMPRIMENTO DE UMA STRING

LEN devolve o comprimento de uma dada string.

Caracteres de controle e brancos também são contados.

Sintaxe

```
LEN(<string>)
```

Exemplos

```
10 A$="OLA"+CHR$(7)
20 B=LEN(A$)
30 PRINT A$
40 PRINT "COMPRIMENTO DE A$:";B
```

LINHA 10 CHR$(7) É O SOM DO BEEP.
LINHA 30 IMPRIME OLA COM O SOM DO BEEP.

```
OLA
COMPRIMENTO DE A$: 4
```

Pontos a serem lembrados

Este comando devolve sempre um inteiro.

Corrigindo erros

O comando devolve um zero, se uma variável string não-existente é referida.

Comandos associados e referências

RIGHT$
LEFT$
MID$
INSTR

Parte 1, Capítulo 15 (Vol. 1): "Funções".

LEFT

Descrição

Quando você estabelecer algum valor para uma string ou para uma variável numérica, utilize a declaração LET como em:

```
LET A 100
```

LET é opcional, sendo melhor não utilizá-la porque é desnecessária e desperdiça espaço de memória, isto é, A = 100 é equivalente ao comando acima.

Sintaxe

```
LEFT<variável> = <expressão>
```

Exemplos

```
LET A = 100
LETWS="AMERICA" + "DO" + "SUL"
```

Comandos associados e referências

Parte 1, Capítulo 2 (Vol. 1): "Modo Direto".

LINE DESENHA UMA LINHA

Descrição

Este comando gráfico desenha linhas e retângulos. Os retângulos podem ser preenchidos com qualquer cor.

Sintaxe

LINE–<especificação de coordenada>
desenha uma linha na cor atual do texto a partir do último ponto referido pelo computador até o ponto fornecido no especificador de coordenadas.

LINE–<esp coord> , <cor>
desenha uma linha na cor especificada a partir do último ponto referido pelo computador até o ponto fornecido no especificador de coordenada.

LINE<esp coord> – <esp coord>
desenha uma linha desde a primeira coordenada até a segunda coordenada na cor atual do texto.

LINE<esp coord> – <esp coord> , <cor>
desenha uma linha a partir da primeira coordenada até a segunda na cor especificada.

LINE<esp coord> – <esp coord> , <cor> , B
desenha um retângulo na cor especificada com o canto superior esquerdo do retângulo dado pelas primeiras coordenadas e o canto inferior direito fornecido pelas segundas coordenadas.

LINE<esp coord> – <esp coord> , <cor> , BF
desenha um retângulo na cor especificada com o canto superior esquerdo do retângulo fornecido no primeiro especificador de coordenadas e o canto inferior direito fornecido no segundo especificador de coordenadas. Depois, o retângulo é preenchido com a cor especificada.

Explicação de <esp coord>

Existem dois formatos de coordenadas: um relativo e outro absoluto.

1. (<coord-x> , <coord-y>)
   Estas especificam uma posição absoluta no modo de tela. x pode estar entre 0 e 255 e y entre 0 e 191.
2. STEP <deslocado-x> , <deslocado-y>
   Estas coordenadas são as de um ponto relativo e referem-se ao último ponto usado. Caso o deslocamento fique fora do modo de tela, o computador desenha até a borda do modo de tela.

<deslocado-x> e <deslocado-y> podem ser negativos dependendo da direção em que deseja desenhar.

Notas: <deslocado-x> , <deslocado-y> , <coord-x> , <coord-y> podem ser variáveis, expressões ou apenas constantes numéricas simples.

Quando você precisar de dois especificadores de coordenadas é perfeitamente normal usar o formato relativo com o absoluto, desde que ambos sejam ligados com um sinal "–" (negativo).

Exemplos

```
10 SCREEN 2
20 COLOR 4,1,1
30 CLS
40 LINE (50,50)-(100,100)
50 LINE (50,50)-(100,100),4,B
60 LINE STEP(-50,20)-STEP(50,50),9,BF
70 X=10:Y=10
80 LINE (15*X,10*Y)-STEP(50,20),6,B
90 GOTO 90
```

Você verá um quadrado com uma diagonal, outro vermelho e um retângulo

Pontos a serem lembrados

Coordenadas de númros reais são truncadas em inteiros. O código de cor tem de ser um número válido entre 0 e 15 (veja COLOR).

Corrigindo erros

As coordenadas podem ser maiores ou menores que o tamanho do modo de tela, mas se estiverem fora da faixa dos inteiros (–32768 até 32767) resultará um erro de "Overflow" (por exemplo: LINE (–100000, 100000) fornecerá "Overflow").

Comandos associados e referências

PAINT
COLOR
DRAW

Parte 1, Capítulo 24 (Vol. 1): "Desenhando Linhas e Retângulos".

Parte 2, Capítulo 41 (Vol. 1): "Gráficos Avançados II".

LINE INPUT

Descrição

Este comando é semelhante ao INPUT mas permite que você dê entrada a uma sentença inteira de até 200 caracteres, incluindo as vírgulas.

Você pode ter mensagens como no comando INPUT, mas um ponto de interrogação não será impresso automaticamente, ao contrário da declaração INPUT.

Sintaxe

LINE INPUT<var-string>
LINE INPUT"<mensagem>";<var-string>

Exemplos

Sub-rotinas de jogos de aventura.

```
10 LINE INPUT"O QUE POSSO FAZER AGORA?";S$
20 PRINT "DE ACORDO: ";S$
```

```
O QUE POSSO FAZER AGORA?APAGAR
DE ACORDO: APAGAR
```

Pontos a serem lembrados

Você pode escapar de LINE INPUT com <CTRL>< C> ou <CTRL> <STOP>. CONT vai deixar que você continue a partir de LINE INPUT.

Você não pode utilizar este comando no modo gráfico. Primeiro você tem de voltar ao modo texto.

O número máximo de caracteres em uma linha é 200. Este limite é estabelecido pelo tamanho da área de tabalho de strings da RAM do sistema. Se você quiser estender o espaço de trabalho das strings utilize CLEAR.

Corrigindo erros

Você pode utilizar este comando com uma variável string. Se você tentar utilizar LINE INPUT com uma variável numérica obterá um erro de desencontro de tipo.

Um erro de indicação de falta de espaço para strings ocorrerá se uma string digitada for muito grande para caber na área de trabalho de strings (o máximo inicial é 200).

Comandos associados e referências

LINE INPUT#
INPUT

Parte 1, Capítulo 10 (Vol. 1): "Programação Interativa".

LINE INPUT#

Descrição

Este comando é semelhante a INPUT#. Ele permite que você leia uma sentença inteira, de até 200 caracteres, de um arquivo seqüencial em um dispositivo externo como, por exemplo, de um cassete.

Você pode ler espaços e vírgulas, bem como caracteres, de um cassete ou de outro dispositivo externo.

Sintaxe

LINE INPUT# <número arquivo> , <var-string>
<número arquivo> é o número do arquivo aberto com OPEN.

Exemplos

```
LINE INPUT#1, A$
```

Pontos a serem lembrados

LINE INPUT# lê tudo até o retorno de carro.

É útil quando cada linha do arquivo foi dividida em campos.

Um programa BASIC armazenado em códigos ASCII pode ser lido, ainda, linha por linha, em um programa utilizando LINE INPUT#.

Corrigindo erros

Você pode utilizá-lo somente com variáveis string. Caso tente utilizá-lo com uma variável numérica, obterá um erro de desencontro de tipo.

Ocorrerá um erro de falta de espaço para string se a string digitada for grande demais para caber na área de trabalho de string (o máximo inicial é de 200 caracteres).

Comando associados e referências

INPUT
INPUT#
INPUTS(#)
OPEN
LINE INPUT

Parte 2, Capítulo 47 (Vol. 1): "Como Usar Arquivos".

LIST

Descrição

Este comando lista as linhas do programa em BASIC contido na memória do computador. Você pode listar todo o programa ou parte dele.

Sintaxe

LIST	Lista todo programa.
LIST<linha>	Lista a linha especificada.
LIST<linha 1> – <linha 2>	Lista o programa da linha 1 à linha 2.
LIST<linha> –	Lista da linha especificada até o fim do programa.
LIST – <linha>	Lista do começo do programa até a linha especificada.
LIST	Lista a última linha referida pelo computador.

Exemplos

LIST 100-200	Lista da linha 100 à 200 inclusive.
LIST –100	Lista até e inclusive a linha 100.

Pontos a serem lembrados

A tecla de função F4 é definida como "LIST" quando o computador é ligado.

Todos os caracteres em minúsculas, excluindo aqueles entre aspas, serão convertidos para maiúsculas na listagem do programa.

Comandos associados e referências

LLIST

Parte 1, Capítulo 7 (Vol. 1): "Comandos Úteis e Dicas para Escrever Programas".

Parte 2, Capítulo 29 (Vol. 1): "Edição Avançada de Programas".

LLIST

Descrição

Este comando lista o programa em BASIC contido na memória do computador para a impressora. Você pode listar todo o programa ou parte dele.

Sintaxe

LLIST	Imprime todo o programa.
LLIST <linha>	Imprime a linha especificada.
LLIST <linha 1> – <linha2>	Imprime o programa da linha 1 até a linha 2.
LLIST <linha> –	Imprime da linha especificada ao fim do programa.
LLIST – <linha>	Imprime do começo do programa à linha especificada.
LLIST	Imprime a última linha referida pelo computador.

Exemplos

LLIST 100-200	Imprime da linha 100 à 200 inclusive.
LLIST –100	Imprime até, e inclusive, a linha 100.

Pontos a serem lembrados

Todos os caracteres em minúsculas excluindo aqueles entre aspas, serão convertidos em maiúsculas durante a listagem do programa.

Corrigindo erros

Caso a impressora não esteja ligada, nada acontecerá.

Comandos associados e referências

LIST

Parte 2, Capítulo 51 (Vol. 1): "Periféricos".

LOAD

Descrição

Este comando carrega um arquivo em BASIC, que tenha sido armazenado com o comando SAVE em formato ASCII, de um dispositivo especificado.

LOAD tem uma opção de autoprocessamento, R, que deixará o arquivo aberto e processará o programa logo depois de carregado.

Sintaxe

LOAD "<nome-dispos>"	Carrega o primeiro arquivo BASIC encontrado, armazenado no formato ASCII.
LOAD"<nome-dispos> , <nome-arq>"	Carrega um arquivo BASIC com o nome especificado.
LOAD"<nome-dispos> , <nome-arq>",R	Carrega um arquivo BASIC com o nome especificado, depois processa-o.
LOAD"<nome-arq>" <nome-dispos> = CAS: no momento	Carrega um arquivo BASIC da unidade de disco.

Exemplos

```
LOAD"CAS:GAME",R
```

Automaticamente carrega e processa o programa GAME, armazenado em código ASCII.

Pontos a serem lembrados

Quando LOAD é executado, o computador fecha todos os arquivos abertos e inicia o carregamento de um novo programa BASIC que apagará o programa anterior da memória. <CTRL> <Z> é tratado como EOF, isto é, fim de arquivo.

Corrigindo erros

Não carregue programas e arquivos em códigos de máquina utilizando LOAD; em seu lugar, utilize BLOAD.

Comandos associados e referências

SAVE
CLOAD
CSAVE
BLOAD
BSAVE

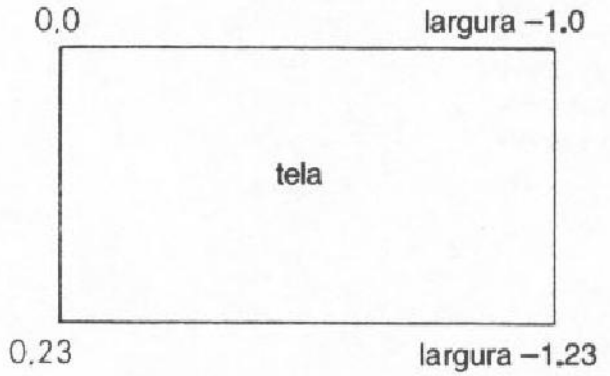
Parte 2, Capítulo 39 (Vol. 1): "Gravando no e Carregando do Cassete".

LOCATE

Descrição

LOCATE movimenta o cursor até a posição especificada. Tem ainda a capacidade de ligar e desligar o cursor. Este comando em geral é utilizado juntamente com o PRINT para imprimir texto em uma determinada parte do modo de tela de texto.

Você pode movimentar o cursor utilizando o LOCATE para qualquer ponto dentro do limite imposto pelo modo texto e pelo comando WIDTH.



A largura inicial para os dois modos de texto é:

MODO 0: WIDTH = 40 (largura máxima = 40)
MODO 1: WIDTH = 29 (largura máxima = 32)

Sintaxe

LOCATE<coord-x> , <coord-y>	Move o cursor para as coordenadas especificadas.
LOCATE<coord-x> , <coord-y>, 0	Move o cursor para as coordenadas especificadas, mas desativa o cursor.
LOCATE<coord-x> , <coord-y>, 1	Move o cursor para as coordenadas especificadas e liga o cursor.

<coord-x> e <coord-y> podem ser variáveis, constantes ou expressões numéricas, mas têm de estar dentro dos limites do modo de tela. Todos os números reais são truncados para números inteiros.

Exemplo

```
10 FOR Y=0 TO 6
20 FOR X=0 TO 20
30 LOCATE X,Y
40 PRINT"E"
50 NEXT:NEXT
60 PRINT "Quantidade de Ys"
```

```
RUN
YYYYYYYYYYYYYYYYYYY
YYYYYYYYYYYYYYYYYYY
YYYYYYYYYYYYYYYYYYY
YYYYYYYYYYYYYYYYYYY
YYYYYYYYYYYYYYYYYYY
YYYYYYYYYYYYYYYYYYY
YYYYYYYYYYYYYYYYYYY
Quantidade de Ys
```

Pontos a serem lembrados

LOCATE não funciona no modo gráfico.

LOCATE pode ser utilizado para posicionar um comando INPUT na tela.

Existe um comando semelhante, TAB, que é utilizado na forma PRINTTAB, mas o seu uso é muito restrito.

Corrigindo erros

Assegure-se de que as coordenadas não estão fora dos limites do modo de tela, se estiverem, isso pode resultar em um erro de chamada ilegal de função.

Comandos associados e referências

PRINT
INPUT
TAB

Parte 1, Capítulo 9 (Vol. 1): "Mais sobre o Comando PRINT e o Modo de Tela".

LOG

Descrição

LOG devolve o logaritmo natural do número em precisão dupla.

Sintaxe

```
LOG(<const-num>)
LOG(<var-num>)
LOG(<exp-num>)
```

Exemplos

```
PRINT LOG(10)
  2.302585092994
```

Pontos a serem lembrados

Este é o logarítmo natural na base *e*, que é diferente do logaritmo na base 10. O argumento tem de ser maior que 0.

Corrigindo erros

Um erro de chamada ilegal de função será emitido caso o argumento seja negativo.

Comandos associados e referências

EXP

Parte 1, Capítulo 19 (Vol. 1): "Funções Matemáticas".

LPOS POSIÇÃO NA LINHA

Descrição

LPOS devolve a posição da cabeça de impressão conforme se encontra no buffer da impressora.

LPOS requer uma variável X sem significado algum.

LPOS não representa necessariamente a posição física da cabeça de impressão.

Sintaxe

```
LPOS(X)
```

Exemplos

```
PRINT LPOS(X)
```

ou

```
A=LPOS(X)
```

Pontos a serem lembrados

Você não pode estabelecer um valor para esta variável do sistema.

Comandos associados e referências

Parte 2, Capítulo 51 (Vol. 1): "Periféricos".

LPRINT

Descrição

LPRINT é o comando "PRINT" para imprimir pela impressora, e é basicamente igual ao PRINT.

Sintaxe

```
LPRINT
```

Exemplo

```
LPRINT"isto é um teste"; 123
```

imprimirá "isto é um teste 123" na impressora.

Pontos a serem lembrados

Lembre-se de ligar a impressora!

Comandos associados e referências

PRINT
LPRINT USING

Parte 2, Capítulo 51 (Vol. 1): "Periféricos".

LPRINT USING

Descrição

LPRINT USING é o mesmo que PRINT USING, exceto que ele imprime em uma impressora. Veja PRINT USING, uma vez que são funções idênticas.

Sintaxe

Veja PRINT USING.

Exemplos

```
LPRINT USING "EU AMO MEU PEQUENO HÁ MUITO";"MSX"
```

imprimirá

```
EU AMO MEU PEQUENO HÁ MUITO MSX
```

Pontos a serem lembrados

Não se esqueça de ligar a impressora quando estiver utilizando este comando.

Corrigindo erros

Veja PRINT USING.

Comandos associados e referências

LPRINT
PRINT USING

Parte 2, Capítulo 36 (Vol. 1): "Imprimindo com o USING".
Parte 2, Capítulo 51 (Vol. 1): "Periféricos".

MAXFILES MÁXIMO NÚMERO DE ARQUIVOS

Descrição

Este comando estabelece o número máximo de arquivos que podem estar abertos num dado momento. A faixa está entre 0 e 15. Quando MAXFILES não é estabelecido, então o computador assume MAXFILES=1. Caso MAXFILES seja 0, então apenas os comandos SAVE e LOAD podem ser executados com relação ao uso de arquivos.

Sintaxe

```
MAXFILES= const-num
MAXFILES= var-num
MAXFILES= exp-num
```

Variando entre 0 e 15.

Exemplo

```
1000 MAXFILES=5
```

Pontos a serem lembrados

MAXFILES tem o efeito de aumentar o tamanho do bloco de controle na memória. O bloco de controle de arquivos é utilizado como área de trabalho quando você estiver acessando periféricos ou utilizando arquivos.

Corrigindo erros

O erro mais comum associado com esta função é esquecer o S em MAXFILES.

Caso o argumento esteja fora de faixa, resultará um erro de chamada ilegal de função.

Comandos associados e referências

OPEN
CLOSE

Parte 2, Capítulo 47 (Vol. 1): "Como Usar Arquivos".

MERGE JUNTAR DOIS PROGRAMAS

Descrição

Este comando permite que você junte dois programas BASIC em um só.

Vamos supor que você tenha dois programas em BASIC (programa 1 e programa 2) e queira juntar o programa 2 com o programa 1, de modo que o programa 2 venha depois do programa 1. Vamos imaginar que os dois são inicialmente gravados em fita cassete.

Primeiro, carregue o programa 2 e renumere-o de tal maneira que os seus números de linha sejam maiores que aqueles do programa 1. Depois grave-o em código ASCII utilizando SAVE.

Carregue o programa 1 do cassete, depois digite MERGE "programa 2" e ligue o cassete. O computador acrescentará o programa 2 ao final do programa 1.

Se alguma numeração de linha coincidir nos dois programas, o programa resultante conterá a(s) linha(s) do programa 2.

Sintaxe

MERGE"<nome-dispositivo>"
MERGE"<nome-dispositivo> <nome-arquivo>"
<nome-dispositivo> = CAS: de cassete

Exemplos

Como juntar dois programas, o 1 e o 2.

```
10 PRINT "OLA"
20 PRINT "BEM-VINDO"
30 PRINT "AO"
```

```
10 FOR I=1 TO 10
20 PRINT "msx"
30 NEXT I
```

Carregue (CLOAD) o programa 2 e depois renumere-o a partir da linha 50 (RENUM 50) de modo que, ao listá-lo, obterá o seguinte:

```
40 FOR I=1 TO 10
50 PRINT "MSX"
60 NEXT I
```

Grave o programa 2 no formato ASCII utilizando SAVE"CAS:PROG2". Agora carregue o programa 1 utilizando CLOAD.

MERGE"CAS:PROG2" vai juntar o programa 2 ao final do programa 1, de modo que ao listar o programa juntando obtém-se o seguinte:

```
10 PRINT "OLA"
20 PRINT "BEM-VINDO"
30 PRINT "AO"
40 FOR I=1 TO 10
50 PRINT "MSX"
60 NEXT I
```

Pontos a serem lembrados

Caso um número de linha exista tanto no programa inicial(1) como no segundo programa(2), o programa resultante da junção dos dois conterá a linha do segundo programa(2).

Caso o nome do arquivo seja omitido, então o próximo arquivo ASCII encontrado na fita será carregado, <CTRL> –Z informa ao computador o fim de arquivo (EOF).

Corrigindo erros

Acerte o nome do segundo programa ou você pode ter problemas.

Comandos associados e referências

SAVE

Parte 2, Capítulo 39 (Vol. 1): "Gravando no e Carregando do Cassete".

MID$

Descrição

Esta função retorna a parte do meio de uma string.

Sintaxe

MIDS(<var-string>,x, y)	Devolve a parte do meio de uma string com comprimento de y caracteres a partir da posição x.
MIDS(<var-string>,x)	Devolve o restante da string a partir da posição x.

Exemplo

MID$ é freqüentemente utilizado na separação de palavras em uma sentença. Aqui está um bom exemplo:

```
10 A$="MARTIN LUTERO KING"
20 B1=INSTR(A$," ")
30 B2=INSTR(B1+1,A$," ")
40 PRINT LEFT$(A$,B1-1)
50 PRINT MID$(A$,B1+1,B2-B1-1)
60 PRINT MID$(A$,B2+1)



RUN
MARTIN
LUTERO
KING
```

Pontos a serem lembrados

x e y têm de estar na faixa de 1 até 255.

Caso x seja maior que o comprimento da string dada, o MID$ devolverá uma string nula (" ").

Corrigindo erros

Esta função só pode ser usada com strings. Procure evitar erros de desencontro de tipo.

Comandos associados e referências

RIGHT$
LEFT$
LEN
INSTR

Parte 1, Capítulo 14 (Vol. 1): "Manipulando Strings".

MOD RESTO (MÓDULOS)

Descrição

Esta função efetua operações aritméticas de módulos, isto é, o operador MOD fornece o resto após uma divisão inteira.

```
10 MOD3 = 1
```

porque

10/3 é 3 e o resto é 1.

Sintaxe

<var-num> = <numérico> MOD <numérico>

Exemplos

```
10 PRINT 15 MOD 5
```

você obtém 0.

Pontos a serem lembrados

Os operandos são truncados em inteiros antes da operação.

Corrigindo erros

Ocorrerá um erro de desencontro de tipo se a função for utilizada com uma variável string.

Ocorrerá um erro de extravasamento de memória quando o argumento estiver fora da faixa dos inteiros (–32768 até 32767).

Comandos associados e referências

\(Divisão)

Parte 2, Capítulo 32 (Vol. 1): "Expressões e Operadores".

MOTOR

Descrição

Este comando permite que você utilize o cassete com controle remoto, desde que o soquete de controle remoto esteja ligado no seu computador e ao seu cassete.

MOTOR refere-se ao motor do cassete.

Sintaxe

MOTOR	Aciona a chave do motor para ON se em OFF, OFF se em ON.
MOTOR ON	Liga o motor.
MOTOR OFF	Desliga o motor.

Pontos a serem lembrados

Este comando é útil apenas se você tiver um cassete com controle remoto.

Comandos associados e referências

Veja o capítulo sobre o cassete.

NEW

Descrição

NEW cancela todo programa contido na memória do computador e reinicializa todas as variáveis e vetores. Você não pode recuperar o programa antigo depois que digitar NEW.

A sua finalidade é preparar o computador para receber um novo programa.

Sintaxe

NEW

Exemplo

```
NEW
```

Pontos a serem lembrados

NEW não limpa o modo de tela.

Utilize sempre NEW antes de recuperar outro programa. Se não o fizer o programa texto vai se misturar com aquele que estiver sendo recuperado.

Corrigindo erros

É fatal a utilização do NEW por engano!

Comandos associados e referências

Parte 1, Capítulo 3 (Vol. 1): "Escrevendo um Programa".

NEXT

Descrição

NEXT é usado com o comando FOR e informa ao computador para que volte ao comando FOR correspondente, a não ser que o loop FOR-NEXT esteja na volta final.

Para maiores informações sobre o loop FOR/NEXT, veja FOR.

Sintaxe

NEXT
NEXT<variável>
NEXT<variável> , <variável>,... lista de variáveis

Exemplo

```
10 FOR I=1 TO 9
20 PRINT I;
30 NEXT I

RUN
 1  2  3  4  5  6  7  8  9
```

Pontos a serem lembrados

A <variável> depois do NEXT pode ser omitida.

Corrigindo erros

O que segue é um engano comum: NEXT I e NEXT J encontram-se em ordem inversa.

```
10 FOR I=1 TO 5
20 FOR J=1 TO 10
30 PRINT I,J
40 NEXT I
50 NEXT J
```

LINHA 40 NEXT WITH OUT FOR.

Comandos associados e referências

FOR
TO
STEP

Parte 1, Capítulo 5 (Vol. 1): "O Uso de Loops".

NOT

Descrição

É utilizado no teste de condição de IF/THEN e nas operações booleanas. Devolve o oposto, isto é, NOT (VERDADEIRO) fornece (FALSO).

Tabela da verdade para NOT:

NOT	X	NOTX
	0	1
	1	0

Sintaxe

<var-num> =NOT(<numérico>)
IF NOT(<condição>)·THEN

Exemplos

```
A=0
PRINT NOT(A)
```

fornecerá –1. Por quê? Bem, A = &B0000000000000000 em binário. NOT (A) torna-se &B1111111111111111 que é –1 em decimal.

```
IF NOT (A=100AND B> 900) THEN GOTO 10000.
```

Pontos a serem lembrados

Os argumentos têm de estar dentro do limite de –32768 a 32767. Qualquer parte fracionária de um número real será truncada.

Comandos associados e referências

IF
THEN
ELSE
AND
OR
XOR
IMP
EQV

Parte 2, Capítulo 34 (Vol. 1): "Álgebra Booleana I: Expressões Lógicas".
Parte 2, Capítulo 35 (Vol. 1): "Álgebra Booleana II: o IF/THEN/ELSE".

OCT$ STRING OCTAL

Descrição

OCT$ fornece uma string que representa o valor em octal do argumento dado em decimal.

Octal é o sistema numérico que utiliza oito como base ou raiz.

O sistema octal utiliza apenas os dígitos 0, 1, 2, 3, 4, 5, 6, 7, e cada posição de dígito representa uma potência de oito.

O argumento tem de ser uma expressão numérica entre –32768 e 65535.

Sintaxe

<var-string> = OCT$(<numérico>)

Exemplos

```
10 PRINT OCT$(8)
20 PRINT OCT$(&HF)
30 PRINT OCT$(8^4)
40 PRINT OCT$(8^4-1)
```

```
10
17
10000
7777
```

Pontos a serem lembrados

Caso o argumento seja negativo, será utilizada a forma complemento de dois, isto é, OCT$(–x)=OCT$(65536 –x).

Corrigindo erros

Resultará num erro "Overflow" (extravasamento) se o argumento estiver fora da faixa especificada anteriormente.

Comandos associados e referências

BIN$
HEX$

Parte 2, Capítulo 33 (Vol. 1): "Introdução aos Sistemas Numéricos Usados no MSX".

ON <EXPRESSÃO> GOSUB

Descrição

O comando ON <expressão> GOSUB <linha1>, <linha2>, <linha3>, ... oferece uma múltipla escolha de sub-rotinas. A opção depende da <expressão> avaliada: se <expressão> devolve 1, o programa vai (GOSUB) para a sub-rotina da <linha1>; caso devolva 2, vai para a <linha2>, e assim por diante.

A <expressão> tem de devolver um número entre 1 e o número de sub-rotinas que são apontadas no comando.

Por exemplo:

```
ON X GOSUB 100, 250, 670, 800
```

Se X é 1 então ele vai para a sub-rotina da linha 100.

Se X é 2 então ele vai para a sub-rotina da linha 250.

Se X é 3 então ele vai para a sub-rotina da linha 670.

Se X é 4 então ele vai para a sub-rotina da linha 800.

Caso X seja 0 ou maior que o número de sub-rotinas listadas, isto é, maior que 4 no exemplo dado, então o BASIC continua no comando seguinte.

Sintaxe

ON <expressão> GOSUB <lista de números de linhas>

Exemplos

```
10 INPUT "1,2 OU 3";I
20 ON I GOSUB 40,60,80
30 GOTO 10
40 PRINT "SUB-ROTINA 1"
50 RETURN
60 PRINT "SUB-ROTINA 2"
70 RETURN
80 PRINT "SUB-ROTINA 3"
90 RETURN
```

```
1,2 ou 3? 2
SUB-ROTINA 2
1,2 ou 3? 1
SUB-ROTINA 1
1,2 ou 3? 3
SUB-ROTINA 3
```

Pontos a serem lembrados

Caso a expressão avaliada seja um número real, a parte fracionária será truncada, isto é, se X = 1.22 então X = 1 e o primeiro GOSUB é executado.

Corrigindo erros

Caso a expressão tenha como resultado um número negativo ou maior que 255, resultará num erro de chamada ilegal de função ("Illegal function call").

Comandos associados e referências

ON ... GOTO

Parte 1, Capítulo 18 (Vol. 1): "Desvios Condicionais".

ON <EXPRESSÃO> GOTO

Descrição

No comando ON <expressão> GOTO <linha1>, <linha2>, <linha3>, ... é oferecida múltipla escolha de desvios GOTO. A opção depende do valor da <expressão> avaliada: se a <expressão> devolver 1, o programa irá (GOTO) para <linha1>, caso devolva 2, irá para a <linha2>, e assim por diante.

A expressão tem de devolver um valor entre um e um número que seja menor, ou igual ao número de números de linha listados no comando.

Por exemplo:

```
ON X GOTO 100, 250, 670, 800
```

Se X é 1 então ele vai para a linha de número 100.

Se X é 2 então ele vai para a linha de número 250.

Se X é 3 então ele vai para a linha de número 670.

Se X é 4 então ele vai para a linha de número 800.

Se X vale zero ou mais que o número de linhas listadas, isto é, mais que 4 no exemplo acima, então o BASIC continua até o comando da linha seguinte.

Sintaxe

ON<expressão> GOTO <lista dos números das linhas>

Exemplos

```
10 INPUT "QUANTAS VEZES";N
20 ON N GOTO 60,50,40
30 GOTO 10
40 PRINT "MSX"
50 PRINT "MSX"
60 PRINT "MSX"
```

```
RUN
QUANTAS VEZES? 3
MSX
MSX
MSX
```

Pontos a serem lembrados

Caso o valor da <expressão> seja um número real, a parte fracionária será truncada, isto é, se C = 1.22, então X = 1 e o desvio é para a primeira linha da lista de números de linhas.

Corrigindo erros

Caso a <expressão> retorne um número negativo ou maior que 255, isso resultará um erro de chamada ilegal de função ("Illegal function call").

Comandos associados e referências

ON...GOSUB

Parte 1, Capítulo 18 (Vol. 1): "Desvios Condicionais".

ON ERROR GOTO

Descrição

ON ERROR GOTO permite detectar erros e especificar para que linha o programa deve desviar assim que um erro for detectado. Assim que o computador for comandado para detectar erros, ele pulará para a linha especificada quando ocorrer um erro durante o processamento do programa.

Utilizando o comando ERROR, você pode criar seu próprio erro. Existem cerca de 36 erros entre os códigos de erro 1 e 60 no BASIC. Códigos entre 60 e 255 estão disponíveis para sua utilização com o comando ERROR.

Para programar seus próprios erros, você terá de, primeiramente, colocar o computador no modo para detectar erros executando o comando ON ERROR GOTO <linha> no início do seu programa. Assim, se no meio da execução do programa aparecer uma condição que faça com que você queira chamar a sua rotina para detectar erros, poderá fazê-lo assim:

IF <condição> THEN ERROR <código de erro>

ON ERROR GOTO será ativado e o computador desviará imediatamente para aquela sub-rotina de manipulação de erro. Você tem de criar uma linha na sub-rotina dizendo:

IF ERR=<código de erro> THEN PRINT "mensagem de erro"

A rotina para detectar erros poderá terminar a execução do programa ou continuar a execução a partir de qualquer ponto do programa.

Sintaxe

ON ERROR GOTO<linha>

Exemplo

Uma armadilha simples para detectar um erro:

```
10 ON ERROR GOTO 1000
20 PRINT "MSX"
30 PRONT "ERROR"
40 END
1000 PRINT "HA UM ERRO NA LINHA";ERL
1010 END
```

LINHA 30 Há um erro neste programa:

```
MSX
HA UM ERRO NA LINHA 30
```

Pontos a serem lembrados

O computador MSX não vai detectar erros enquanto estiver executando a sub-rotina para pegar erros. Caso haja um erro dentro de uma rotina para detectar erros, o BASIC pára e apresenta a mensagem de erro.

Não é possível detectar todos os erros com rotinas para pegar erros. É, portanto, recomendável encerrar a rotina para detectar erros com ON ERROR GOTO 0, que pára a execução do programa BASIC e apresenta a mensagem de erro.

Assim que a rotina estiver ativada, o computador desviará para a sub-rotina para detectar erros, mesmo que você esteja no modo direto.

Quando estiver definindo seus próprios erros, comece de 255 e vá descendo.

Se o computador encontrar um erro, sem uma mensagem de erro predefinida, imprimirá "Unprintable Error" (Erro impossível de ser impresso).

ON ERROR desativa qualquer desvio baseado em eventos como ON INTERVAL e ON STRIG.

Corrigindo erros

Se você esquecer o número da linha, obterá um erro de número de linha não definida.

Comandos associados e referências

ERL
ERR
ERROR
RESUME

Parte 2, Capítulo 38 (Vol. 1): "Tratamento de Erros".

ON INTERVAL GOSUB

Descrição

ON INTERVAL estabelece a sub-rotina de interrupção por intervalos de tempo em um programa BASIC. Quando o intervalo de interrupção é ativado (com INTERVAL ON), o computador desviará, a cada intervalo de tempo, para a sub-rotina especificada. (Veja mais sobre a sua aplicação, na seção de Manipulação de eventos e interrupções.)

Os intervalos estão calculados em 1/60 de segundos e podem ter qualquer duração.

Assim que a interrupção ocorre, é executado, automaticamente, um INTERVAL STOP. Isto impede que a interrupção por tempo ocorra durante a sub-rotina de interrupção

de tempo atual. Ela lembra, no entanto, se uma interrupção ocorreu durante a sub-rotina atual e, se isto ocorreu, o computador vai imediatamente executar a sub-rotina novamente, a não ser que a sub-rotina atual desative completamente a interrupção por tempo com o comando INTERVAL OFF. Depois de executar a sub-rotina de interrupção, o computador executará, automaticamente, o comando INTERVAL ON para ativar a interrupção novamente.

Sintaxe

ON INTERVAL =<intervalo de tempo em 1/60 de segundos> GOSUB <linha>

Exemplos

```
10 ON INTERVAL=500 GOSUB 40
20 INTERVAL ON
30 GOTO 30
35 REM Sub-rotina de intervalo
40 PRINT "Ja passaram 10 segundos"
50 PRINT "FIM DO PROGRAMA"
60 END
```

LINHA 10 Estabelece o intervalo de tempo em 600 unidades (10 segundos). Sub-rotina em 40.

LINHA 20 Ativa a interrupção por tempo.

Quando processar este programa, você verá:

```
Ja passaram 10 segundos
FIM DO PROGRAMA
```

Pontos a serem lembrados

A interrupção por tempo é desativada assim que o BASIC sair do programa, ou seja, nenhuma interrupção por tempo ocorre no modo direto. Também é desativada nas sub-rotinas para detectar erros.

Corrigindo erros

Não se esqueça de ligar a interrupção por tempo com INTERVAL ON.

Comandos associados e referências

INTERVAL ON/OFF/STOP

Parte 2, Capítulo 37 (Vol. 1): "Tratamento de Intervalos e Interrupções pelo BASIC".

ON KEY GOSUB

Descrição

ON KEY GOSUB estabelece os números das linhas para as sub-rotinas de interrupção das teclas de função. KEY (<número>) ON ativa uma interrupção de tecla de função para cada uma das teclas de função detectadas pelo computador. Quando houver uma interrupção, o computador desviará para a sub-rotina especificada por ON KEY GOSUB.

No comando ON KEY GOSUB você tem de relacionar todos os números das linhas das sub-rotinas com uma tecla de função correspondente. Caso não haja nenhuma sub-rotina para uma determinada tecla de função, pule o número de linha e coloque apenas uma vírgula.

Assim que a interrupção da tecla de função ocorre, é executado um KEY (<número>) STOP automático. Isto impede que a interrupção da tecla ocorra durante a sub-rotina de interrupção atual. Ela se lembra, no entanto, se uma tecla de função foi operada durante a sub-rotina atual, e o computador vai imediatamente para a sub-rotina correspondente àquela tecla, a não ser que a sub-rotina atual desative totalmente a interrupção KEY pela execução de KEY <número> OFF. Depois de executar a sub-rotina de interrupção, o computador executará automaticamente KEY ON para ativar a interrupção novamente.

Sintaxe

ON KEY GOSUB <lista dos números das linhas>

Exemplos

Se você pressionar a tecla de função F1 o computador emitirá um som duas vezes; se pressionar F2, haverá apenas um som:

```
10 ON KEY GOSUB 50,60
20 KEY(1) ON
30 KEY(2) ON
40 GOTO 40
50 BEEP
60 BEEP
70 RETURN
```

LINHA 10 ESTABELECE A SUB-ROTINA DA TECLA F1 NA LINHA 50 E 52 E DA TECLA F2 NA LINHA 60.
LINHA 20 ATIVA A TECLA DE FUNÇÃO F1.
LINHA 30 ATIVA A TECLA DE FUNÇÃO F2.
LINHA 40 UM LOOP INFINITO PARA AGUARDAR UMA TECLA DE FUNÇÃO.
LINHA 50 SOM (PARA F1).
LINHA 60 SOM (PARA AMBOS).
LINHA 70 RETORNA PARA ONDE VEIO; NESTE CASO, À LINHA 40.

Pontos a serem lembrados

A interrupção de teclas é desativada tanto quando o programa não é processado, quanto durante as rotinas para detectar erros.

Corrigindo erros

Você terá um erro de número de linha indefinido se colocar um número de linha inexistente.

Não se esqueça do KEY ON.

Comandos associados e referências

KEY ON/OFF/STOP

Parte 2, Capítulo 37 (Vol. 1): "Tratamento de Intervalos e Interrupções pelo BASIC".

ON SPRITE GOSUB

Descrição

ON SPRITE GOSUB estabelece a sub-rotina de interrupção para colisão de sprites. O comando SPRITE ON ativa uma rotina que desvia o programa para a sub-rotina especificada por ON SPRITE GOSUB quando dois sprites colidem.

Assim que a interrupção ocorre, é executado um SPRITE STOP automático. Isto impede que ocorra uma outra interrupção durante a sub-rotina atual. Ela se lembra, entretanto, se houve outra colisão de sprites durante a sub-rotina e, neste caso, o computador executará a sub-rotina novamente, assim que deixar a sub-rotina atual, a não ser que a sub-rotina atual desative completamente a interrupção de colisão de sprites pela execução de SPRITE OFF. Depois de deixar a sub-rotina de interrupção, o computador executa um SPRITE ON para ativar a interrupção novamente.

Sintaxe

ON SPRITE GOSUB<linha>

Exemplo

Neste exemplo você verá dois sprites quadrados, um amarelo e o outro branco, se aproximando de lados opostos do modo de tela. Quando colidem, o ON SPRITE

GOSUB começa a atuar fazendo o desvio para a sub-rotina de interrupção de colisão de sprites. O SPRITE OFF na rotina de interrupção de colisão de sprites impede qualquer detecção adicional de colisão de sprites (tente esta sub-rotina sem SPRITE OFF e veja o que acontece).

```
10 ON SPRITE GOSUB 110
20 SCREEN 2,0
30 SPRITE$(0)=STRING$(8,CHR$(255))
40 SPRITE$(1)=STRING$(8,CHR$(255))
50 SPRITE ON
60 FOR I=10 TO 240
70 PUT SPRITE 0,(I,100),11,0
80 PUT SPRITE 1,(250-I,100),15,1
90 NEXT I
100 END
105 'Sub-rotina de colisao de sprites
110 SPRITE OFF
120 BEEP
130 RETURN
```

LINHA 10 COLOCA A SUB-ROTINA EM 110.
LINHA 20 ESTABELECE O MODO GRÁFICO DE TELA.
LINHA 30 O SPRITE 0 É UM QUADRADO.
LINHA 40 ASSIM TAMBÉM É O SPRITE 1.
LINHA 50 LIGA A ROTINA DE DETECÇÃO DE COLISÃO DE SPRITES.
LINHA 60 LOOP.
LINHA 70 O SPRITE (AMARELO) SE MOVE A PARTIR DA ESQUERDA.
LINHA 80 O SPRITE (BRANCO) SE MOVE A PARTIR DA DIREITA.
LINHA 90 PRÓXIMO LOOP.
LINHA 100 TÉRMINÓ DO PROGRAMA.
LINHA 110 DESLIGA A ROTINA (UMA VEZ É SUFICIENTE).
LINHA 120 ACIONA O SOM.
LINHA 130 RETORNA AO PONTO EM QUE O COMPUTADOR SAIU.

Pontos a serem lembrados

A interrupção SPRITE é desativada quando o programa não estiver sendo processado e também durante as rotinas de detecção de erros.

Corrigindo erros

Você obtém um erro de número de linha não definido se fornecer um número de linha inexistente para a sub-rotina de interrupção.

Não se esqueça de usar o SPRITE ON.

Comandos associados e referências

SPRITE ON/OFF/STOP
SPRITE$
PUT SPRITE

Parte 2, Capítulo 43 (Vol. 1): "Gráficos Avançados IV".

ON STOP GOSUB

Descrição

ON STOP GOSUB é um comando de manuseio de interrupções que impede ao usuário de irromper um programa em processamento pela operação das teclas <CTRL> <STOP>.

A única maneira de se sair de um programa "bloqueado" é a inicialização (RESET) do computador. Portanto, você deve se lembrar de armazenar o seu programa "bloqueado" antes de executá-lo (RUN).

STOP ON/OFF/STOP ativa/desativa o desvio pela tecla <STOP>. Quando o STOP ON é executado, o BASIC começa a verificar se <CTRL> <STOP> foram pressionados a cada vez que executa um novo comando.

Se um <CTRL> <STOP> foi detectado o BASIC é desviado à sub-rotina especificada no comando ON STOP GOSUB executado anteriormente no programa.

Assim que a interrupção ocorrer, será executado um STOP STOP automático. Isto impede que a interrupção ocorra novamente durante a sub-rotina atual. Ele se lembra, no entanto, se <CTRL> <STOP> foi pressionado novamente durante a sub-rotina atual, quando o computador irá à sub-rotina depois de ter deixado a sub-rotina atual (a não ser que a sub-rotina atual desative completamente a interrupção STOP pela execução de STOP OFF). Depois de deixar a sub-rotina de interrupção, o computador executará automaticamente um STOP ON para ativar a interrupção novamente.

Sintaxe

ON STOP GOSUB <linha>

Exemplo

Este programa mostra como o ON STOP GOSUB impede que o usuário interrompa o programa. Se você pressionar <CTRL> <STOP>, aparecerá "control stop desativado" no modo de tela e continuará a execução do programa. Esta rotina tem uma rotina de saída especial: digite <s> para sair; caso contrário, o seu MSX imprimirá indefinidamente a frase acima quando você pressionar <CTRL> <STOP>.

```
10 ON STOP GOSUB 100
20 STOP ON
30 IF INKEY$="s" THEN END
40 PRINT "MSX"
50 GOTO 30
90 REM Sub-rotina de <CONTROL><STOP>
100 BEEP
110 PRINT "<CONTROL><STOP> desativado"
120 RETURN
```

LINHA 10 A SUB-ROTINA É ESTABELECIDA NA LINHA 100.
LINHA 20 ACIONA A ROTINA DE INTERRUPÇÃO.
LINHA 30 SE <S> É DIGITADO, ENTÃO TERMINE.
LINHA 40 IMPRIME MSX.
LINHA 50 DESVIA PARA A LINHA 30.
LINHA 100 UM SOM DE AVISO.
LINHA 110 MENSAGEM.
LINHA 120 RETORNE AO PONTO EM QUE A INTERRUPÇÃO FOI DETECTADA.

Pontos a serem lembrados

Observe que o ON STOP não impede que a tecla <STOP> pare o programa. Ele apenas impede que você termine a execução de um programa com <CTRL> <STOP>. Tente pressionar apenas a tecla <STOP> na execução do programa anterior. Você verá que o programa parará e o cursor será visto. Se você pressionar a tecla <STOP> pela segunda vez, o programa continuará a ser executado.

Lembre-se de executar STOP ON para ativar o STOP. A interrupção STOP será desativada quando o programa não estiver sendo processado e, também, durante as rotinas para detecção de erros.

Corrigindo erros

Você obtém um erro de número de linha indefinido se fornecer um número de linha inexistente para a sub-rotina de interrupção STOP.

Comandos associados e referências

STOP ON/OFF/STOP

Parte 2, Capítulo 37 (Vol. 1): "Tratamento de Intervalos e Interrupções pelo BASIC".

ON STRIG GOSUB

Descrição

ON STRIG GOSUB estabelece sub-rotinas de interrupção por acionamento do *joystick* (controle de jogos) e barra de espaço. STRIG(<n>) ON ativa a rotina por acionamento e é utilizado em conjunto com ON STRIG GOSUB.

Existem cinco valores para <n> :

0= Barra de espaço.
1= botão 1 do *joystick* 1.
2= botão 1 do *joystick* 2.
3= botão 2 do *joystick* 1.
4= botão 2 do *joystick* 2.

No comando ON STRIG GOSUB, você tem de especificar os números de linha das sub-rotinas para todos os botões (exceto quando utilizar apenas a barra de espaço (veja a seção de sintaxe). Caso não haja nenhum sub-rotina para um determinado botão, você poderá omiti-la digitando uma vírgula.

Assim que a interrupção ocorrer, será executado um STRIG (<n>) STOP automático (<n> refere-se ao botão). Isto impede que outra interrupção ocorra durante a execução da sub-rotina do botão atual. Ela se lembra, no entanto, se um gatilho foi pressionado durante a sub-rotina, caso no qual o computador irá imediatamente à sub-rotina correspondente deste gatilho depois que tiver deixado a sub-rotina atual (a não ser que a sub-rotina atual desative a interrupção do gatilho pela execução de STRIG(<n>) OFF). Depois de deixar a sub-rotina de interrupção, o computador executará um STRIG(<n>)ON para ativar a interrupção novamente.

Sintaxe

ON STRIG GOSUB <lista dos números das linhas>.
ON STRIG GOSUB <linha> apenas quando a barra de espaço é utilizada sozinha.

Exemplo

Aqui está um programa curto que demonstra como a rotina de gatilho funciona com a barra de espaço. Quando a barra de espaço é pressionada, o programa desvia para a sub-rotina da interrupção. Caso contrário, será impressa a palavra MSX continuamente até que a tecla "s" seja pressionada.

```
10 ON STRIG GOSUB 100
20 STRIG(0) ON
30 IF INKEY$="s" THEN END
40 PRINT "MSX"
50 GOTO 30
90 REM Rotina da barra de espaco
100 BEEP
110 PRINT "Precionaste o espaco?"
120 RETURN
```

LINHA 10 ESTABELECE A ROTINA DE BOTÃO NA LINHA 100.

LINHA 20 LIGA A DETECÇÃO.

LINHA 30 SE <S> É TECLADO, ENTÃO ENCERRE A EXECUÇÃO DO PROGRAMA.

LINHA 40 IMPRIME MSX.

LINHA 50 DESVIA PARA A LINHA 30.

LINHA 100 SOM DE AVISO.

LINHA 110 FORNECE UMA MENSAGEM.

LINHA 120 RETORNA AO LUGAR EM QUE A INTERRUPÇÃO FOI DETECTADA.

```
MSX
MSX
MSX
MSX
MSX
MSX
MSX
MSX
MSX
Precionaste o espaco?
MSX
MSX
MSX
MSX
MSX
MSX
MSX
MSX
MSX
MSX
MSX
MSX
MSX
```

Pontos a serem lembrados

A interrupção STRIG é desativada quando o programa não estiver sendo processado e também durante as rotinas de detecção de erros.

Corrigindo erros

Um erro comum:

```
STRIG (0) ON ...
```

Isto está errado e causa um erro de sintaxe. Não deve haver espaço entre STRIG e (0).

Você obtém um erro de número de linha indefinido se colocar um número de linha inexistente na <lista dos números das linhas>.

Comandos associados e referências

```
STRIG ( )
STRIG ON/OFF/STOP
```

Parte 2, Capítulo 37 (Vol. 1): "Tratamento de Intervalos e Interrupções pelo BASIC".

OPEN

Descrição

OPEN declara aberto um determinado dispositivo. Aloca um buffer para entrada/saída (E/S) e estabelece o modo de operação E/S para o buffer. Se você estiver

utilizando um arquivo para E/S, primeiro terá de tê-lo aberto. Você tem de executar OPEN antes dos seguintes comandos:

PRINT#	PRINT#USING
INPUT#	LINE INPUT#
INPUT$(#)	EOF

Existem quatro dispositivos atendidos pela versão atual do MSX <descritor de dispositivo>:

CAS: cassete
CRT: modo de tela de texto
GRP: modo gráfico de tela
LPT: impressora de linha

O descritor de dispositivo pode ser ampliado utilizando um cartucho ROM. Não se esqueça dos dois-pontos após o descritor de dispositivo. Existem três modos de E/S:

OUTPUT	Especifica modo de saída seqüencial.
INPUT	Especifica modo de entrada seqüencial.
APPEND	Especifica modo anexação seqüencial.

<número de arquivo> é um inteiro cujo valor está entre um e MAXFILES, o número máximo de arquivos. Números de arquivos também são colocados nos comandos PRINT# etc. O <número de arquivo> é associado ao arquivo desde que este não seja fechado (CLOSE).

Sintaxe

OPEN"<descritor de dispositivo> [<nome do arquivo>] "[FOR<modo>] AS[#] <número do arquivo> [<nome do arquivo>], [FOR<modo>], e [#] são opcionais.

Exemplos

Para escrever caracteres no modo gráfico de tela, é necessário abrir um arquivo

para o GRP. Depois, poderá utilizar um PRINT# para imprimir no modo gráfico de tela na posição do cursor gráfico.

```
10 OPEN "GRP:" FOR OUTPUT AS #1
20 SCREEN 2
30 DRAW "BM45,145"
40 PRINT #1,"ESCRITURA NO MODO 2"
50 GOTO 50
```

LINHA 10 ABRE UM ARQUIVO PARA MODO GRÁFICO DE TELA.
LINHA 20 MODO GRÁFICO DE TELA.
LINHA 30 APONTA PARA 30, 145.
LINHA 40 PRINT#1.
LINHA 50 RETÉM MODO GRÁFICO.

Pontos a serem lembrados

MAXFILES é 1 em default, mas se estiver utilizando um número de dispositivos em um programa é conveniente colocar MAXFILES em um número maior.

Corrigindo erros

Caso o número do arquivo seja maior que aquele estabelecido por MAXFILES, então resultará em uma mensagem de erro.

Comandos associados e referências

MAXFILES
PRINT#
PRINT#USING
INPUT#
LINE INPUT#
INPUT$
VARPTR
EOF

Parte 2, Capítulo 42: "Gráficos Avançados III".
Parte 2, Capítulo 47: "Como Usar Arquivos".

OR OPERADOR LÓGICO OR

Descrição

OR é um dos operadores lógicos que executam manipulação de bit e álgebra booleana. É utilizado também em IF/THEN/ELSE para testar mais que uma condição, antes que uma ação resultante seja tomada.

A tabela da verdade para OR é:

OR	X	Y	X OR Y
	0	0	0
	1	0	1
	0	1	1
	1	1	1

Exemplo 34 OR 67

	34	0000000000100010
OR	67	0000000001000011
	99	0000000001100011

OR é utilizado nos comandos IF/THEN/ELSE e fornece uma escolha múltipla das possíveis condições.

Sintaxe

IF<condição>OR<condição> ... THEN...ELSE<var-num>=
<numérico>OR<numérico>

Exemplos

Álgebra booleana:

```
10 A=134
20 B=213
30 PRINT "A      ";A;BIN$(A)
40 PRINT "B      ";B;BIN$(B)
45 PRINT"        --- --------"
50 PRINT "A OR B";A OR B;BIN$(A OR B)
```

```
RUN
A       134 10000110
B       213 11010101
        --- --------
A OR B 215 11010111
```

IF<condição>OR<condição> THEN

```
10 INPUT "A=";A
20 INPUT "B=";B
30 IF A>100 OR B>100 THEN PRINT "ou A ou B e maior que
 100"
40 END
```

```
A=? 56
B=? 148
ou A ou B e maior que 100
```

Pontos a serem lembrados

Os operandos booleanos têm de estar na faixa dos inteiros de –32768 e 32767.

Os números reais são truncados em inteiros.

Corrigindo erros

Resultará um erro de extravasamento quando o número estiver fora da faixa de inteiros. Ocorre desencontro de tipo se o operando é uma string.

Comandos associados e referências

IF
THEN
ELSE
AND
NOT
IMP
XOR
EQV

Parte 1, Capítulo 6 (Vol. 1): "O Uso de Condições".
Parte 2, Capítulo 34 (Vol. 1): "Álgebra Booleana I: Expressões Lógicas".
Parte 2, Capítulo 35 (Vol. 1): "Álgebra Booleana II: o IF/THEN/ELSE".

OUT

Descrição

Este comando envia um byte para uma determinada porta de saída. Não utilize este comando em software comercial, uma vez que será dependente da máquina e perderá a compatibilidade MSX.

Sintaxe

OUT<porta> , <data>
<porta> = número da porta entre 0 e 255
<data> = byte de dado entre 0 e 255

Exemplos

```
OUT 1, 166
```

Pontos a serem lembrados

Utilize entradas da BIOS em lugar de OUT, para não perder a compatibilidade MSX.

Corrigindo erros

Qualquer número negativo ou maior que 255 não tem sentido.

Comandos associados e referências

WAIT
INP

PAD PAD DE TOQUE

Descrição

MSX pode sustentar dois "PADs" de toque ligados às portas do *joystick*. PAD retorna os valores de status dos atenuadores de toque.

Sintaxe

PAD(<n>

n=0 a 3 para atenuadores de toque ligados ao *joystick* da porta 1.

PAD(0) = − 1 se tocado.
0 se não tocado.

PAD(1) =coordenada–x(0–255) no pad 1.
PAD(2) =coordenada–y(0–255) no pad 1.
PAD(3) = −1 se a chave do atenuador de toque é operado.
0 se a chave do atenuador de toque é operado.
n=4 a 7 para o atenuador de toque ligado ao *joystick* porta 2.
PAD(4)=−1 se tocado.
0 se não tocado.

PAD(5)=coordenada–x(0–255) no pad 2.
PAD(6)=coordenada–y(0–255) no pad 2.
PAD(7)=−1 se a chave do atenuador de toque é operado.
0 se a chave do atenuador de toque é operado.

Exemplos

Este é um programa curto para utilizar um atenuador de toque como mesa de desenho.

```
10 SCREEN 2,0,0
20 IF NOT PAD(0) THEN 20
30 X=PAD(1)
40 Y=INT(PAD(2)*192/255)
50 PSET X,Y
60 GOTO 20
```

LINHA 10 MODO GRÁFICO DE ALTA RESOLUÇÃO.
LINHA 20 SE NÃO TIVER SIDO TOCADO, REPETE A MESMA LINHA.
LINHA 30 PEGA X.
LINHA 40 PEGA Y PARA CABER NO MODO DE TELA.
LINHA 50 DESENHA UM PONTO.

Pontos a serem lembrados

Observe que as coordenadas são válidas apenas quando PAD(0) (ou PAD(4)) estão ativos (= –1). Quando o PAD(0) é avaliado, PAD(5) e PAD(6) são afetados, e quando PAD(4) é ativado, PAD(1) e PAD(2) são afetados.

Comandos associados e referências

Parte 2, Capítulo 51 (Vol. 1): "Periféricos"

PAINT

Descrição

PAINT preenche uma área envolvida por uma linha com a cor especificada.

Você precisa especificar as coordenadas onde deseja que o computador inicie a pintura. Se estiver preenchendo uma determinada forma, então aponte as coordenadas em qualquer ponto dentro da forma.

Sintaxe

PAINT<especificador de coordenadas>
pinta na cor atual do texto.
PAINT<especificador de coordenadas> , <cor>
pinta com a cor especificada.
PAINT<especificador de coordenadas> , <cor> , <cor da linha de fronteira>
pinta com a cor especificada a área envolvida pela linha de fronteira com a cor especificada (apenas modo multicolorido).
<especificador de coordenadas>

(1) <coordenada–x> , <coordenada–y>
faixa do x = 0–255
faixa do y = 0–191

2) STEP(<desvio–x> , <desvio–y>)
Estas coordenadas são de um ponto relativo ao último ponto referido.

Notas: <desvio–x>, <desvio–y>, <coordenada–x>, <coordenada–y> podem ser <var-num>, <exp-num>, ou simplesmente <con-num>.

CÓDIGO DE CORES

0 Transparente	8 Vermelho-médio
1 Preto	9 Vermelho-claro
2 Verde-médio	10 Amarelo-escuro
3 Verde-claro	11 Amarelo-claro
4 Azul-escuro	12 Verde-escuro
5 Azul-médio	13 Roxo
6 Vermelho-escuro	14 Cinza
7 Azul-Claro	15 Branco

Exemplos

1. Modo gráfico de alta resolução (SCREEN 2)

No modo de alta resolução, a cor da fronteira tem de ser da mesma cor que a de pintura, caso contrário a tinta será "derramada". Neste exemplo nenhuma cor é especificada de modo que utiliza a cor default: significa que as linhas são brancas, em um fundo azul e depois a forma é pintada em branco.

```
10  SCREEN 2
20  PSET (100,100)
30  LINE-STEP(-50,50)
40  LINE-STEP(100,0)
50  LINE-STEP(50,-50)
60  LINE-STEP(-100,0)
70  PAINT (110,110)
100 GOTO 100
```

2. No modo 3, o modo gráfico de baixa resolução, você pode pintar com uma cor diferente da fronteira.

```
10  SCREEN 3
20  PSET (100,100)
30  LINE-STEP(-50,50)
40  LINE-STEP(100,0)
50  LINE-STEP(50,-50)
60  LINE-STEP(-100,0)
70  PAINT (110,110),9,15
100 GOTO 100
```

Pontos a serem lembrados

Você pode especificar a cor da linha de fronteira apenas no modo policromático. Pode utilizar apenas uma cor por forma pintada. Isto significa que não pode ter linhas de fronteiras multicoloridas.

Corrigindo erros

Caso as coordenadas especificadas estejam fora da tela, poderá resultar um erro de chamada à função ilegal.

Comandos associados e referências

DRAW
SCREEN

Parte 1, Capítulo 27 (Vol. 1): "Pintando".

PDL PADDLE

Descrição

PDL devolve o valor do Paddle.

Você pode ligar até 12 controladores de jogo, 6 em cada porta do *joystick*.

Sintaxe

```
PDL(<n>)
Port 1 n=1, 3, 5, 7, 9, 11
Port 2 n=2, 4, 6, 8, 10, 12
```

PDL devolve o valor entre 0 e 255.

Exemplo

```
P1 = PDL(1)
```

Comandos associados e referências

Parte 2, Capítulo 51 (Vol. 1): "Periféricos".

PEEK

Descrição

PEEK devolve um byte, lido de uma determinada posição da memória.

Sintaxe

PEEK<numérico-inteiro>

Faixa: –32768 a 65535.

Exemplos

```
10 FOR I=1 TO 10000
20 A=PEEK(I)
30 IF A>31 AND A<127 THEN PRINT HEX$(I);" ";CHR$(A
)
40 NEXT I
```

```
RUN
A &
12 &
19 E
21 j
25.^
```

Pontos a serem lembrados

POKE é a declaração complementar de PEEK.

Não podemos olhar a RAM de vídeo com isto: em seu lugar utilize VPEEK.

Comandos associados e referências

POKE
VPOKE
VPEEK

Veja a Tabela de Variáveis do Sistema.

Parte 2, Capítulo 48 (Vol. 1): "Mapa da Memória".

PLAY TOCAR MÚSICA

Descrição

PLAY toca música de acordo com a Macrolinguagem Musical que é introduzida em um formato de string de uma forma semelhante àquela da Macrolinguagem Gráfica.

A macrolinguagem musical:

A B, C, D, E, F, G com opção #, +, –.

Toca a nota indicada na oitava atual.

ou + depois da letra significa SUSTENIDO (SHARP)

– <menos> significa BEMOL (FLAT).

Observe que a utilização de #, +, – é válido apenas se tal nota existe, isto é, correspondendo à nota preta do piano.

0<n> n = 1 a 8

Selecione a oitava de 1 a 8.
Cada oitava vai de C a B.
A oitava default inicial é 4, onde a oitava, C coresponde ao "C do meio" em um piano.

N<n> n = 0 a 96

Toca a nota n. Esta é uma forma alterativa para tocar qualquer nota das 8 oitavas disponíveis.

L<n> n = 1 a 64

Estabelece a duração da nota.
L1= toda nota (semibreve)
L2= meia nota (mínima)
L4= quarto da nota (colcheia)
L8= oitavo da nota (fusa)
etc.
A duração de qualquer nota em particular poderá ser alterada com <n> depois da nota. Por exemplo, L8C é o mesmo que C8. Isto não pode ser utilizado com o comando N.

O valor default é L4.

R<n> n = 1 a 64

Pausa. A duração da pausa é feita de forma semelhante à L.

R1 = pausa uma nota inteira.
R2 = pausa de meia nota.
R4 = pausa de um quarto de nota.
R8 = pausa de um oitavo da nota.
etc.

Nota: R e R0 são iguais ao período de pausa default que é R4. .(ponto)

Um ponto depois de uma nota de A a G, N ou pausa R aumenta a duração em 3/2, isto é, tocado como uma nota com ponto.

Você pode colocar mais que um ponto depois de uma nota.

T<n> n = 32 a 255 TEMPO

O tempo estabelece o número de notas de um quarto (ou colcheias) em um minuto.

n pode variar de 32 a 255. O valor default é 120.

V<n> n = 0 a 15

Volume. V estabelece o volume de saída: O sendo 0 volume mais baixo e 15 o mais alto.

O valor default é 8.

M<n> n = 1 a 65535

Modulação. Isto estabelece o período da envoltória especificada pelo S.

O valor default é 255.

S<n> n = 0 a 15

Isto fornece a forma da envoltória. Existem diversos padrões de envoltórias predefinidas a escolher.

Veja SOUND para maiores detalhes.

x<variável>

O comando X executa aquilo que estiver na variável string como a linguagem macromusical.

Em todos os comandos acima, n pode ser uma constante numérica inteira, ou pode ser uma constante numérica inteira, ou pode ser uma variável da forma "=<variável>;" onde o nome da variável é envolvido por " = "and";".

Quando você executa um BEEP, estará repondo o sistema de som aos valores default.

Sintaxe

PLAY<exp-string para voz 1>, <exp-string para voz 2>, <exp-string para voz 3>

onde a voz 2 e 3 são opcionais.

Exemplo

```
10 A$="CDEFGAB"
20 FOR I=1 TO 8
30 PLAY "O=I;XA$;"
40 NEXT I
```

Comandos associados e referências

Parte 1, Capítulo 28 (Vol. 1): "A Macrolinguagem Musical".

PLAY( ) FUNÇÃO PLAY

Descrição

PLAY( ) informa se o computador está tocando música em um determinado canal.

Devolve – 1 se está tocando música.
0 em caso negativo.

PLAY(0) verifica os três canais, 1, 2 e 3, e se um ou mais de um deles está tocando, devolve –1; se nenhum, então 0.

Sintaxe

<var-num> = PLAY(<tocar canal>)
<tocar canal> = 0 a 3

Pontos a serem lembrados

PLAY( ) é diferente do comando PLAY.

Comandos associados e referências

PLAY

Parte 1, Capítulo 28 (Vol. 1): "A Macrolinguagem Musical".

POINT

Descrição

Encontra a cor do ponto especificado.

Não devolve a cor de um sprite.

Sintaxe

POINT(<coordenada-x> <coordenada-y>)

Exemplo

Este é um programa curto que encontra a cor do modo gráfico de tela.

```
10 OPEN "GRP:" FOR OUTPUT AS #1
20 SCREEN 2
30 COLOR 14,8,9
40 CLS
50 P=POINT(100,100)
60 PRINT #1,"O CODIGO DA COR E:";P
70 GOTO 70
```

Você vê o modo de tela ficar vermelho

Pontos a serem lembrados

Devolve 0 nos modos de texto 0 e 1.

Devolve – 1 quando ás coordenadas estão fora do modo de tela.

Comandos associados e referências

COLOR
SCREEN
PSET
PRESET

Parte 1, Capítulo 23 (Vol. 1): "Desenhando Pontos".

POKE

Descrição

POKE escreve um byte em um determinado endereço de memória.

POKE é utilizado principalmente para inserir programas em linguagem de máquina em um programa BASIC. A área mais útil para uma rotina em programas em linguagem de máquina é a área de trabalho do usuário criada pelo comando CLEAR.

Sintaxe

POKE<endereço>,<ex-inteira>

<endereço> pode estar entre –32786 e 65535. Caso o número seja negativo, o endereço é calculado pela subtração de 65535.

<exp-inteira> deve ser um número de um byte, isto é, entre 0 e 255.

Exemplos

POKE 65535,0

Pontos a serem lembrados

Tenha cuidado com este comando, uma vez que pode destruir o sistema ao mexer em determinadas posições de memória.

Para mexer na RAM de vídeo, utilize VPOKE.

Corrigindo erros

Ocorrerá um erro de extravasamento se <endereço> ou a <exp-inteira> estiver fora da faixa.

Comandos associados e referências

CLEAR
PEEK
VPEEK
VPOKE

Veja Tabela de Variáveis do Sistema.

POS POSIÇÃO DO CURSOR

Descrição

Devolve a posição atual do cursor de texto. Você necessita de uma variável numérica qualquer para executar essa função.

Sintaxe

```
POS(0)
```

Exemplo

```
LET P=POS(0)
```

Pontos a serem lembrados

A posição mais à esquerda é 0.

A posição mais à direita depende do modo de tela.

LARGURA

	usual	máxima
Screen 0	37	40
Screen 1	29	32

Corrigindo erros

Um erro comum é esquecer a variável chamariz (0).

Comandos associados e referências

CRSLIN
WIDTH

Parte 2, Capítulo 40 (Vol. 1): "Gráficos Avançados I".

PRESET REPOSIÇÃO DE PONTO

Descrição

Coloca um determinado ponto à cor de fundo.

Caso uma cor seja especificada, PRESET coloca esta cor no ponto dado. (É o mesmo que PSET.)

Sintaxe

PRESET <especificador de coordenada>
PRESET <especificador de coordenada>, <cor>
especificador de coordenada

1. (<coordenada-x>, <coordenada-y>
2. STEP(<desvio-x>), <desvio-y>

<desvio-x>, <desvio-y>, <coordenada-x>, <coordenada-y> podem ser <var-num>, <exp-num>, ou simplesmente <con-num>

Exemplo

```
10 SCREEN 2
20 CLS
30 PAINT (10,10),15
40 X=RND(1)*255
50 Y=RND(1)*255
60 PRESET (X,Y)
70 GOTO 40
```

LINHA 10 GRÁFICO DE ALTA RESOLUÇÃO MODO 2.
LINHA 20 LIMPA O MODO DE TELA PARA AZUL.
LINHA 30 PINTA O MODO DE TELA EM BRANCO (COR DO TEXTO).
LINHA 40 COORDENADA X.
LINHA 50 COORDENADA X.
LINHA 60 O PONTO REPOSTO À COR DO FUNDO (AZUL).
LINHA 70 VOLTE BUSCANDO MAIS.

Você vê um modo tela azul pintado em branco, depois diversos pontos azuis aparecem ao acaso.

Pontos a serem lembrados.

PRESET <especificador de coordenada>, <cor> é o mesmo que PSET <especificador>, <cor>.

Corrigindo erros

PRESET não fará nada se os pontos estiverem fora da tela. Entretanto, fornecerá um erro de extravasamento se as coordenadas estiverem fora da faixa dos inteiros (–32678 a 32767).

Comandos associados e referências

PSET
POINT
COLOR

Parte 1, Capítulo 23 (Vol. 1): "Desenhando Pontos".

PRINT

Descrição

Este comando é um dos mais utilizados no BASIC. Simplesmente imprime aquilo que você quer na tela.

Depois do comando PRINT, você pode ter uma lista inteira de itens para ser impressa. Tem de ser variáveis string ou numéricas, ou strings entre aspas. A sua posição é determinada pela pontuação utilizada no comando ou pela utilização dos comandos TAB ou LOCATE.

O BASIC divide cada linha do modo de tela em regiões, com 14 caracteres de comprimento. Uma vírgula informa ao computador para imprimir cada item da lista, no início de cada região de 14 caracteres.

Um ponto-e-vírgula, por outro lado, fará com que cada item da lista seja impresso seqüencialmente na linha. Digitar um ou mais <espaços> entre os itens tem exatamente o mesmo efeito que um ponto-e-vírgula.

Caso um comando PRINT termine com um ponto-e-vírgula, o próximo comando PRINT imprimirá na mesma linha com o mesmo espaçamento dado acima, isto é, não haverá retorno de cursor.

Caso a <lista de itens> a ser impressa termine sem pontuação, um retorno de cursor será executado e o próximo comando PRINT imprimirá na linha seguinte.

Caso haja caracteres em excesso para caberem na linha, o computador continuará a impressão na linha seguinte.

Números impressos são sempre seguidos e precedidos por um <espaço>. Números negativos impressos são precedidos por um sinal de menos.

A abreviação de PRINT é "?".

Sintaxe

PRINT<lista de itens>
? <lista de itens>

Exemplo

```
10 PRINT "MENSAGENS DEVEM VIR ENTRE ASPAS"
20 PRINT "O PONTO E VIRGULA INDICA CONTINUACAO ";
30 PRINT "DE ONDE VOCE HAVIA PARADO"
40 A=100
50 B=300
60 PRINT "A=";A,"B=";B
70 PRINT "A+B=";A+B

MENSAGENS DEVEM VIR ENTRE ASPAS
O PONTO E VIRGULA INDICA CONTINUACAO
DE ONDE VOCE HAVIA PARADO
A= 100          B= 300
A+B= 400
```

Esta apresentação assume que a largura do modo de tela seja de 29 caracteres.

Pontos a serem lembrados

Observe que a impressão abreviada "?" torna-se "PRINT" automaticamente ao ser listado.

É mais fácil utilizar PRINT USING ao imprimir uma tabela.

Corrigindo erros

Utilize sempre aspas para as strings.

Comandos associados e referências

PRINT USING
PRINT#
TAB
LOCATE

Parte 1, Capítulo 2 (Vol. 1): "Modo Direto".
Parte 1, Capítulo 9 (Vol. 1): "Mais sobre o Comando PRINT e o Modo de Tela".

PRINT

Descrição

Para se imprimir em vários dispositivos tais como o modo gráfico de telas, você terá de abrir (OPEN) um canal e depois utilizar um comando de impressão especial, o PRINT#. O sinal "#" é seguido pelo número do canal, especificado no comando OPEN.

PRINT# também é utilizado com freqüência no armazenamento de arquivos em um cassete (veja o exemplo 2 para o procedimento completo).

Sintaxe

PRINT# <número do arquivo>

Exemplo

```
10 OPEN "GRP:" FOR OUTPUT AS #1
20 SCREEN 2
30 DRAW "BM45,100"
40 PRINT #1,"ESCRITA NO MODO 2"
50 GOTO 50
```

LINHA 10 ABRE UM ARQUIVO AO MODO GRÁFICO DE TELA COMO SENDO #1.
LINHA 20 SELECIONA O MODO GRÁFICO DE TELA 2.
LINHA 30 MOVE O CURSOR GRÁFICO PARA 30, 100.
LINHA 40 IMPRIME NO MODO GRÁFICO DE TELA.
LINHA 50 TRAVA O MODO GRÁFICO DE TELA.

Pontos a serem lembrados

Um canal tem de ser aberto (OPEN) antes da utilização de PRINT#. INPUT# é complementar a PRINT#.

Corrigindo erros

Ocorre um erro de número de arquivo incorreto quando o número de canal referenciado não é aberto.

Comandos associados e referências

MAXFILES
INPUT#
OPEN
CLOSE

Parte 2, Capítulo 42 (Vol. 1): "Gráficos Avançados III".
Parte 2, Capítulo 47 (Vol. 1): "Como Usar Arquivos".

PRINT USING

Descrição

O comando PRINT USING permite que você imprima strings ou números utilizando um formato especial. Vai ajudá-lo na tabulação da informação de uma maneira apresentável.

Sintaxe

PRINT USING<exp-string>;<lista de itens>

<lista de itens> pode ser números ou strings. Têm de estar separados por ponto-e-vírgula.

<exp-string> é um dos caracteres especiais de formatação. Pode ser uma string ou uma variável string.

Explicação dos caracteres de formatação

! Especifica que o primeiro caractere da string deve ser impresso.
F$="MSX":PRINT USING "!";F$
M

& Insere uma string especificada na posição dada por &.
PRINT USING "ABC&EFG";"MSX"
ABCMSXEFG

O sinal "#" indica que um dígito vai ser impresso, por exemplo:
PRINT USING "#;###";1.3 imprime
1.300 isto é, formata o número a ser apresentado.

Como pode ser visto no exemplo acima, você pode incluir um ponto decimal na string formatada com "#". Caso o número tenha menos dígitos do que o especificado pelos caracteres de formatação, será justificado à direita com espaços precedentes.
PRINT USING "###.##"; 65.87
65.87

+ Um sinal "+" no começo ou fim da string de formatação imprimirá o sinal do número, isto é, "+" ou "–" na posição especificada.

```
PRINT USING "+###.#####";–0.123422
–0.12342
PRINT USING "####.#+";10.71
  10.7 +
```

** O asterisco duplo faz com que os espaços anteriores no campo numérico sejam preenchidos com "*". Representam mais dois espaços de dígitos onde você pode ter números no campo.

```
PRINT USING "**#.##";5.55,–5.55
**5.55*–5.55
```

$$ O duplo cifrão $$ coloca um cifrão antes de um número. $$ especifica mais dois espaços de dígitos no campo, um dos quais será o sinal $.

```
PRINT USING "$$#####.##";10000,99999.99,–100.55
$10000.00$99999.99–$100.55
```

Não podemos utilizar o formato exponencial juntamente com o sinal $$.

**$ Este sinal é um combinação de ** e o sinal $$. Todos os espaços em branco são preenchidos por * e o número é precedido por um sinal $. **$ especifica mais três posições de dígitos, uma das quais é o sinal $.

```
PRINT USING "**$###.##", 34.99
***$34.99
```

Uma vírgula colocada à esquerda do ponto decimal na string de formatação faz com que seja impressa um vírgula a cada três dígitos à esquerda do ponto decimal.

```
PRINT USING X "######,.##";2000000
2,000,000.00
```

Uma vírgula no fim da string de formatação faz com que uma vírgula seja impressa no final do número.

```
PRINT USING "##.##,":12.567
12.57
```

^^^^ Este é o caractere de formatação exponencial e permite o espaço de E+xx. Você pode especificar a posição do ponto decimal. Dígitos significativos são justificados à esquerda, e o expoente é ajustado.

```
PRINT USING "##.## ^^^^ ";200.00
2.00E+02
```

Exemplos

```
PRINT USING "$$#####.##";10000;2000.50;3000.555
$10,000.00  $22,000,50  $33,000.56
```

Existem muitos exemplos dados no Guia do Programador Avançado.

Pontos a serem lembrados

Caso o número a ser impresso não caiba no campo especificado pelos caracteres de formatação, um % é impresso na frente do número. Isto também acontece quando o número arredondado excede o tamanho do campo.

Por exemplo:

```
PRINT USING "##.##";1000
%1000.00
PRINT USING "##.##"99.999
%100.00
```

Corrigindo erros

Caso o número de dígitos especificados excederem 24, resultará um erro de chamada de função ilegal.

Comandos associados e referências

PRINT#USING

Parte 2, Capítulo 36 (Vol. 1): "Imprimindo com o USING".

PRINT # USING

Descrição

PRINT#USING é exatamente o mesmo que PRINT USING, mas este escreve em vários dispositivos. Você precisa executar um comando OPEN para abrir um canal para um dispositivo específico, antes de utilizar este comando.

Sintaxe

PRINT#<número do arquivo>, USING<exp-string>;<lista de expressões>

Exemplo

PRINT#2,USING "##.###"3.453729

Pontos a serem lembrados

Um canal tem de ser aberto antes que você utilize PRINT#USING.

Corrigindo erros

Resultará um erro de número de arquivo errado quando o número do canal referenciado não estiver aberto.

Comandos associados e referências

PRINT USING
MAXFILES
PRINT#
OPEN
CLOSE

Parte 2, Capítulo 36 (Vol. 1): "Imprimindo com o USING".
Parte 2, Capítulo 42 (Vol. 1): "Gráficos Avançados".
Parte 2, Capítulo 47 (Vol. 1): "Como Usar Arquivos".

PSET ESTABELECIMENTO DE PONTO

Descrição

PSET coloca um ponto em coordenadas especificadas na cor especificada ou na cor do texto atual.

Sintaxe

PSET <especificador de coordenada>
PSET <especificador de coordenada>, <cor>
<especificador de coordenadas>
1) (<coordenada-x>, <coordenada-y>)
2) STEP (<desvio-x>, <desvio-y>)

<desvio-x>, <desvio-y>, <coordenada-x>, <coordenada-y> e <cor> podem ser <var-num>, <exp-num>, ou simplesmente <const-num>.

Exemplo

Este programa mostra um céu preto cheio de estrelas multicoloridas.

```
10 COLOR 15,1,1
20 SCREEN 2
30 X=RND(1)*255
40 Y=RND(1)*255
50 Z=INT(RND(1)*16)
60 PSET (X,Y),Z
70 GOTO 30
```

LINHA 10 FUNDO E CONTORNO PRETOS.
LINHA 50 COR RANDÔMICA.

Pontos a serem lembrados

PRESET com a cor especificada é exatamente o mesmo que PSET com a cor especificada.

Corrigindo erros

PSET não vai fazer nada se o ponto referido estiver fora do modo de tela. Entretanto, dará um erro de extravasamento se estiver fora da faixa dos inteiros.

Comandos associados e referências

COLOR
PRESET

Parte 1, Capítulo 23 (Vol. 1): "Desenhando Pontos".

PUT SPRITE

Descrição

Coloca um sprite no modo de tela. Você pode colocar um sprite por plano de sprite. Existem 32 planos de sprite, portanto o número máximo de sprites apresentados em qualquer momento é 32.

O especificador de coordenadas informa ao computador onde colocar o sprite. Você pode esconder um sprite além do limite do modo de tela, ou se deslocar atrás ou na frente de outro sprite.

Pode utilizar todas as 16 cores. Entretanto, pode utilzar apenas um cor por sprite.

Veja a seção de sprites no Guia de Programação Avançada de como utilizar os sprites.

Sintaxe

PUT SPRITE <número do plano sprite> [,<especificador de coord>]
[,<cor>][,<número do padrão>].

Todos os argumentos podem ser <const-num>, <var-num>, ou <exp-num> mas nos seus limites permitidos.

<número do plano sprite> = 0 a 31
<especificador de coordenadas>
1) (<coordenada-x>, <coordenada-y>)
2) STEP (<desvio-x>, <desvio-y>)

Faixa das coordenadas: entre –32768 e 32767.
Faixa das coordenadas do modo de tela sprite: X = – 32 a 255
Y = – 32 a 191

Caso as coordenadas caiam fora da faixa do modo de tela sprite, ocorrerá um giro automático. (Veja o Guia de Programação Avançada.)
<número do padrão> = 0 a 256, se o tamanho do sprite é 8 por 8 pontos.
= 0 a 64, se o tamanho do sprite é 16 por 16 pontos.

Exemplo

```
10 SCREEN 2
20 SPRITE$(0)=STRING$(8,CHR$(255))
30 X=100 : Y=120
40 C=7
50 PUT SPRITE 0,(X,Y),C,0
60 GOTO 60
```

LINHA 10 SELECIONA O MODO DE TELA 2.
LINHA 20 DEFINE UM SPRITE QUADRADO (0).
LINHA 30 COORDENADAS X E Y.
LINHA 40 A COR É AZUL CELESTE (7).
LINHA 50 COLOCA A PADRÃO DE SPRITE 0 NO PLANO DE SPRITE 0 NAS COORDENADAS X E Y EM AZUL CELESTE.

Pontos a serem lembrados

Quando 208 (&HDO) ou 209 (&HD1) é fornecido à coordenada Y, tem o efeito de esconder um ou mais sprites temporariamente. (Veja Guia de Programação Avançada: seção sprite, para maiores detalhes em sua aplicação.)

Você não pode utilizar sprites no modo de texto 0.

O tamanho do sprite é determinado pelo comando SCREEN.

Caso o <especificador de coordenadas> seja omitido, o computador colocará o sprite no último lugar referenciado.

Corrigindo erros

Resultará um erro de extravasamento se o especificador de coordenadas estiver fora da faixa dos inteiros.

Comandos associados e referências

ON SPRITE GOSUB
SPRITE$
SPRITE ON/OFF/STOP
SCREEN
COLOR

Parte 2, Capítulo 43 (Vol. 1): "Gráficos Avançados IV".

READ

Descrição

Lê dados do comando DATA e os coloca nas variáveis do comando READ.

O comando READ é sempre utilizado juntamente com o comando DATA. Os dados são armazenados em DATA, que são comandos não executáveis, cuja única finalidade é armazenar informações para o programa. Para acessar a informação de um DATA, utilize o comando READ.

Você pode ler uma lista de variáveis em apenas um comando READ. Podem ser elementos de matrizes, variáveis string ou numéricas, mas os tipos de dados têm de coincidir ou haverá um erro de sintaxe.

O READ lê dos comandos DATA a partir da linha mais inferior no programa avançado para as superiores. Se desejar ler os dados de uma determinada linha, terá de utilizar READ juntamente com o comando RESTORE, que aponta ao comando DATA ao qual READ deverá se referir.

Sintaxe

READ <lista de variáveis>

Exemplo

```
10 READ A$,B
20 PRINT A$;B
30 READ C$,D
40 PRINT C$;D
50 DATA "ASTRONOMIA",500,"FISICA",600

ASTRONOMIA 500
FISICA 600
```

Pontos a serem lembrados

Conforme mostrado no programa acima, o computador lembra o último item de dado lido. O próximo comando READ lê o primeiro item de DATA após o último item de dado lido.

Corrigindo erros

Resulta erro de sintaxe quando a variável READ não combina com o tipo de DATA.

Ocorre um erro do tipo Out of DATA (falta de dados) caso tente ler, quando você já leu todos os dados. Solução: utilizar RESTORE.

Comandos associados e referências

DATA
RESTORE

Parte 1, Capítulo 12 (Vol. 1): "Lendo Dados em Matrizes".

REM COMENTÁRIO

Descrição

Permite ao programador colocar comentários no programa. Um comando REM não é executado, isto é, o computador pula as linhas com REM.

Os comandos REM são utilizados para documentar os programas. Quando estiver escrevendo um programa é aconselhável colocar comandos REM para anotar o que cada seção faz. Caso contrário, é fácil esquecer o que está acontecendo no programa. Fazendo uma boa documentação de seu programa com vários REM, você poderá se lembrar como é que o programa funciona. Os comandos REM tornam seu programa muito mais compreensível aos outros.

REM pode ser abreviado por apóstrofo.

Sintaxe

REM comentário ...
' comentário ...

Exemplo

```
10 REM Meu primeiro programa
20 REM Titulo  : OVNI
30 REM Inicio  : 18/06/87
40 REM Versao  : 1
50 REM
```

Pontos a serem lembrados

REM pode ser colocado no final de uma linha, (:REM..), porém, não faça isto nos comandos DATA, pois o computador vai considerar como sendo mais dados.

Os comandos REM facilitam a compreensão do programa, mas utilizam também muita memória. É conveniente utilizá-los até que fique sem memória, depois apague conforme a necessidade.

Nunca utilize GOTO ou GOSUB para atingir um REM. Se depois tiver de apagar os comandos REM referidos em GOTO ou GOSUB, haverá um erro de linha indefinida.

Comandos associados e referências

Parte 1, Capítulo 3 (Vol. 1): "Escrevendo um Programa".

RENUM RENUMERAR

Descrição

Este comando renumera as linhas do programa. Com freqüência verá que seus programas estão em desordem porque os números das linhas não estão com um incremento fixo. Utilize RENUM pois é o melhor modo para se melhorar a listagem de um programa e facilitar a inserção de novas linhas.

RENUM faz apenas a renumeração a partir de 10 com incrementos de 10; este é o esquema de numeração de linhas mais utilizado, mas você pode renumerar a partir de uma determinada linha com qualquer incremento.

RENUM alterará automaticamente os números das linhas associados a GOTO, GOSUB, RESTORE, THEN, ELSE, ON GOTO, ON GOSUB.

Sintaxe

RENUM

Renumera desde 10 com incrementos de 10.

RENUM<linha 1>

Renumera do novo número de linha 1 em incrementos de 10.

RENUM<linha1>,<linha2>

Renumera do antigo número de linha 2, iniciando com um novo número de linha 1, em incrementos de 10.

RENUM<linha 1>, <linha 2>, <incremento>

Renumera iniciando com o antigo número de linha 2, iniciando com um novo número de linha 1, com o incremento dado.

RENUM, <linha2>, <incremento>

Renumera a partir do antigo número de linha 2, iniciando com um novo número de linha 10, com o incremento dado.

RENUM<linha 2>

Renumera a partir do antigo número de linha 2, iniciando com um novo número de linha 10 em incrementos de 10.

RENUM,,<incremento>

Renumera a partir do antigo número de linha 10 iniciando com um novo número de linha 10, no incremento dado.

RENUM<linha 1>,,<incremento>

Renumera a partir do antigo número de linha 10, iniciando com um novo número de linha 1, no incremento dado.

Exemplos

RENUM,,100

Renumera a partir do 10 com incrementos de 100.

RENUM 1000,200,20

Renumera a partir do antigo número de linha 200, iniciando com um novo número de linha 1000, em incrementos de 20.

```
10 REM
20 REM
30 REM
40 REM
50 REM
```

```
RENUM ,10,20
OK
LIST
10 REM
30 REM
50 REM
70 REM
90 REM
```

Corrigindo erros

Caso um número de linha não-existente aparecer depois de GOTO ou GOSUB ou qualquer outro elemento associado em ter um número e linha depois dele, isto provocará um erro "Undefined line"

NN em MMMM2.

Por exemplo!

```
10 GOTO 97
```

RENUM
Linha indefinida 97 em 10.

Uma numeração sem cuidados como RENUM 10,40 quando as linhas antigas têm números 10,20,30 não funciona. O resultado será um erro de chamada ilegal de função.

Incrementos negativos também não são permitidos.

Você não pode ter números de linha maiores que 65535; assim, se durante a renumeração os números das linhas ultrapassarem este valor, resultará um erro de chamada ilegal de função.

Comandos associados e referências

Parte 1, Capítulo 7 (Vol. 1): "Comandos Úteis e Dicas para Escrever Programas".

Parte 2, Capítulo 29 (Vol. 1): "Edição Avançada de Programas".

RESTORE

Descrição

Restaura comandos DATA. Quando o computador lê comandos DATA, isto é, quando usamos o comando READ, ele inicia com o DATA com o número de linha menor, e continua pelas linhas crescentes até ficar sem DATA. Para fazer com que o computador releia dados de uma determinada linha, você pode apontar para o número da linha que contém o DATA desejado, utilizando o RESTORE.

Depois do RESTORE, o próximo comando READ lerá os primeiros dados da linha apontada pelo RESTORE. Caso não haja nenhum número de linha especificado, então o computador lerá do primeiro DATA que encontrar no programa.

Sintaxe

RESTORE
RESTORE <linha>

Exemplo

```
10 READ A$
20 PRINT A$
30 RESTORE
40 READ B$
50 PRINT B$
60 DATA "IDADE DE OURO"

IDADE DE OURO
IDADE DE OURO
```

Pontos a serem lembrados

Utilize RESTORE para impedir um erro de falta de dados.

Corrigindo erros

Caso RESTORE aponte para um número de linha não existente, resultará um erro de número de linha indefinido.

Comandos associados e referências

READ
DATA

Parte 1, Capítulo 12 (Vol. 1): "Lendo Dados em Matrizes".

RESUME

Descrição

RESUME significa continuar a operação BASIC depois de ter sido encontrado um comando de tratamento de erros do tipo: ON ERROR GOTO. Depois de termos cuidado do erro, RESUME informa ao computador para que continue a execução a partir do ponto que você desejar.

Caso não haja um RESUME na rotina de tratamento de erros, ocorrerá um erro "NO RESUME" (Nenhum RESUME), a não ser que você pare o programa utilizando STOP ou END na rotina de tratamento de erro. Existem três formas de RESUME.

Sintaxe

RESUME ou RESUME0

Reassume a exceção a partir do comando que provocou o erro.

RESUME NEXT

Reassume a operação a partir do comando seguindo aquele que provocou o erro.

RESUME<linha>

Reassume a execução a partir do número de linha especificado.

Exemplo

Nesta rotina, todos os comandos colocados que tenham KILL neles serão tratados como um erro novo (n º 255) e serão tratados na rotina de tratamento de erros utilizando a função ERR. RESUME 20 informa ao computador para que continue a operação na linha 20 depois que a mensagem de alerta é mostrada.

```
10 ON ERROR GOTO 1000
20 INPUT "MEU AMO,O QUE DESEJAS";A$
30 IF INSTR(A$,"MATAR") THEN ERROR 255
40 PRINT "OK!" :END
1000 IF ERR=255 THEN PRINT "O ASSASSINATO E ILEGAL NES
TA AVENTURA":RESUME 20
1010 END
```

```
MEU AMO,O QUE DESEJAS? RIR E MATAR
O ASSASSINATO E ILEGAL NESTA AVENTURA

MEU AMO,O QUE DESEJAS? CHORAR E COMER

OK!
```

Pontos a serem lembrados

Você não pode utilizar RESUME para voltar ao BASIC a partir do modo direto.

Corrigindo erros

Um comando RESUME que não esteja em uma sub-rotina de tratamento de erros provoca um erro de RESUME sem erro.

Teremos um erro de número de linha indefinido se RESUME se referir a uma linha inexistente.

Comandos associados e referências

ON ERROR GOTO
ERL
ERR
ERROR

Parte 2, Capítulo 38 (Vol. 1): "Tratamento de Erros".

RETURN

Descrição

RETURN de uma sub-rotina chamada por um GOSUB.

RETURN faz com que o computador pule ao comando situado imediatamente após o GOSUB por último executado.

Sintaxe

RETURN

Exemplo

```
10 PRINT 1000
20 GOSUB 50
30 PRINT 2000
40 STOP
50 PRINT "SUB-ROTINA"
60 RETURN

 1000
SUB-ROTINA
 2000
Break in 40
```

Pontos a serem lembrados

É muito fácil entrar em uma sub-rotina por engano. É portanto conveniente tornar a sub-rotina distinguível e ter um END ou STOP no fim do programa principal para impedir uma entrada acidental.

Caso a lógica de seu programa assim o desejar, é perfeitamente possível ter mais que um RETURN em qualquer sub-rotina.

RETURN também é utilizado em diversas sub-rotinas de tratamento de erros.

Corrigindo erros

Um erro de RETURN sem GOSUB acontece quando o computador encontra um RETURN inesperadamente, sem um GOSUB anterior.

Comandos associados e referências

GOSUB
ON ... GOSUB
ON ERROR GOSUB
ON INTERVAL GOSUB
ON KEY GOSUB
ON SPRITE GOSUB
ON STOP GOSUB
ON STRIG GOSUB

Parte 1, Capítulo 17 (Vol. 1): "Estruturando o seu Programa".

RIGHTS

Descrição

Esta é uma das funções de manipulação de string. RIGHT$(A$,n) devolve os n caracteres mais à direita de uma string A$, por exemplo. Caso n seja igual ao comprimento de A$, então obviamente conseguirá toda string A$ de volta. Caso n seja zero, então será devolvido uma string nula.

Sintaxe

RIGHT$ (<exp-string>, <numérico-inteiro>)

Exemplo

```
10 A$="TORRE DE CONTROLE A MAMAE GANSO"
20 PRINT RIGHT$(A$,11)
30 B$="**MISSAO CANCELADA**,POTENCIA"
40 C=8
50 D$=RIGHT$(B$,C)
60 PRINT D$

MAMAE GANSO
POTENCIA
```

Corrigindo erros

O argumento de RIGHT$ é uma string seguida por um número. Caso estes estejam na ordem reversa, teremos um erro de Desencontro de Tipo.

Comandos associados e referências

LEFT$
MID$
INSTR

Parte 1, Capítulo 14 (Vol. 1): "Manipulando Strings".

RND NÚMERO RANDÔMICO

Descrição

RND gera um número randômico entre 0 e 1. A mesma série de números randômicos será gerada cada vez que o programa for processado. Para gerar números realmente randômicos que não segue esta série, utilize RND(–TIME).

Sintaxe

RND(<numérico>)

Caso <numérico> seja negativo, faz o gerador de números randômicos fornecer uma série diferente de números randômicos, de acordo com o número dado.

Se <numérico> é 0, então repete o último número dado.

Se <numérico> é positivo, o próximo número randômico da string será gerado.

Exemplo

```
10 FOR I=1 TO 5
20 PRINT RND(1)
30 NEXT I
```

```
RUN
 .59521943994623
 .10658628050158
 .76597651772823
 .57756392935958
 .73474759503023
Ok
```

Nota: Este programa gera a mesma série de números randômicos cada vez que é processado.

Para gerar um inteiro randomicamente, entre 0 e 9, utilize esta função.

```
X=INT(RND(1)*10)
```

Pontos a serem lembrados

RND gera um número de 14 dígitos, números de precisão dupla, entre 0 e 0.99999999999999.

RND(–TIME) fornece um número realmente randômico, ao passo que RND(1) fornece a mesma série de números randômicos.

Comandos associados e referências

Parte 1, Capítulo 15 (Vol. 1): "Funções".

RUN

Descrição

Executa um programa.

Sintaxe

RUN

RUN<linha>... Executa um programa a partir da linha dada.

Exemplos

```
10 PRINT 10000
20 A=300
30 B=200
40 PRINT A+B
```

```
RUN
 10000
 500
Ok
```

```
RUN 20
500
Ok
```

Pontos a serem lembrados

Os comandos END ou STOP dentro do programa interrompem a execução.

<CTRL><STOP> termina a execução a partir do teclado.

Ao ligar o computador, a tecla de função F5 é programada como RUN <RETURN>.

Corrigindo erros

Caso não haja nenhum programa BASIC na memória, RUN resultará em "OK" apresentado no modo de tela.

Comandos associados e referências

END
STOP

Parte 1, Capítulo 3 (Vol. 1): "Escrevendo um Programa".
Parte 1, Capítulo 7 (Vol. 1): "Comandos Úteis e Dicas para Escrever Programas".

SAVE

Descrição

Grava um programa em BASIC no dispositivo especificado, no formato ASCII.

SAVE é utilizado quando se deseja juntar dois programas. Veja MERGE e a Parte de Operação Cassete no Guia de Programação Avançada.

Sintaxe

SAVE "<nome do programa>"

SAVE "<descritor de dispositivo> [<nome arquivo>]"

<descritor de dispositivo> = CAS: para cassete.

Exemplo

Vamos dizer que o programa a seguir está na memória do computador:

```
10 PRINT "OLA"
20 PRINT "BEM-VINDO"
30 PRINT "AO"
40 FOR I=1 TO 10
50 PRINT "MSX"
60 NEXT I
```

Para gravá-lo no formato ASCII com o nome de programa "PROG" utilize: SAVE "PROG" ou SAVE "CAS:PROG"
Para carregá-lo novamente para ser utilizado:

LOAD "PROG" ou LOAD "CAS:PROG"

Pontos a serem lembrados

SAVE é diferente de CSAVE, porque SAVE grava o programa BASIC em arquivo ASCII, ao passo que CSAVE grava-o em forma criptografada, o que é bem diferente.

Comandos associados e referências

MERGE

LOAD

Parte 2, Capítulo 39 (Vol. 1): "Gravando no e Carregando do Cassete".

SCREEN

Descrição

SCREEN é um comando que possui um grande número de funções. É utilizado para estabelecer os diversos modos de operação da tela, velocidade do cassete, clik do teclado, tamanho dos sprites e impressora. É utilizado principalmente para estabelecer os modos de tela.

Utilizando o comando SCREEN, você pode especificar o tamanho do sprite, mas pode utilizar apenas um tamanho por vez.

Outras opções estabelecidas por SCREEN são click das teclas, velocidade em baud do cassete e seleção da impressora.

Sintaxe

SCREEN [<modo>][,<tamanho do sprite>][,<click das teclas>]
[, <velocidade em baud do cassete>][,<opção de impressora>]

	EXEMPLOS
<modo> = 0,1,2,3	
0 = modo texto 40 x 24	SCREEN 0
1 = modo texto 32 x 24	SCREEN 1
2 = modo de alta resolução	SCREEN 2
3 = modo multicolorido	SCREEN 3
<tamanho do sprite> = 0,1,2,3	
0 = sem ampliação 8 x 8	SCREEN,0
1 = ampliada 8 x 8	SCREEN,1
2 = sem ampliação 16 x 16	SCREEN,2
3 = ampliado 16 x 16	SCREEN,3

Nota: Quando o tamanho do sprite é especificado, o conteúdo do SPRITE$ ficará claro.

<click das teclas> = 0 = desativa o click das teclas		
	não-zero = ativa o click das teclas	SCREEN,,0
		SCREEN,,1
<velocidade em baud do cassete>=0,1		
0 = 1200 baud		SCREEN,,,0
1 = 2400 baud		SCREEN,,,1

Nota: A velocidade em baud também pode ser alterada utilizando CSAVE.

<opção de impressora> = 0, não zero		
0	= impressão com impressora MSX que tem capacidade gráfica MSX	SCREEN,,,0
não zero	= impressora padrão não MSX, todos os caracteres gráficos são alterados para espaços	SCREEN,,,,1

Exemplo

Este exemplo mostra como o modo gráfico 2 (alta resolução) e 3 (baixa resolução) diferem. O programa executa a mesma tarefa em cada modo, mostrando-lhe as vantagens e desvantagens de cada um.

```
10  SCREEN 2
20  GOSUB 90
30  PAINT (105,120)
40  FOR I=1 TO 1000 : NEXT
50  SCREEN 3
60  GOSUB 90
70  PAINT (105,120),8,15
80  GOTO 80
90  PSET (100,100)
100 DRAW "R50D50L50U50F50"
110 RETURN
```

```
LINHA 10    MODO GRÁFICO DE ALTA RESOLUÇÃO.
LINHA 20    VAI ATÉ SUB-ROTINA.
LINHA 30    PINTA COM COR ATUAL DO TEXTO.
LINHA 40    ATRASO.
LINHA 50    MODO GRÁFICO POLICROMÁTICO.
LINHA 60    VAI ATÉ SUB-ROTINA.
LINHA 70    PINTA EM VERMELHO A ÁREA ENVOLVIDA PELA LINHA.
LINHA 90    SUB-ROTINA: POSICIONA O CURSOR GRÁFICO.
LINHA 100   DESENHA UM QUADRADO COM UMA LINHA DIAGONAL.
LINHA 110   VOLTA.
```

Pontos a serem lembrados

As características de cada modo de tela são dadas na Parte Gráfico de Computação Avançada.

Corrigindo erros

Utilizando INPUT nos modos de tela 2 e 3, força você a voltar ao último modo de texto utilizado.

A utilização de qualquer comando gráfico em um modo texto, exceto PUT SPRITE no modo 1, resulta em uma chamada ilegal de uma função.

Comandos associados e referências

Parte 1, Capítulo 9 (Vol. 1): "Mais sobre o comando PRINT e o Modo de Tela".

Parte 1, Capítulo 21 (Vol. 1): "Modos de Tela".

Parte 2, Capítulo 39 (Vol. 1): "Gravando no e Carregando do Cassete".

Parte 2, Capítulo 40 (Vol. 1): "Gráficos Avançados I".

Parte 2, Capítulo 43 (Vol. 1): "Gráficos Avançados IV".

Parte 2, Capítulo 51 (Vol. 1): "Periféricos".

SGN SINAL

Descrição

SGN devolve o sinal de um determinado número.

–1 se o argumento é negativo.
0 se o argumento é zero.
1 se o argumento é positivo.

Sintaxe

SGN(<numérico>)

Exemplo

```
PRINT SGN(–1000)
–1
```

Comandos associados e referências

ABS

Parte 1, Capítulo 15 (Vol. 1): "Funções".

SIN SENO

Descrição

SIN devolve o seno do argumento dado em radianos.

Sintaxe

SIN(<numérico>)

Exemplo

```
PRINT SIN(1)
.84147098480792
```

Pontos a serem lembrados

SIN calcula em precisão dupla, e fornece até 14 dígitos significativos.

Comandos associados e referências

COS
TAN
ATN

Parte 1, Capítulo 19 (Vol. 1): "Funções Matemáticas".

SOUND

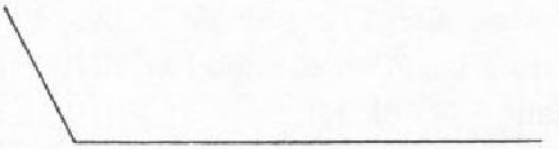
Descrição

SOUND escreve diretamente nos registradores do circuito integrado, responsáveis pela emissão de sons do MSX. O PSG tem 14 registros de som. Para uma explicação passo a passo do comando SOUND, veja a seção de efeitos avançados de som (Parte 2, Capítulo 18 (Vol. 1)).

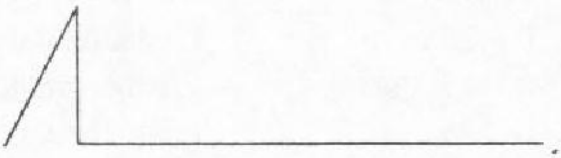
REGISTRO	FAIXA	DESCRIÇÃO
0	0 – 255	Freqüência do canal A.
1	0 – 15	Freqüência do canal A (mais significativo).
2	0 – 255	Freqüência do canal B.
3	0 – 15	Freqüência do canal B (mais significativo).
4	0 – 255	Freqüência do canal C.
5	0 – 15	Freqüência do canal C (mais significativo).
6	0 – 31	Freqüência do gerador de ruído.
7	0 – 63	Mixer (cada bit tem um controle único).
	bit 0	Tom no canal A 0=ON 1=OFF
	bit 1	Tom no canal B 0=ON 1=OFF
	bit 2	Tom no canal C 0=ON 1=OFF
	bit 3	Ruído no canal A 0=ON 1=OFF
	bit 4	Ruído no canal B 0=ON 1=OFF
	bit 5	Ruído no canal C 0=ON 1=OFF
8	0 – 16	Controle de volume para o canal A (utiliza envoltória quando = 16)
9	0 – 16	Controle de volume para o canal B (utiliza envoltória quando = 16).
10	0 – 16	Controle de volume para o canal C (utiliza envoltória quando = 16).
11	0 – 255	Período da envoltória (baixo).
12	0 – 255	Período da envoltória (alto). Período da envoltória = (0–65535)=R12*256+R11

Registro	Envoltória Nº	
13	0 – 15	Padrão da envoltória

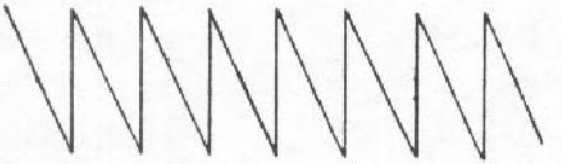
0,1,2,3,9



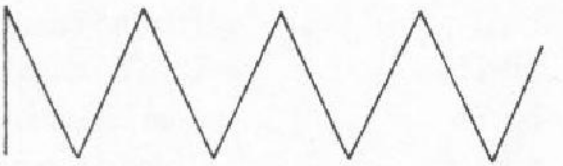
4,5,6,7,15



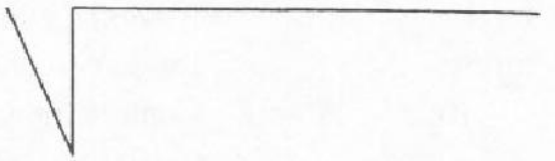
8



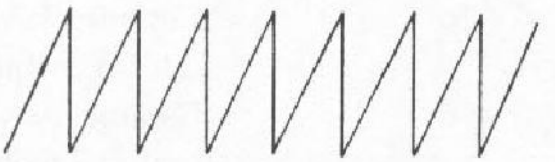
10



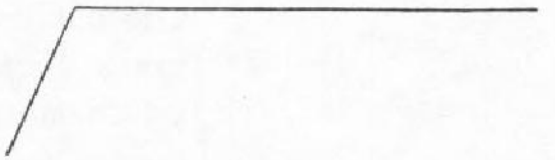
11



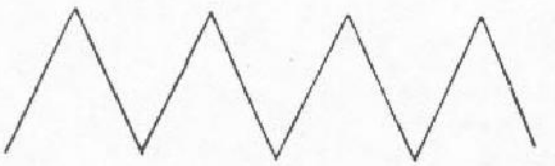
12



13



14



Sintaxe

Sound <registro>, <dados>

Exemplo

Efeitos sonoros especiais:

```
10 SOUND 7,62
20 SOUND 1,0
30 SOUND 0,254
40 SOUND 8,16
50 SOUND 13,9
60 SOUND 12,60
70 SOUND 11,0
```

Corrigindo erros

Caso você não escute nenhum som, a sintaxe está errada, ou o controle de volume de seu televisor (ou computador) está desligado.

Comandos associados e referências

PLAY

Parte 2, Capítulo 46 (Vol. 1): "Efeitos Avançados de Som usando o PSG".

SPACE$ STRING DE ESPAÇO

Descrição

SPACE$ fornece uma string de <espaços> de um determinado tamanho. O argumento pode ser real mas tem de estar dentro da faixa de 0 a 255. As partes fracionárias de um número real são descartadas.

Sintaxe

SPACE$(<numérico>)

faixa de 0 a 255.

Exemplo

```
10 A$="ORDENADOR"+SPACE$(5)+"MSX"
20 PRINT A$

ORDENADOR      MSX
```

Pontos a serem lembrados

Existe uma função semelhante, SPC, que também imprime espaços em branco, mas esta só pode ser utilizada com os comandos PRINT e LPRINT.

Corrigindo erros

Ocorre um erro de Desencontro de Tipo ao ser equiparado a uma variável numérica.

Comandos associados e referências

SPC

Parte 1, Capítulo 9 (Vol. 1): "Mais sobre o comando PRINT e o Modo de Tela".

SPC ESPAÇO

Descrição

SPC imprime espaços na tela. Pode ser utilizado apenas nos comandos PRINT e LPRINT. É utilizado principalmente para economizar memória quando você tem de incorporar uma grande quantidade de espaço em branco em um comando PRINT.

Sintaxe

SPC(<espaço>)

faixa de 0 a 255.

Exemplo

```
10 A$="ORDENADOR":B$="MSX"
20 PRINT A$;SPC(10);B$

ORDENADOR          MSX
```

Pontos a serem lembrados

SPC é diferente de SPACE$ porque não pode ser tratado como uma variável string.

Corrigindo erros

Resultará uma chamada a uma função ilegal se o argumento ultrapassar 255.

Comandos associados e referências

PRINT
LPRINT
SPACE$

Parte 1, Capítulo 9 (Vol. 1): "Mais sobre o Comando PRINT e o Modo de Tela".

SPRITE ON/OFF/STOP

Descrição

Este comando é utilizado para a manipulação de eventos gráficos de sprite. Ele ativa (ON) ou desativa (OFF ou STOP) a condição para detectar colisão de sprites em um programa BASIC.

Um SPRITE ON tem de ser executado para ativar a detecção da colisão de sprite e fazer o computador pular para a sub-rotina especificada no comando ON SPRITE GOSUB. Depois de SPRITE ON, o computador verifica a existência de qualquer colisão de sprite toda vez que o BASIC inicia um novo comando. Quando uma colisão é detectada, o computador vai imediatamente à sub-rotina.

Caso um SPRITE STOP seja executado, o computador interromperá a detecção de colisão de sprites, mas ela se lembrará caso tenha havido colisão enquanto SPRITE STOP estava agindo e, assim que SPRITE ON for executado, ele fará com que o computador vá até a sub-rotina sprite.

SPRITE OFF interrompe completamente a procura pela colisão de sprite. Nenhuma colisão será lembrada.

Sintaxe

SPRITE ON
SPRITE OFF
SPRITE STOP

Exemplo

Neste exemplo, você verá dois sprites quadrados, um amarelo e outro branco, aproximando-se de dados opostos da tela. Quando colidem, ON SPRITE GOSUB entra em ação fazendo com que o BASIC vá até a sub-rotina de interrupção. O SPRITE OFF na rotina de interrupção de sprite impede qualquer outra detecção de colisão de sprite. (Tente esta rotina sem SPRITE OFF e veja o que acontece.)

```
10 ON SPRITE GOSUB 110
20 SCREEN 2,0
30 SPRITE$(0)=STRING$(8,CHR$(255))
40 SPRITE$(1)=STRING$(8,CHR$(255))
50 SPRITE ON
60 FOR I=10 TO 240
70 PUT SPRITE 0,(I,100),11,0
80 PUT SPRITE 1,(250-I,100),15,1
90 NEXT I
100 END
105 'Sub-rotina de colisao de sprites
110 SPRITE OFF
120 BEEP
130 RETURN
```

```
LINHA 10    COLOCA A SUB-ROTINA DE CILADA EM 110.
LINHA 20    ESTABELECE O MODO GRÁFICO DE TELA.
LINHA 30    SPRITE 0 É UM QUADRADO.
LINHA 40    ASSIM TAMBÉM O É O SPRITE 1.
LINHA 50    DETECÇÃO DE COLISÃO DE SPRITE LIGADO.
LINHA 60    LOOP.
LINHA 70    UM SPRITE (AMARELO) SE DESLOCA DA ESQUERDA.
LINHA 80    UM SPRITE (BRANCO) SE DESLOCA DA DIREITA.
LINHA 90    PRÓXIMO LOOP.
LINHA 100   TÉRMINO.
LINHA 110   MATA A CILADA (UMA VEZ É SUFICIENTE).
LINHA 120   INDICADOR DE RUÍDO.
LINHA 130   VOLTE AO LOOP.
```

Pontos a serem lembrados

A interrupção do SPRITE será desativada quando o programa não estiver sendo processado e também durante as rotinas para detectar erros.

Corrigindo erros

Você tem de executar um comando ON SPRITE GOSUB antes de utilizar o comando SPRITE ON/OFF/STOP. Caso contrário, o computador não detectará nenhuma colisão de sprites.

Comandos associados e referências

ON SPRITE GOSUB
SPRITE$
PUT SPRITE

Parte 2, Capítulo 43 (Vol. 1): "Gráficos Avançados IV".

SPRITE$

Descrição

Você pode definir sprites utilizando o comando SPRITE$. Você pode ter até 256 padrões de sprite quando estiver utilizando tamanho de sprite 0 ou 1 (não ampliado), ou 64 deles nos tamanhos de sprite 2 ou 3 (ampliado).

O tamanho de um sprite é fixo em 32 bytes, mas para um pequeno sprite é necessário definir apenas os primeiros 8 caracteres. O restante é automaticamente definido, como CHR$(0).

Para definir um sprite de 16 por 16 pontos, você tem de definir todos os 32 bytes do padrão do sprite.

Os detalhes de como utilizar sprites são fornecidos no Guia de Programação Avançada: Sprites.

Sintaxe

SPRITE$(<inteiro>)=<exp-string>

Exemplos

```
10 SCREEN 2
20 SPRITE$(0)=CHR$(16)+CHR$(48)+CHR$(112)+CHR$(255)+CH
R$(255)+CHR$(112)+CHR$(48)+CHR$(16)
30 PUT SPRITE 0,(100,100),9,0
40 GOTO 40
```

LINHA 10 MODO 2 DE ALTA RESOLUÇÃO.
LINHA 20 DEFINE SPRITE 0.
LINHA 70 COLOCA SPRITE 0 EM X = 100, Y = 100; COR BRANCA NO PLANO DE SPRITE 0

Resultado: Você vê uma seta ao meio da tela.

Pontos a serem lembrados

Você pode utilizar SPRITES apenas nos modos 1, 2 e 3 da tela.

O tamanho do sprite é definido utilizando o comando SCREEN. O comando SCREEN também elimina todos os sprites anteriormente definidos.

Você não consegue misturar dois tamanhos de sprite.

Comandos associados e referências

PUS SPRITE
SCREEN
SPRITE ON/OFF/STOP

Parte 2, Capítulo 43 (Vol. 1): "Gráficos Avançados IV".

SQR RAIZ QUADRADA

Descrição

Esta é a função raiz quadrada.

O argumento tem de ser positivo. SQR do MSX não trata com números complexos.

Sintaxe

SQR (<numérico>)

Exemplo

```
10 PRINT SQR(9)
20 PRINT SQR(ATN(1)*4)


RUN
 3
 1.7724538509055
```

Pontos a serem lembrados

SQR é calculado em precisão dupla.

Comandos associados e referências

Parte 1, Capítulo 19 (Vol. 1): "Funções Matemáticas".

STEP

Descrição

STEP é parte do comando FOR/TO/NEXT. STEP indica quando é que a variável utilizada no FOR/TO deve ser incrementada em cada loop.

Você pode avançar em passos menores que 1 ou ter mesmo um passo negativo se quiser. STEP 1 não será necessário, como todos loops FOR/TO/NEXT sem o STEP especificado são incrementados em 1.

Sintaxe

FOR<var-num>=<numérico> TO<numérico> STEP <numérico>

Exemplo

```
10 FOR I=1 TO 3 STEP .5
20 PRINT I;
30 NEXT I
40 PRINT
50 FOR J=3 TO 1 STEP -1
60 PRINT J;
70 NEXT J
```

```
RUN
 1   1.5   2   2.5   3
 3   2   1
```

Corrigindo erros

```
FOR I = 1 TO 10 STEP – 1
```

Este é um erro comum. Este loop se repete apenas uma vez.

Comandos associados e referências

FOR
TO
NEXT

Parte 1, Capítulo 5 (Vol. 1): "O Uso de Loops".

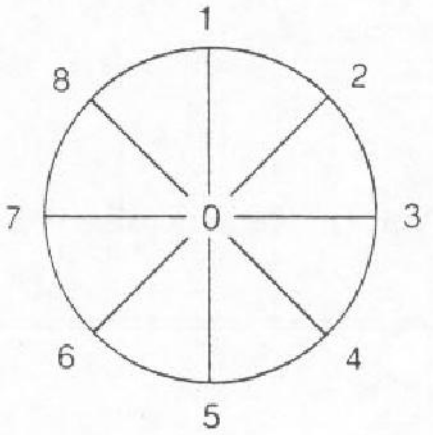
STICK JOYSTICK

Descrição

STICK devolve a direção de um *joystick* ou as teclas de cursor quando são utilizadas como *joystick*.
STICK(<n>) ... <n> pode ser 0,1 ou 2.

n = 0 teclas de cursor.
n = 1 joystick 1.
n = 2 joystick 2.

Quando o *joystick* estiver na posição neutra, um 0 será devolvido. Caso contrário, será como segue:



Você tem de pressionar duas teclas de cursor ao mesmo tempo para conseguir a direção diagonal, isto é, pressione ↑ e → para a direção 2.

Sintaxe

STICK(<n>)

Exemplos

Este exemplo apresenta os valores de direção quando se utiliza as teclas de cursor ou o *joystick*.

```
10 A=STICK(0)
20 LOCATE 15,10 : PRINT A
30 GOTO 10
```

Pontos a serem lembrados

STICK devolve sempre um inteiro entre 0 e 8.

Para encontrar o estado do gatilho do *joystick*, utilize a função STRIG.

Comandos associados e referências

STRIG

Parte 2, Capítulo 51 (Vol. 1): "Periféricos".

STOP

Descrição

STOP encerra a execução do programa BASIC e volta ao nível de comando.

A mensagem "Break in <linha>" é apresentada depois da terminação STOP.

Sintaxe

STOP

Exemplo

```
10 PRINT 1909
20 STOP


RUN
 1909
Break in 20
Ok
```

Pontos a serem lembrados

O comando STOP não fecha os arquivos no fim do programa, ao passo que END o faz.

A execução pode continuar a partir da próxima linha utilizando CONT.

Você pode utilizar quantos STOP desejar.

STOP é diferente da declaração STOP/ON/OFF/STOP.

Comandos associados e referências

END

Parte 1, Capítulo 7 (Vol. 1): "Comandos Úteis e Dicas para Escrever Programas".

STOP ON/OFF/STOP

Descrição

Este comando é bem diferente do comando STOP, que encerra a execução de um programa. STOP ON/OFF/OFF/STOP ativa/desativa a combinação <CTRL> <STOP>. Quando STOP ON é executado, BASIC verifica-se <CTRL> <STOP> foi pressionado toda vez que executa um novo comando. Caso <CTRL> <STOP> tenha sido detectado, então BASIC será desviado à sub-rotina especificada pelo comando ON STOP GOSUB, executado anteriormente no programa.

Caso STOP STOP seja executado, o computador não irá até a sub-rotina especificada se <CTRL><STOP> for pressionado, mas o computador irá imediatamente à sub-rotina quando o próximo STOP ON for executado. STOP OFF pára completamente a detecção de <CTRL><STOP>.

Sintaxe

```
STOP ON
STOP STOP
STOP OFF
```

Exemplo

Este programa mostra como ON STOP GOSUB pode impedir que um usuário irrompa em seu programa. A linha 20 ativa a detecção de <CTRL><STOP>. Se você pressionar <CTRL><STOP>, ele dirá "control stop desativado" e continuará a execução do programa. Esta rotina tem um mecanismo de saída especial. Pressione "s" para desativar <CTRL><STOP> como cilada. Caso contrário, irá imprimir MSX indefinidamente.

```
10 ON STOP GOSUB 100
20 STOP ON
30 IF INKEY$="S" THEN STOP OFF :PRINT "<CTRL><STOP>ati
vado"
40 PRINT "MSX"
```

```
50 GOTO 30
100 BEEP
110 PRINT "<CTRL><STOP> desativado"
120 RETURN
```

LINHA 10 ESPECIFICA A POSIÇÃO DA SUB-ROTINA <CTRL> <STOP>.
LINHA 20 ATIVA O DETECTOR.
LINHA 30 DESATIVA O DETECTOR E FORNECE UMA MENSAGEM.
LINHA 40 IMPRIME MSX.
LINHA 50 VOLTA PARA 30.
LINHA 100 SOM DE AVISO.
LINHA 110 MENSAGEM.
LINHA 120 VOLTA AO PONTO EM QUE <CTRL><STOP> FOI DETECTADO.

Pontos a serem lembrados

Observe que ON STOP não impede a interrupção da tecla <STOP>. Apenas impede que você interrompa um programa. Isto é tudo. Tente pressionar apenas a tecla <STOP> no programa acima. Você verá que ela vai parar a execução do programa e apresentar o cursor. Se você pressionar a tecla <STOP>pela segunda vez, o programa continuará.

Lembre-se de executar STOP ON para ativar a detecção do <CTRL><STOP>. A interrupção <CTRL><STOP> será desativada quando o programa não estiver sendo processado e também durante as rotinas para detectar erros.

Corrigindo erros

Você tem de ter um comando ON STOP GOSUB e uma sub-rotina para cuidar do <CTRL><STOP>.

Comandos associados e referências

ON STOP GOSUB

Parte 2, Capítulo 37 (Vol. 1): "Tratamento de Intervalos e Interrupções pelo BASIC".

STR$

Descrição

STR$ converte o argumento numérico em uma string.

Sintaxe

STR$ (<numérico>)

Exemplo

```
10 A$="PI="+STR$(3.14)
20 PRINT A$

RUN
PI= 3.14
```

Pontos a serem lembrados

A função oposta ao STR$ é executada por VAL.

Corrigindo erros

O argumento tem de ser numérico.

Comandos associados e referências

VAL

Parte 1, Capítulo 15 (Vol. 1): "Funções".
Parte 2, Capítulo 31 (Vol. 1): "Conversão de Tipos de Variáveis".

STRIG STATUS DO GATILHO

Descrição

Esta função devolve o status do gatilho de um *joystick*.

Sintaxe

STRIG(<n>)

<n> = 0	A barra de espaço é utilizada como gatilho.
<n> = 1,3	*Joystick* 1.
<n> = 2,4	*Joystick* 2.
STRIG(<n>) = 0	não pressionado.
STRIG(<n>) = –1	pressionado.

Exemplo

Caso você pressione a barra de espaço, obterá uma mensagem:

```
10 IF STRIG(0) THEN PRINT "espaco pressionado"
20 GOTO 10
```

Pontos a serem lembrados

STRIG é utilizado mais nos jogos do tipo flipperama.

Comandos associados e referências

STICK

Parte 2, Capítulo 51 (Vol. 1): "Periféricos".

STRIG ON/OFF/STOP

Descrição

Este comando é bem diferente do comando STRIG, que devolve o status dos botões tipo gatilho. STRIG(<n>) ON/OFF/STOP ativa/desativa as rotinas de tratamento, dos botões tipo gatilho. Quando STRIG(<n>) ON é executado, o BASIC começa a verificar se o botão do tipo gatilho <n> foi pressionado, cada vez que executa um novo comando. Caso o gatilho <n> tenha sido detectado, então BASIC é desviado à sub-rotina especificada pelo comando ON STRIG GOSUB executada anteriormente no programa.

Caso STRIG(<n>) STOP seja executado, o computador não irá até a sub-rotina especificada quando o gatilho <n> for pressionado, mas se lembrará se o gatilho <n> foi pressionado, e o computador irá imediatamente até a sub-rotina quando o próximo STRIG(<n>) ON for executado.

STRIG(<n>) OFF vai parar completamente a detecção do gatilho <n>.

Sintaxe

STRING(<n>)ON
STRIG(<n>)STOP
STRIG(<n>)OFF
<n> é o número do gatilho.

Existem cinco gatilhos:

0 = Barra de espaço.
1 = gatilho do *joystick* 1.
2 = gatilho do *joystick* 2.
3 = gatilho do *joystick* 1.
4 = gatilho do *joystick* 2.

Exemplo

Aqui está um exemplo de um pequeno programa que demonstra como funciona a

interrupção por meio de um gatilho tendo a barra de espaço como gatilho. Quando a barra de espaço for pressionada, o programa irá até a rotina para pegar o gatilho. Caso contrário, imprimirá MSX continuamente até que a tecla <s> seja pressionada.

```
10 ON STRIG GOSUB 100
20 STRIG(0) ON
30 IF INKEY$="s" THEN END
40 PRINT "MSX"
50 GOTO 30
90 REM Rotina da barra de espaco
100 BEEP
110 PRINT "barra de espaco pressionada"
120 RETURN
```

LINHA 10 COLOCA A ROTINA DE GATILHO NA LINHA 100.
LINHA 20 LIGANDO.
LINHA 30 CASO <S> SEJA OPERADO, ENTÃO É O FIM.
LINHA 40 IMPRIME MSX.
LINHA 50 VOLTA À LINHA 30.
LINHA 100 SOM DE AVISO.
LINHA 110 FORNECE UMA MENSAGEM.
LINHA 120 VOLTA AO PONTO EM QUE A INTERRUPÇÃO FOI DETECTADA.

```
MSX
MSX
MSX
MSX
MSX
MSX
MSX
MSX
barra de espaco pressionada
MSX
MSX
MSX
MSX
MSX
MSX
MSX
MSX
MSX
MSX
MSX
MSX
MSX
```

Pontos a serem lembrados

A interrupção STRIG é desativada quando programa não está sendo processado e também durante as rotinas para detectar erros.

Corrigindo erros

Um erro comum:
STRIG (0)ON... está errado. Não deveria haver espaço entre STRIG e (0).

Caso você coloque o número de linha errado no comando ON STRIG GOSUB, obterá um erro de número de linha indefinido.

Comandos associados e referências

ON STRIG GOSUB
STRIG ON/OFF/STOP

Parte 2, Capítulo 37 (Vol. 1): "Tratamento de Intervalos e Interrupções pelo BASIC".

STRING$

Descrição

Este comando devolte uma string formada por uma determinada quantidade de caracteres especificados.

Sintaxe

STRING$(<n>,<código ASCII>)

Isto devolte uma string formada por <n> caracteres dados pelo <código ASCII> especificado.

STRING$(<comprimento>, <string>)

Devolve uma string de n caracteres, sendo que todos os caracteres são o primeiro caractere da variável string especificada. <n> pode ser uma variável ou uma constante numérica.

Exemplo

```
10 INPUT A
20 PRINT STRING$(A,"X")
30 GOTO 10


RUN
 10
XXXXXXXXXX
 30
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Pontos a serem lembrados

STRING$ é útil para definir um sprite quadrado:

```
SPRITE$(0)=STRING$(8,CHR$(255))
```

Corrigindo erros

Ocorre um erro "Out of String Space" (sem espaço de string) quando o comprimento da string excede o tamanho do espaço de string, que possui 200 caracteres quando o micro é ligado.

Comandos associados e referências

Parte 1, Capítulo 20 (Vol. 1): "O Código de ASCII".

SWAP

Descrição

Troca o conteúdo de duas variáveis.

Sintaxe

SWAP <variável>,<variável>

Exemplo

```
10 A=100 : B=200
20 C$="MSX" : D$="ASCII"
30 SWAP A,B
40 SWAP C$,D$
50 PRINT A,B
60 PRINT C$,D$
```

```
RUN
 200          100
 ASCII        MSX
```

Corrigindo erros

Ocorre um erro de desencontro de tipos caso você tente utilizar o SWAP com diferentes tipos de variáveis.

Comandos associados e referências

Parte 1, Capítulo 6 (Vol. 1): "O Uso de Condições".

TAB

Descrição

TAB é sempre utilizado em conjunto com os comandos PRINT ou LPRINT e posiciona o cursor na posição especificada, contado a partir do lado esquerdo da tela.

Sintaxe

PRINT TAB(<numérico>)
LPRINT TAB(<numérico>)

Exemplo

```
10 B=100
20 PRINT TAB(5)"B=" TAB(10) B

RUN
     B=     100
```

Pontos a serem lembrados

<numérico> tem de ser menor que WIDTH–1 ou irá para a próxima linha.

Caso <numérico> seja um número real, então o ponto decimal será retirado para fornecer um inteiro.

O comando LOCATE é muito mais flexível que o TAB, uma vez que você pode especificar as coordenadas x e y.

Corrigindo erros

Não deve haver nenhum espaço entre TAB e o parênteses. Por exemplo: PRINTTAB (10) gera um erro.

Uma chamada ilegal de função ocorrerá se o argumento numérico ultrapassar 255.

Comandos associados e referências

PRINT
LPRINT
LOCATE

Parte 1, Capítulo 9 (Vol. 1): "Mais sobre o Comando PRINT e o Modo de Tela".

TAN TANGENTE

Descrição

Retorna a tangente do argumento em radianos. TAN é calculado com precisão dupla.

Sintaxe

TAN(<numérico>)

Exemplo

```
PRINT TAN(0.5)
.54630248984381
```

Corrigindo erros

O argumento tem de ser numérico. Caso seja uma string, ocorrerá um erro de desencontro de tipo.

Comandos associados e referências

ATN
COS
SIN

Parte 1, Capítulo 19 (Vol. 1): "Funções Matemáticas".

THEN

Descrição

Este comando é utilizado com o comando IF, sendo que freqüentemente são referenciados como comandos IF THEN. O comando depois do THEN é executado quando as condições depois do IF são satisfeitas. Caso haja um número de linha depois de THEN, será entendido como uma abreviação de GOTO <linha>.

Você pode usar diversos comandos após o THEN, desde que separados por dois-pontos, se desejar que o computador execute diversas ações caso as condições no IF sejam satisfeitas.

Sintaxe

IF <condições> THEN <comandos>
IF <condições> THEN <comandos> ELSE <declarações>
IF <condições> THEN <linha>

Exemplo

```
10 INPUT X
20 IF X=10 THEN PRINT "SI"
30 GOTO 10


RUN
? 12
? 10
SI
```

Pontos a serem lembrados

THEN pode ser substituído por um comando GOTO apenas quando você estiver desviando o programa para outra linha.

IF X = 1 GOTO 2000

Comandos associados e referências

IF
GOTO
ELSE

Parte 1, Capítulo 6 (Vol. 1): "O Uso de Condições".
Parte 2, Capítulo 35 (Vol. 1): "Álgebra Booleana II: o IF/THEN/ELSE".

TIME

Descrição

A função TIME devolve o valor do temporizador interno do sistema. O temporizador é colocado em zero quando a máquina é ligada e é incrementado automaticamente toda vez que processador de display de vídeo (VDP) gera uma interrupção. Isto acontece 60 vezes em um segundo.

Você pode atribuir qualquer valor inteiro ao temporizador utilizando a função TIME. Também pode zerar o temporizador se desejar.

Sintaxe

```
TIME
```

Exemplo

Relógio digital

```
10 CLS
20 TIME=0
30 MIN%=TIME/3600
40 SEG%=TIME/60-MIN%*60
50 LOCATE 10,11 : PRINT "MINUTOS";MIN%
60 LOCATE 10,12 : PRINT "SEGUNDOS";SEG%
70 GOTO 30
```

Resultado: Aparecerá um relógio digital no meio do modo de tela.

```
RUN
                MINUTOS 1
                SEGUNDOS 45
```

Pontos a serem lembrados

Quando a interrupção do VDP é desativada, o relógio pára. Isto ocorre quando você utiliza um dispositivo externo como um cassete para carregar programas.

TIME sempre devolve um inteiro positivo.

TIME é utilizado em RND(–TIME) para randomizar o gerador de números aleatórios.

Comandos associados e referências

RND

Parte 1, Capítulo 15 (Vol. 1): "Funções".

TO

Descrição

TO é parte da estrutura FOR...TO...STEP...NEXT e é sempre utilizado depois do FOR. O valor inicial é dado antes de TO e o valor final, de encerramento do loop, é dado depois do TO.

Sintaxe

FOR <var-num> = <numérico> TO <numérico>
FOR <var-num> = <numérico> TO <numérico> STEP <numérico>

Exemplos

```
10 FOR I=1 TO 2
20 PRINT I
30 NEXT I


RUN
 1
 2
```

Pontos a serem lembrados

Você realmente não necessita de um espaço entre o TO e o valor numérico, mas é mais fácil de se ler uma linha que utiliza espaços:

FOR I = 100TO200

Comandos associados e referências

FOR
STEP
NEXT

Parte 1, Capítulo 5 (Vol. 1): "O Uso de Loops".

TROFF DESLIGAMENTO DE RASTREIO

Descrição

Pára TRON ou pára o rastreio.

TROFF tem de ser utilizado para interromper a rastreio. Não há outra maneira de sair do TRON a não ser utilizando TROFF.

Sintaxe

TROFF

Pontos a serem lembrados

Para utilizar a facilidade de rastreio, utilize TRON.

Comandos associados e referências

TRON

Parte 1, Capítulo 7 (Vol. 1): "Comandos Úteis e Dicas para Escrever Programas".

TRON RASTREIO LIGADO

Descrição

TRON faz com que o computador imprima o número da linha que estiver sendo executada.

TRON é utilizado apenas para depuração de programas (debug). Você recorre a ele apenas quando seu programa se tornou um programa "espaguete" e não é possível segui-lo de outra forma!

Sintaxe

TRON

Exemplo

```
10 FOR I=1 TO 2
20 PRINT I
30 NEXT I
```

```
TRON
OK
RUN
[10][20] 1
[30][20] 2
[30]
Ok
```

Pontos a serem lembrados

TRON cria confusão na tela quando da impressão de um excessivo número de linhas. TRON não funciona no modo gráfico.

Para desligar TRON, utilize TROFF.

Corrigindo erros

É utilizado para procurar erros em programas.

Comandos associados e referências

TROFF

Parte 1, Capítulo 7 (Vol. 1): "Comandos Úteis e Dicas para Escrever Programas".

USR

Descrição

USR chama uma sub-rotina em linguagem de máquina do usuário a partir do BASIC.

A sub-rotina em linguagem de máquina tem sua posição de início definida pela função DEFUSR.

Utilizando esta função, você pode passar dados de/para sub-rotinas em linguagem de máquina.

Sintaxe

```
USR[<dígito>](<argumento>)
```

<dígito> de 0 a 9

<argumento> (qualquer tipo) a ser passado para rotina em linguagem de máquina.
máquina.

Exemplo

```
PRINT USRO ("TESTE")
DMY=USR4(B%)
X=USR9(1000)
```

Pontos a serem lembrados

Não é possível explicar como utilizar linguagem de máquina nesta parte, de modo que você deve ir à parte de como utilizar a função USR para maiores detalhes.

Corrigindo erros

Quando estiver utilizando a sua própria rotina em linguagem de máquina, tome cuidado. Sempre utilize CSAVE em seu programa antes da execução, porque se a máquina travar devido a uma falha na sub-rotina em linguagem de máquina, existirá pouca ou nenhuma chance de recuperar o programa perdido.

Comandos associados e referências

DEFUSR

Parte 2, Capítulo 49 (Vol. 1): "O Comando USR e a Linguagem de Máquina".

VAL VALOR

Descrição

VAL devolve o valor de uma string que contém um número. Caso uma string seja a representação string de um número, então a mesma pode ser avaliada. Entretanto, VAL não funciona em uma expressão interna a uma string.

VAL ignorará espaços e tabulações antes de um número em uma string (veja os exemplos). Não consegue, entretanto, trabalhar com outros caracteres antes de um número, por exemplo, VAL("ZXY100") não fornecerá 100.

VAL reconhece os sinais de mais e de menos.

Sintaxe

VAL(<string>

Exemplo

```
PRINT VAL("                    – 100")
– 100
PRINT     VAL("GARBAGE     1000")
0
A$="+ 10" :PRINT VAL(A$)
10
```

Pontos a serem lembrados

A função oposta a VAL é executada por STR$.

Corrigindo erros

VAL("–100+900") devolverá apenas –100, uma vez que VAL não consegue resolver expressões dentro de uma string.

VAL(A%) causará um erro de desencontro de tipo.

Comandos associados e referências

STR$

Parte 1, Capítulo 15 (Vol. 1): "Funções".
Parte 2, Capítulo 31 (Vol. 1): "Conversão de Tipos Variáveis".

VARPTR

Descrição

VARPTR(<nome da variável>) fornece o endereço do primeiro byte dos dados identificados com o nome da variável dada. A variável pode ser de qualquer tipo, mas obviamente aquela variável terá de existir antes que você utilize VARPTR.

Poderá também devolver o endereço do começo de um bloco de controle de arquivo.

O endereço devolvido será um inteiro na faixa –32768 a 32767.

Caso retorne um número negativo, acrescente 65536 para conseguir o endereço correto.

Sintaxe

```
VARPTR(<nome da variável>)
VARPTR(#<nome do arquivo>)
```

Exemplo

```
PRINT VARPTR (A(0))
S$="IIII" :PRINT 65536 + VARPTR(S$)
D%= 100 :PRINT "H";HEX$(65536 + VARPTR(D%))
```

Pontos a serem lembrados

VARPTR algumas vezes é utilizado para a obtenção do endereço de uma matriz, para que possa ser passado para uma sub-rotina em linguagem de máquina. Uma chamada de função da forma de VARPTR(0)) é normalmente especificada ao passar uma matriz, de modo que o endereço mais baixo da matriz é devolvido.

Corrigindo erros

Caso a variável referida no argumento não exista, resultará num erro de chamada de função ilegal.

Comandos associados e referências

Parte 2, Capítulo 48 (Vol. 1): "Mapa da Memória".

VDP REGISTRADOR DO PROCESSADOR DE DISPLAY DE VÍDEO

Descrição

VDP é utilizado para acessar o processador de display de vídeo. Existem 9 registros:

0 a 7 são apenas de escrita, de modo que você pode estabelecer novos valores para eles.
8 é apenas de leitura.

Os detalhes dos registradores VDP são fornecidos detalhadamente na Parte 2, Capítulo 44 (Vol. 1).

Sintaxe

Para registros 0 a 7, <dígito> = 0 a 7

VDP(<dígito>) = <numérico>

Registro 8

<var-num> =VDP(8)

Exemplo

```
PRINT BIN$(VDP(8))
```

O exemplo imprime a representação binária do registrador de número 8 do VDP.

Corrigindo erros

Caso você utilize a função VDP de forma errada, possivelmente criará confusão na tela, a não ser que saiba o que está fazendo. Caso a tela fique confusa, desligue e ligue o micro.

Comandos associados e referências

Parte 2, Capítulo 44 (Vol. 1): "Gráficos Avançados V".

VPEEK

Descrição

VPEEK devolve o conteúdo de um byte da VRAM (RAM de vídeo).

Sintaxe

```
VPEEK(<endereço>)
<endereço> de 0 até 16383
```

Exemplo

```
PRINT VPEEK(100)
V%=VPEEK(999)
```

Pontos a serem lembrados

É aconselhável que esta função seja utilizada apenas por usuários bem experientes.

Uma vez que a RAM de vídeo é separada da RAM da memória principal, você necessita desta função especial para acessá-la.

Todos os computadores MSX têm pelo menos 16K bytes de RAM de vídeo.

VPEEK é o comando complementar de VPOKE.

Corrigindo erros

Resultará uma chamada de função ilegal se o <endereço> estiver fora da faixa limite.

Comandos associados e referências

VPOKE

Parte 1, Capítulo 45 (Vol. 1): "Gráficos Avançados VI".

VPOKE ALTERANDO RAM DE VÍDEO

Descrição

VPOKE coloca um byte na RAM de vídeo. Você tem de fornecer o endereço da RAM de vídeo, e o byte de dado que deve ser colocado ali.

O valor dos dados tem de estar entre 0 e 255.

Sintaxe

VPOKE<endereço>,<byte>

<endereço> faixa de 0 a 16383

<byte> faixa de 0 a 255

Exemplos

```
VPOKE 998,255
```

Pontos a serem lembrados

Não o utilize a não ser que esteja seguro do que está fazendo.

Uma vez que a RAM de vídeo é separada da RAM da memória principal, você necessita desta função especial para acessá-la.

Todos os computadores MSX têm pelo menos 16Kbytes de RAM de vídeo.

VPOKE é o comando complementar do VPEEK.

Corrigindo erros

Caso <endereço> esteja fora da faixa especificada, isso resultará num erro de extravasamento.

Caso <byte> esteja fora da faixa especificada, isso resultará uma chamada de função ilegal.

Comandos associados e referências

VPEEK

Parte 2, Capítulo 45 (Vol. 1): "Gráficos Avançados VI".

WAIT

Descrição

WAIT suspende a execução de um programa e aguarda até que uma determinada porta de entrada, especificada, apresente um padrão especificado de bit.

Sintaxe

WAIT<número da porta>,<I>[,<J>

Faixa:

<número da porta> = 0 até 255
<I> e <J> = 0 até 255

Pontos a serem lembrados

Aplicamos a função XOR entre os dados lidos à porta e J; aplicamos a função AND entre os dados lidos à porta e I. Caso o resultado seja zero, BASIC volta e lê novamente os dados colocados na porta. Caso J seja omitido, então J=0.

Caso o resultado seja diferente de zero, a execução do programa começará novamente.

Este comando acessa diretamente a porta de E/S sem passar pela BIOS. Portanto, é bem possível que se você utilizar este comando em um programa, o mesmo perderá sua compatibilidade MSX e se tornará dependente da máquina.

MSX utiliza apenas portas entre 0 e 255; qualquer coisa acima de 255 não tem sentido.

Comandos associados e referências

OUT
INP

WIDTH

Descrição

O comando WIDTH estabelece a largura da tela, limpando-a também após a sua execução.

modo de tela	máximo
tela 0	40
tela 1	32

Sintaxe

WIDTH<numérico>

Exemplos

```
WIDTH 30
WIDTH 20
```

Pontos a serem lembrados

A largura mínima é 1.

Corrigindo erros

Resultará numa chamada de função ilegal quando o argumento estiver fora da faixa permitida.

Comandos associados e referências

Parte 1, Capítulo 21 (Vol. 1): "Modos de Tela".

XOR OR EXCLUSIVO

Descrição

XOR, OR EXCLUSIVO é um dos operadores lógicos que executam álgebra booleana.

XOR	X	Y	X XOR Y
	0	0	0
	1	0	1
	0	1	1
	1	1	0

Portanto 100 XOR 50, pode ser calculado como segue:

	X	100	0000000001100100
XOR	Y	50	0000000000110010
		86	0000000001010110

Sintaxe

<var-num>=<numérico> XOR <numérico>

Pontos a serem lembrados

A seguinte relação é verdadeira:

A = A XOR Y XOR Y

Corrigindo erros

Ocorre um erro de extravasamento se o argumento estiver fora da faixa dos inteiros.

Comandos associados e referências

AND, OR, EQV, NOT, IMP

Parte 2, Capítulo 34 (Vol. 1): "Álgebra Booleana I: Expressões Lógicas".

Parte IV

SISTEMA OPERACIONAL E BIOS

Esta parte trata da BIOS (BASIC INPUT/OUTPUT SYSTEM), (Sistema Básico de Entrada/Saída); explica como utilizar os pontos de entrada e fornece uma listagem da área RAM do sistema. Para compreender essa parte do livro, o leitor deverá ter um conhecimento de programação em linguagem assembly no Z80. Não é feita nenhuma tentativa para ensinar isto.

Os capítulos 53 a 59 descrevem as chamadas a BIOS disponíveis em qualquer computador MSX. Chamadas relacionadas são agrupadas em cada um destes capítulos. O formato de cada descrição de ponto de entrada BIOS foi padronizado. A primeira linha contém o nome da rotina seguido pelo seu endereço de ponto de entrada em hexadecimal. A linha seguinte fornece o comando do Z80, que deve ser executado para chamar a rotina BIOS. Em seguida a isto, vem uma explicação da função da chamada e a informação de como passar e receber parâmetros das rotinas BIOS. A parte seguinte contém uma listagem dos registradores e posições de memória do Z80 que podem ser alterados como uma conseqüência da chamada da rotina. A última parte fornece informação referente ao estado das interrupções e utilizações possíveis da chamada.

O capítulo 60 fornece uma explicação sobre os hooks, seguidos de uma relação alfabética de cada hook, com seu endereço e de onde e chamado.

O capítulo final contém uma relação alfabética das posições da RAM do sistema com seus endereços e tamanhos (em bytes).

CAPÍTULO 53

COMANDOS RST

Este capítulo trata com os oito comandos que podem ser executados utilizando o comando RST do Z80:

RST0	CHKRAM
RST1	SYNCHR
RST2	CHRGTR
RST3	OUTDO
RST4	DCOMPR
RST5	GETYPR
RST6	CALLF
RST7	KEYINT

CHKRAM 0000

Executada por meio da RST 00.

Quando o computador é ligado, é aqui que a execução começa. Pular para este endereço por qualquer meio causará uma reinicialização total.

PARÂMETROS DE ENTRADA

Nenhum parâmetro de entrada é requerido.

PARÂMETROS DE SAÍDA

Eles não são aplicáveis, pois o computador nunca retorna após uma chamada para este endereço.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

Todos os registros são alterados e qualquer dado armazenado na memória está perdido.

UTILIZAÇÕES POSSÍVEIS

Existe apenas uma utilização para esta "rotina": inicializar o computador.

SYNCHR 0008

Executada por meio de RST1.

Esta rotina é utilizada pelo interpretador BASIC para verificar a existência de erros de sintaxe. Caso nenhum seja encontrado, o processamento continuará com a chamada CHRGTR(0010). Caso contrário, será gerado um "erro de sintaxe".

PARÂMETROS DE ENTRADA

HL aponta para o próximo caractere de texto BASIC, e o byte que segue ao comando RST1 contém o caractere com o qual deverá ser comparado.

PARÂMETROS DE SAÍDA

No retorno, HL aponta para o próximo caractere e A tem aquele caractere. O flag de carry é setado se o caractere é um número, e o flag de zero é setado se chegamos ao fim da função.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

O acumulador, flags e o par de registros HL são afetados com esta chamada.

Nota: Esta chamada não tem utilidade para programadores em código de máquina.

CHRGTR 0010

Executado por meio de RST2.

Este é utilizado pelo interpretador BASIC para pegar o próximo caractere ou token do programa BASIC. É normalmente chamado da rotina SYNCHR (RST1).

PARÂMETROS DE ENTRADA

HL tem de apontar ao caractere que será pego.

PARÂMETROS DE SAÍDA

No retorno, HL aponta para o próximo caractere do programa, o flag de carry é setado se um número é lido, e o flag de zero é setado se o fim da função atual é encontrado.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

AF e HL são alterados por este RST.

Nota: Da mesma forma que com SUNCHR (RST1), esta rotina não é util para programadores em linguagens de máquina.

OUTDO 0018

Executado por meio de RST3.

Este comando escreve o conteúdo do acumulador no dispositivo atualmente selecionado.

PARÂMETROS DE ENTRADA

O acumulador deve conter o código que vai ser escrito. Se o código é escrito em um arquivo em disco, PTRFIL deve conter o ponteiro àquele arquivo. Se os dados são impressos na impressora, então PTRFIL deve conter zero e PRTFLG deve ser não-zero. Se o código é enviado ao terminal, então tanto PTRFIL como PTRFLG devem ser zero.

PARÂMETROS DE SAÍDA

Nenhum parâmetro é devolvido.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

Nenhum registro ou posição de memória é afetado por esta chamada.

Nota: Depois de colocar AF na pilha (stack), esta rotina chama o hook H.OUTD, antes de continuar a sua execução.

DCOMPR 0020

Executado por meio de RST4.

Esta chamada compara o par de registro HL e DE do Z80.

PARÂMETROS DE ENTRADA

Nenhum parâmetro de entrada é requerido, além dos pares de registros HL e DE.

PARÂMETROS DE SAÍDA

No retorno, o flag de carry será setado se HL for menor que DE; caso contrário, será resetado. Ainda, o flag de zero será setado se HL = DE.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

Apenas o acumulador e ponteiros são afetados por este comando.

GETYPR 0028

Executado por meio de RST5.

Este comando é utilizado pelo interpretador BASIC para descobrir que tipo de acumulador de ponto flutuante está sendo utilizado.

PARÂMETROS DE ENTRADA

Nenhum parâmetro de entrada é requerido.

PARÂMETROS DE SAÍDA

Diversos flags são estabelecidos de acordo com o tipo de acumulador de ponto flutuante atualmente em uso.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

Apenas os flags são afetados por esta rotina.

CALLF 0030

Executado por meio de RST6.

RST6 executa uma chamada entre slots.

PARÂMETROS DE ENTRADA

O byte que segue o comando RST6 contém o byte de descrição de slot, e os dois bytes seguintes contêm o endereço que deve ser chamado.

PARÂMETROS DE SAÍDA

Estes dependem da rotina chamada.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

Estes dependem da chamada à rotina.

Nota: O byte de descrição do slot tem o seguinte formato:

Nº do bit	7 6 5 4 3 2 1 0
Conteúdo	F x x x S S P P

onde:

PP (0-3) é o número do slot primário a selecionar.
SS (0-3) é o número do slot secundário.
F (0-1) é um flag colocado em 1 se os slots secundários estiverem em uso e 0 se não estiverem.

Bits 4-6 não são utilizados.

KEYINT 0038

Executado por meio de RST7 ou uma interrupção.

O sistema MSX opera no modo 1 de interrupção, de maneira que este é o comando·RST executado quando ocorre uma interrupção mascarada. Ocorrem interrupções a cada 1/60s devido ao temporizador de 60Hz. Quando isto ocorre, o contador de intervalo é decrementado, JIFFY é incrementado, as "queues" musicais são atualizadas, os gatilhos dos *joystick* são verificados e é executada uma varredura do teclado.

PARÂMETROS DE ENTRADA

Por sua natureza (isto é, interrupção de hardware), não poderá haver nenhum parâmetro de entrada a esta rotina.

PARÂMETROS DE SAÍDA

Vários eventos podem ser setados (isto é, ON SPRITE), e outra tecla poderá ser colocada no buffer do teclado.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

Todos os registros permanecem inalterados, mas muitas posições de memória do sistema são afetadas por esta chamada.

Nota: O hook H.KEYI é chamado imediatamente após serem colocados todos os registros na pilha (stack), permitindo que outras interrupções sejam processadas.

CAPÍTULO 54

PONTOS DE ENTRADA ASSOCIADOS À MANIPULAÇÃO DE CONECTORES

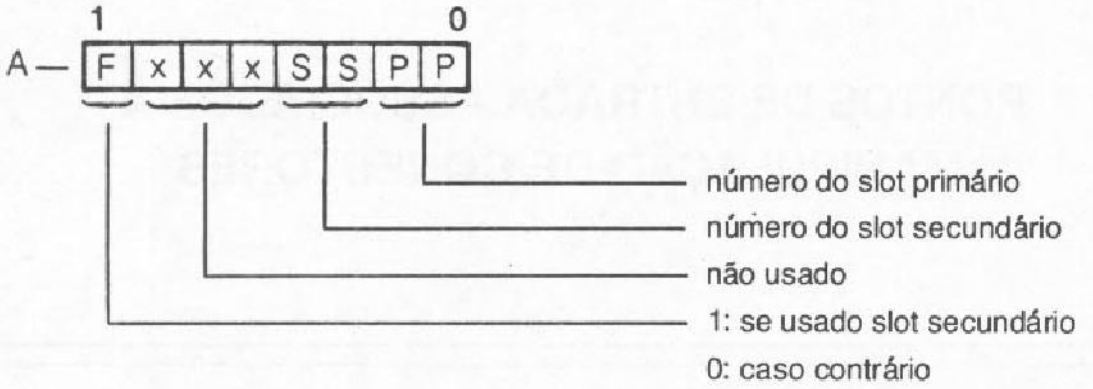
Os pontos de entrada da BIOS descritos neste capítulo são utilizados para gerenciar o sistema de slots (veja Parte 2, Capítulo 50 (Vol. 1), "Gerenciamento de Memória do MSX e o Mecanismo dos Cartuchos" para maiores detalhes).

RDSLT 000C

Esta chamada é utilizada para ler uma posição de memória em qualquer slot.

PARÂMETROS DE ENTRADA

O par de registros HL deve conter o endereço da posição que será lida e o acumulador deve especificar de qual slot deve ser feita a leitura. O valor do acumulador tem o seguinte significado: os dois bits menos significativos contêm o número do slot primário (0-3), que deve ser utilizado; os dois próximos bits contêm o número do slot secundário (0-3); e os três bits seguintes não são utilizados e, se um slot secundário tiver sido especificado, então o bit mais significativo tem de ser setado; caso contrário deve ser resetado.



PARÂMETROS DE SAÍDA

O acumulador tem o conteúdo da posição de memória.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

Os registradores AF, BC e DE são afetados por esta chamada.

Nota: Esta rotina deativa todas as interrupções. Uma tentativa de selecionar a página 3 de um slot, utilizando esta rotina, causará o "crash" do computador (veja ENASLT para uma solução a este problema).

WRST 0014

Executado via CALL &H0014.

Esta chamada é utilizada para escrever em qualquer posição de memória em qualquer slot.

PARÂMETROS DE ENTRADA

O par de registro HL deve conter o endereço da posição onde será escrito; o acumulador deve conter um número especificando em que slot se encontra, e o registro E, o número a ser escrito. O conteúdo do acumulador toma o mesmo formato que aquele descrito para o ponto de entrada RDSLT (OOOC).

PARÂMETROS DE SAÍDA

Nenhum parâmetro é devolvido desta rotina.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

Os registros AF, BC e D são afetados por esta chamada.

Nota: Esta rotina desativa todas as interrupções. A tentativa de selecionar a página 3 de um conector, utilizando esta rotina, fará com que ocorra um "crash" do computador (veja ENASLT para uma solução a este problema).

CALSLT 001C

Executado por meio de CALL &H001C.

Esta rotina executa uma chamada entre slots, selecionando uma página em outro slot e depois chamando um endereço ali.

PARÂMETROS DE ENTRADA

O registrador IX deve conter o endereço que vai ser chamado, e o byte mais significativo do registrador indexado IY deve especificar o conector. O formato requerido do byte mais significativo de IY é idêntico ao formato do acumulador para o ponto de entrada RDSLT. Parâmetros podem ser passados nos registradores AF, BC, DE e HL, mas não nos registros alternativos.

PARÂMETROS DE SAÍDA

Nenhum parâmetro será devolvido a não ser que sejam criados pela chamada entre-slots. Parâmetros podem ser devolvidos em qualquer dos registros, exceto no acumulador alternativo, uma vez que este contém o status do mecanismo de seleção do slot antes da chamada.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

Registros AF', BC', DE' e HL' são alterados antes que a chamada entre-slots seja executada.

Nota: Interrupções são desativadas por esta rotina. Esta chamada pode também ser executada por meio do comando RST6 (veja CALLF, Parte 2, Capítulo 29 (Vol. I). A tentativa em selecionar a página 3 de um slot, utilizando esta rotina, fará um "crash" no computador (veja ENASLT para uma solução a este problema).

ENALST 0024

Executado por intermédio de CALL &H0024.

Esta chamada seleciona uma página em um dos slots.

PARÂMETROS DE ENTRADA

Os dois bits mais significativos do registro H são utilizados para selecionar a página apropriada como segue:

MSB	Seleção de página
00	0000-3FFF
01	4000-7FFF
10	8000-BFFF
11	C000-FFFF

O acumulador deve especificar o slot que vai ser selecionado, conforme explicado no ponto de entrada RDSLT.

PARÂMETROS DE SAÍDA

Nenhum parâmetro será devolvido.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

Os registros AF, BC, DE e HL são afetados por esta chamada.

Nota: Esta rotina desativa todas as interrupções.

O problema de não ser possível selecionar a página 3 de qualquer conector, nos pontos de entrada RDLST, WRSLT e CALSLT, pode ser contornado utilizando esta chamada. Por exemplo, ler o endereço $HD000 no slot 3, utilize os seguintes códigos:

```
CALL    &H0138      ;LEIA O REGISTRO DE SELEÇÃO DO SLOT PRIMÁRIO.
PUSH    AF          ;GRAVE-O.
LD      HL,&HD000   ;O ENDEREÇO DE MEMÓRIA A SER LIDO.
PUSH    HL          ;GRAVE ESTE ENDEREÇO.
LD      A,3
DI                  ;INTERRUPÇÕES TÊM DE SER DESATIVADAS UMA
                    VEZ QUE ALTERAM A PÁGINA 3.
CALL    &H0024      ;ATIVA SLOT 3, PÁGINA 3.
POP     HL          ;RESTAURA ENDEREÇO.
LD      H,(HL)      ;OBTÉM CONTÉUDO DESEJADO.
POP     AF
CALL    &H013B      ;REATIVA PÁGINA 3 DO SISTEMA.
EI
LD      A,H         ;DEVOLVE O VALOR DESEJADO NO ACUMULADOR.
RET
```

Métodos semelhantes podem ser utilizados para se chamar uma sub-rotina na página 3 de um conector diferente.

RSLREG 0138

Executado por meio de CALL &H0138.

Esta chamada lê o conteúdo do registrador de seleção do slot primário.

PARÂMETROS DE ENTRADA

Nenhum parâmetro de entrada é necessário.

PARÂMETROS DE SAÍDA

Uma cópia do registro de seleção do slot primário é devolvida no acumulador.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

Apenas o acumulador é alterado por esta chamada.

Nota: Esta rotina é simplesmente um comando IN (IN &HA8 provavelmente), seguida por um RET.

WSLREG 013B

Executado por meio de CALL &H013B.

Esta chamada escreve no registro de seleção do slot primário.

PARÂMETROS DE ENTRADA

O acumulador deveria conter o valor a ser escrito.

PARÂMETROS DE SAÍDA

Nenhum parâmetro é devolvido por esta rotina.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

Nenhum registro ou posição de memória é alterado por esta chamada.

Nota: Esta rotina é simplesmente um comando OUT (OUT &HA8 provavelmente), seguido por RET.

CALBAS 0159

Executado por meio de um CALL &H0159.

Este ponto de entrada é utilizado pelo interpretador BASIC para executar uma chamada entre slots em um cartucho de expansão do interpretador BASIC.

PARÂMETROS DE ENTRADA

O endereço a ser chamado é passado pelo registro IX.

PARÂMETROS DE SAÍDA

A saída desta chamada depende da chamada entre slots.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

Como anteriormente, isto não é definido.

CAPÍTULO 55

PONTOS DE ENTRADA DA BIOS USADOS PARA ACESSAR O TECLADO, O VÍDEO E A IMPRESSORA

Este capítulo descreve as chamadas que são utilizadas para controlar o "console", isto é, o teclado, o display de texto e a impressora.

CHSNS 009C

Executado por meio de um CALL &H009C.

Esta chamada executa duas operações. Primeiro verifica o status das teclas "shift". Se uma tecla estiver sendo pressionada e o display das teclas de função estiver ativo, então as teclas de função 6-10 serão apresentadas. Em segundo lugar, esta rotina verifica o status do buffer do teclado.

PARÂMETROS DE ENTRADA

Não são necessários parâmetros de entrada.

PARÂMETROS DE SAÍDA

Caso o buffer de teclado estiver vazio, o flag Z será ativado; caso contrário, será liberado.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

Apenas o acumulador e flags são afetados por esta rotina.

Nota: As interrupções são ativadas por esta chamada.

CHGET 009F

Executado por meio de um CALL &H009F.

Esta rotina devolve um caractere do buffer de teclado. Caso o buffer esteja vazio, a rotina espera que uma tecla seja pressionada, apresentando o cursor se ativado.

PARÂMETROS DE ENTRADA

Nenhum parâmetro de entrada será requerido por esta rotina.

PARÂMETROS DE SAÍDA

O código de caractere da tecla pressionada está contida no acumulador.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

Apenas o acumulador e flags são afetados por esta chamada.

Nota: Esta rotina chama o hook H.CHGE imediatamente após colocar na pilha os registros HL, DE e BC (nesta ordem). Isto permite que outros dispositivos de entrada sejam utilizados.

CHPUT 00A2

Executado por meio de um CALL &H00A2

Esta chamada envia um caractere ao console, movendo-se para a próxima linha e

rolando a tela quando necessário. Os códigos de controle 7-13 e 27-31 inclusive são aceitos.

PARÂMETROS DE ENTRADA

O acumulador deve conter o código do caractere a ser enviado ao console. A posição do cursor na tela é armazenada nas posições de memória CSRX, CSRY.

PARÂMETROS DE SAÍDA

Na saída, a posição do cursor (CSRX, CSRY) terá sido atualizada.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

Nenhum registro é alterado, mas as posições de memória CSRX, CSRY e TTYPOS podem ser afetados.

Nota: Depois de colocar os registros HL, DE, BC e AF na pilha, esta rotina chama o hook H.CHPU, permitindo que outros dispositivos de saída sejam utilizados (por exemplo, um modo de tela de 80 colunas).

Esta rotina não faz nada se o modo gráfico II ou o modo multicolorido estiver ativo.

As interrupções são ativadas por esta chamada.

LPTOUT 00A5

Executado por meio de um CALL &H00A5.

Esta rotina envia um caractere a impressora, desde que haja uma ligada. Caso a impressora não esteja pronta para receber o caractere, a rotina vai esperar até que esteja pronta ou a tecla de STOP seja pressionada.

PARÂMETROS DE ENTRADA

O caractere a ser impresso deve ser colocado no acumulador.

PARÂMETROS DE SAÍDA

Ao voltar, o flag de carry terá sido resetado caso o caractere tenha sido impresso com sucesso, e setado se a tecla STOP foi utilizado para abortar a operação.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

Os flags poderão ser alterados por esta chamada.

Nota: A primeira coisa que esta rotina faz é chamar o hook H.LPTO, permitindo que um processamento adicional seja executado. Um exemplo disto é programar o hook para que ignore caracteres enviados à impressora, via

```
H.LPTO:   INC     SP
          INC     SP
          RET
```

O hook H.LPTO está em &HFFB6, de modo que o exemplo acima pode ser simplesmente programado utilizando os comandos BASIC:

```
POKE &HFFB6,&H33
POKE &HFFB7,&H33
```

(Para que a impressora volte a funcionar, digite POKE $HFFB6, HC0.)

LPTSTT 00A8

Executado por meio do CALL &H00A8.

Esta chamada verifica se a impressora está pronta para receber um caractere.

PARÂMETROS DE ENTRADA

Não são necessários parâmetros de entrada.

PARÂMETROS DE SAÍDA

Se a impressora estiver pronta para receber dados, o acumulador conterá 255 e o flag Z terá sido resetado, caso contrário, o acumulador conterá zero e o flag estará setado.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

Apenas o acumulador e flags são afetados por esta chamada.

Nota: LPTSTT é chamado pelo ponto de entrada LPTOUT.

A primeira coisa feita por esta rotina é chamar o hook H.LPTS.

CNVCHR 00AB

Executado por meio de CALL &H00AB.

Esta chamada converte um código de caractere no número do padrão que o representa no modo de tela do processador de vídeo. Para os códigos de caractere 32-255, estes números são os mesmos que o número do padrão daquele caractere; entretanto, os códigos de caractere 0-31 são códigos de controle, mas os números de padrão 0-31 representam os caracteres obtidos mantendo a tecla gráfica pressionada. Estes padrões são representados por dois códigos de caracteres, o código de controle 1, seguido por um caractere na faixa 64-95.

PARÂMETROS DE ENTRADA

O caractere a ser convertido deve ser colocado no acumulador.

PARÂMETROS DE SAÍDA

Quatro resultados diferentes podem ser obtidos desta chamada, dependendo do estado inicial do byte de cabeçalho do gráfico (GRPHED) e do acumulador.

1. Caso GRPHED contenha zero e o acumulador contenha código de caractere 1, o acumulador ficará inalterado, GRPHED é colocado em 1, e os flags de vai-um e zero serão liberados.

2. Caso GRPHED contenha zero e o acumulador tenha qualquer valor que não seja um, então GRPHED e o acumulador não serão alterados e C e Z serão setados.
3. Caso GRPHED contenha um número que não seja zero e o acumulador contenha um número na faixa 64-95, no encerramento o acumulador terá 64 a menos do valor inicial, HRPHED conterá zero, C será setado e Z resetado.
4. Se GRPHED contiver um número que não seja zero e o acumulador contiver um número fora da faixa 64-95, GRPHED será colocado em zero, o acumulador não será alterado e C e Z serão setados.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

O acumulador, flags e a posição de memória GRPHED poderão ser alterados por esta chamada.

PINLIN 00AE

Executado por meio de CALL &H00AE.

Esta rotina é semelhante à INLIN, mas quando o computador estiver no modo de numeração automático, a operação de <CTRL-U> não apagará o número da linha.

PARÂMETROS DE ENTRADA

Nenhum parâmetro de entrada é necessário.

PARÂMETROS DE SAÍDA

O registrador HL aponta para a posição de memória abaixo do primeiro caractere no buffer e o flag de carry é setado <CTRL> <STOP> foi pressionado.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

Os registros AF, DE e HL poderão ser alterados por esta chamada.

INLIN 00B1

Executado por meio de CALL &H00B1.

Esta rotina é utilizada pelo interpretador BASIC para aceitar uma linha do teclado, apresentando os caracteres no modo de tela à medida que são introduzidos e armazenando a linha em um buffer, até <RETURN> ou <CTRL> <STOP> ser pressionado.

PARÂMETROS DE ENTRADA

Não são necessários parâmetros de entrada.

PARÂMETROS DE SAÍDA

O registro HL aponta para a posição de memória abaixo do primeiro caractere no buffer e o flag de carry é setado se <CTRL> <STOP> foi pressionado.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

Registros AF, DE, BC e HL podem ser alterados por esta chamada.

QUINLIN 00B4

Executado por meio de CALL &H00B4.

Esta rotina é semelhante a INLIN; é utilizada pelo comando INPUT para imprimir um ponto de interrogação e depois aceitar uma linha de entrada do teclado.

PARÂMETROS DE ENTRADA

Nenhum parâmetro de entrada é requerido.

PARÂMETROS DE SAÍDA

O registro HL aponta para a posição de memória abaixo do primeiro caractere do buffer e o flag de carry é setado se <CTRL><STOP> foi pressionado.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

Os registros AF, BC, DE e HL poderão ser alterados por esta chamada.

BREAKX 00B7

Executado por meio de CALL &H00B7.

Esta chamada verifica se <CTRL><STOP> foi pressionado.

PARÂMETROS DE ENTRADA

Nenhum parâmetro de entrada é requerido.

PARÂMETROS DE SAÍDA

Caso <CTRL><STOP> seja pressionado, o flag de carry será setado; caso contrário, será resetado.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

O acumulador e flags são alterados por esta chamada.

Nota: Esta rotina precisa ser utilizada apenas quando as interrupções são desativadas; caso contrário <CTRL><STOP>será descoberto pela interrupção de 60Hz.

ISCNTC 00BA

Executado por meio de CALL &H00BA.

Esta chamada verifica se a tecla <STOP> ou <CTRL> <STOP> é operada. Caso a tecla <STOP> estiver sendo pressionada, o programa se repete até que seja pressionada novamente ou que <CTRL><STOP> seja pressionado. Caso <CTRL><STOP> seja pressionado e não tenha sido pego, o computador passa pelas

suas seqüências de interrupção, isto é, passa ao próximo modo, ativando o slot que contém o interpretador BASIC e pulando até o modo direto BASIC. Se nem <STOP> ou <CTRL><STOP> são pressionados, esta rotina nada faz.

PARÂMETROS DE ENTRADA

Parâmetros de entrada não são requeridos, mas INTFLAG conterá 3 caso <CTRL><STOP> estiver sendo pressionado, e 4 se <STOP> estiver sendo pressionado, devido à interrupção do temporizador de 60Hz.

PARÂMETROS DE SAÍDA

Nenhum parâmetro será devolvido.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

O acumulador, flags e posição de memória INTFLG e KILBUF poderão ser afetados por esta chamada.

Nota: A primeira coisa que esta rotina faz é inspecionar a posição BASROM. Se esta é zero a rotina continua como anteriormente; caso contrário, retorna imediatamente. Portanto, colocar um número que não seja zero em BASROM impedirá que os programas BASIC sejam interrompidos (utilize com cuidado!).

As interrupções devem ser ativadas se esta rotina for utilizada.

CKCNTC 00BD

Executado por meio de CALL &H00BD.

Esta rotina faz a mesma coisa que ISCNTC, mas é ligeiramente mais lenta. Por este motivo é melhor utilizar ISCNTC.

PARÂMETROS DE ENTRADA

Não são necessários parâmetros de entrada.

PARÂMETROS DE SAÍDA

Nenhum parâmetro é devolvido.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

O acumulador, flags e a posição de memória INTFLG e KILBUF podem ser afetados por esta chamada.

BEEP 00C0

Executado por intermédio de CALL &H00C0.

Esta chamada emite um BEEP como se <CTRL><G> tivesse sido pressionado.

PARÂMETROS DE ENTRADA

Nenhum parâmetro de entrada é necessário.

PARÂMETROS DE SAÍDA

Nenhum parâmetro de saída é devolvido.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

Os registros AF, BC, DE, HL e as posições de memória MUSICF,PLYCNT VCBA até VCBA+4, VCBB até VCBB+4 e VCBC até VCBC+4 poderão ser alterados por esta rotina.

Nota: No fim desta rotina, um "jump" será feito ao ponto de entrada GICINI do BIOS, repondo o gerador de som.

CLS 00C3

Executado por meio de CALL &H00C3.

Esta chamada limpa o modo de tela (inclusive os modos gráficos).

PARÂMETROS DE ENTRADA

Caso o flag Z esteja ativado, o modo de tela será apagado; caso contrário, esta chamada nada fará.

PARÂMETROS DE SAÍDA

Nenhum parâmetro será devolvido.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

Os registros AF, BC, DE e as posições de memória LINTTB, CSRX e CSRY serão alterados por esta chamada.

POSIT 00C6

Executado por meio de CALL &H00C6

Esta rotina move o cursor de texto para uma determinada posição.

PARÂMETROS DE ENTRADA

A coluna da posição desejada deve ser colocada em H e a linha em L.

PARÂMETROS DE SAÍDA

CSRX e CSRY serão atualizados por esta chamada.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

O registro AF e posições de memória CSRX e CSRY serão alterados por esta chamada.

FNKSB 00C9

Executado por meio de CALL &H00C9.

Esta chamada verifica se o display da tecla de função está ativo e imprime as definições da tecla de função se estiver.

PARÂMETROS DE ENTRADA

Caso CNSDFG contiver zero, esta rotina não fará nada; caso contrário, acabará caindo em DSPFNK.

PARÂMETROS DE SAÍDA

Nenhum parâmetro é devolvido.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

Os registros AF, BC e DE são afetados por esta chamada.

Nota: As interrupções são ativadas por esta chamada.

ERAFNK 00CC

Executado por meio de CALL &H00CC.

Esta chamada cancela o display da tecla de função, desde que o processador de display do vídeo esteja em um dos modos texto.

PARÂMETROS DE ENTRADA

Nenhum parâmetro de entrada será necessário.

PARÂMETROS DE SAÍDA

Nenhum parâmetro de saída é devolvido.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

Os registros AF, BC e DE serão alterados por esta chamada.

DSPFNK 00CF

Executado por meio de CALL &H00CF.

Esta chamada apresenta as definições das teclas de funções na parte inferior do modo de tela.

PARÂMETROS DE ENTRADA

Nenhum parâmetro de entrada será necessário.

PARÂMETROS DE SAÍDA

A posição de memória CNSDFG é colocada em 255, indicando que o display da tecla de função está ativo.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

Os registros AF, BC, DE e a posição de memória CNSDFG serão alterados por esta chamada.

Nota: As interrupções são ativadas com esta chamada.

TOTEXT 00D2

Executado por meio de CALL &H00D2.

Esta rotina obriga o modo de tela para um dos modos texto, desde que não esteja em um.

PARÂMETROS DE ENTRADA

O modo texto que será colocado é armazenado na posição de memória OLDSCR.

PARÂMETROS DE SAÍDA

Nenhum parâmetro é devolvido.

REGISTRO E POSIÇÕES DE MEMÓRIA ALTERADOS

Esta chamada altera os registros AF, BC, DE e HL e as posições de memória LINLEN, NAMBAS, CGPBAS e SCRMOD.

Nota: Desde que o VDP já não esteja em um modo texto, esta rotina chama o hook H-TOTE com os conteúdos de OLDSCR no acumulador. Depois de voltar do hook o controle é passado ao ponto de entrada CHGMOD do BIOS.

As interrupções são ativadas durante esta rotina.

CHGCAP 0132

Executado por meio e CALL &H0132.

Esta chamada controla a situação da lâmpada do <CAPS LOCK>.

Observação: Nos MSX brasileiros não existe esta lâmpada.

PARÂMETROS DE ENTRADA

Caso o acumulador contiver zero, a lâmpada de <CAPS LOCK> será desligada; caso contrário, será ligada.

PARÂMETROS DE SAÍDA

Nenhum parâmetro é devolvido desta rotina.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

Apenas o acumulador e flags são alterados por esta chamada.

SNSMAT 0141

Executado por meio de CALL &H0141.

Esta chamada varre uma linha da matriz do teclado e devolve o estado das teclas daquela linha.

PARÂMETROS DE ENTRADA

O número de linha a ser varrida deve ser passado ao acumulador.

PARÂMETROS DE SAÍDA

A situação das teclas daquela linha é devolvida ao acumulador. Caso uma das teclas estiver sendo operada, o bit correspondente conterá zero; caso contrário, será 1.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

Apenas o acumulador e flags serão alterados por esta chamada.

Nota: As interrupções serão ativadas por esta chamada.

ISFLIO 014A

Executado por meio de CALL &H014A.

Esta chamada verifica se qualquer dispositivo E/S está sendo atendido.

PARÂMETROS DE ENTRADA

Nenhum parâmetro de entrada é necessário.

PARÂMETROS DE SAÍDA

Caso nenhum dispositivo de E/S estiver sendo atendido, o acumulador será zero e o flag Z será setado; caso contrário, o acumulador será diferente de zero e o flag Z será resetado.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

Apenas o acumulador e os flags são afetados por esta chamada.

OUTDLP 014D

Executado por meio de CALL &H014D.

Esta chamada é utilizada pelo interpretador BASIC para escrever um caractere na impressora.

PARÂMETROS DE ENTRADA

O caractere a ser impresso deve ser colocado no acumulador.

PARÂMETROS DE SAÍDA

Nenhum parâmetro é devolvido desta chamada, mas se a operação é abortada, o controle passa para a seção de tratamento de erros do interpretador BASIC, e um "device I/O erro" será mencionado.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

Apenas os flags são afetados por esta chamada.

Nota: Os caracteres TAB são expandidos em espaços por esta rotina.

KILBUF 0156

Executado por meio de CALL &H0156.

Esta chamada limpa o buffer de teclado.

PARÂMETROS DE ENTRADA

Nenhum parâmetro de entrada é necessário.

PARÂMETROS DE SAÍDA

Nenhum parâmetro será devolvido.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

Apenas o registrador HL é alterado por esta chamada.

CAPÍTULO 56

PONTOS DE ENTRADA DA BIOS QUE CONTROLAM AS PORTAS DO JOYSTICK

Os pontos de entrada descritos nas páginas seguintes fornecem controle sobre as portas dos *joystick* e dos dispositivos que podem ser ligadas a elas.

GTSTCK 00D5

Executado por meio de CALL &H00D5.

Esta chamada devolve o status das teclas de cursor ou de um dos *joystick*.

PARÂMETROS DE ENTRADA

O acumulador deve conter o identificador do *joystick*. Este é zero para as teclas de cursor, 1 para *joystick* 1 e 2 para *joystick* 2.

PARÂMETRO DE SAÍDA

A direção do *joystick* selecionado (ou das teclas de cursor) é devolvida no acumulador. Estas posições são:

0 para um *joystick* centralizado.
1 para cima.
2 para cima e para direita.

3 para direita.
4 para baixo e para a direita.
5 para baixo.
6 para baixo e para esquerda.
7 para esquerda.
8 para cima e para esquerda.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

Os registros AF, BC, DE e HL serão alterados com esta chamada.

Nota: As interrupções são ativadas com esta chamada.

GTTRIG 00D8

Executado por meio de CALL &H00D8.

Esta chamada devolve o status atual da barra de espaço, ou um dos botões de disparo do *joystick*.

PARÂMETROS DE ENTRADA

O acumulador especifica qual é o gatilho que deve ser verificado, conforme indicado a seguir:

0 = → a barra de espaço.
1 = → gatilho 1A.
2 = → gatilho 2A.
3 = → gatilho 1B.
4 = → gatilho 2B.

PARÂMETROS DE SAÍDA

Caso o gatilho relevante esteja sendo pressionado, o acumulador devolve 255; caso contrário, devolve zero.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS.

Os registros AF, BC, DE e HL poderão ser alterados com esta chamada.

Nota: As interrupções são ativadas por esta chamada.

GTPAD 00DB

Executado por meio e CALL &H00DB.

Esta chamada devolve o status de um "touch PAD" ligado a uma das portas *joystick*.

PARÂMETROS DE ENTRADA

O acumulador deve conter um número na faixa de 0-7, dependendo da informação requerida. Para A na faixa de 0-3, será devolvida a informação a respeito do "PAD" ligado à porta 1 do *joystick*, e uma informação análoga é devolvida a respeito de um "PAD" ligado à porta 2, caso A esteja entre 4 e 7.

Para determinar apenas se o "PAD" está sendo pressionado, utilize 0 e 4. Para encontrar a coordenada x de um ponto que está sendo pressionado, utilize 1 e 5, e utilize A = 2 ou 6 para a coordenada y. Para determinar se um "PAD" está sendo pressionado passe 3 ou 7 ao acumulador.

PARÂMETROS DE SAÍDA

O parâmetro de saída é colocado no acumulador e depende do parâmetro de entrada da seguinte forma. No caso de 0 ou 4, 0 será devolvido se o "PAD" relevante estiver sendo pressionado e, em caso negativo, o acumulador conterá 255. O mesmo será verdade para o estado de um "PAD" para um parâmetro de entrada de 3 ou 7. A coordenada x ou y (na faixa 0-255) de um ponto pressionado, será devolvida se o parâmetro de entrada foi 1 ou 2, respectivamente (5 ou 6 para o *joystick* da porta 2).

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

Os registros AF, BC, DE e HL poderão ser afetados por esta chamada.

Nota: As interrupções são ativadas por esta chamada. Veja o comando PAD do BASIC para maiores informações.

GTPDL 00DE

Executado por meio de &H00DE.

Esta chamada devolve o estado de um dos doze "PAD" possíveis ligados às portas de *joystick*.

PARÂMETROS DE ENTRADA

O número do "PAD" deve ser passado ao acumulador. Este é um número ímpar para um "PAD" ligado à porta 1 de *joystick*, e para a porta 2 será par.

PARÂMETROS DE SAÍDA

Um número da faixa 0-255 é devolvido ao acumulador, especificando o estado do "PAD" solicitado.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

Os registros AF, BC, DE e HL podem ser alterados por esta chamada.

Nota: As interrupções são ativadas por esta chamada.

CAPÍTULO 57

CHAMADAS DA BIOS ASSOCIADAS AO CASSETE

As chamadas descritas neste capítulo são utilizadas para controlar a interface do cassete e o sistema de arquivo.

TAPION 00E1

Executado por meio de CALL &H00E1.

Esta rotina liga o motor do cassete e lê o cabeçalho da fita.

PARÂMETROS DE ENTRADA

Nenhum parâmetro de entrada é necessário.

PARÂMETROS DE SAÍDA

O flag de carry é setado se a operação é abortada.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

Esta rotina poderá alterar os registros AF, BC, DE e HL.

Nota: As interrupções são desativadas durante a leitura do cassete.

TAPIN 00E4

Executado por meio de CALL &H00E4.

Esta chamada lê um byte do cassete.

PARÂMETROS DE ENTRADA

Nenhum parâmetro de entrada será necessário.

PARÂMETROS DE SAÍDA

Os dados são colocados no acumulador. O flag de carry é setado se a operação foi abortada.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

Registros AF, BC, DE e HL poderão ser alterados com esta chamada.

Nota: As interrupções são desativadas durante a leitura do cassete.

TAPIOF 00E7

Executado por meio de CALL &H00E7.

Esta chamada interrompe a leitura da fita pelo computador.

PARÂMETROS DE ENTRADA

Nenhum parâmetro de entrada será necessário.

PARÂMETROS DE SAÍDA

Nenhum parâmetro será devolvido.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

Nenhum registro ou posição de memória é afetado.

TAPOON 00EA

Executado por meio de CALL &H00EA.

Esta chamada liga o motor do cassete e escreve um bloco de cabeçalho no cassete.

PARÂMETROS DE ENTRADA

O acumulador deve conter zero se um cabeçalho curto é desejado; caso contrário, um cabeçalho longo será utilizado.

PARÂMETROS DE SAÍDA

O flag de carry será setado se a operação for abortada.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

Os registros AF, BC, DE e HL poderão ser alterados por esta chamada.

Nota: As interrupções são desativadas por esta chamada.

TAPOUT 00ED

Executado por meio de CALL &H00ED.

Esta chamada escreve um byte no cassete.

PARÂMETROS DE ENTRADA

O acumulador deve conter o byte que vai ser escrito.

PARÂMETROS DE SAÍDA

Se a operação é abortada o flag de carry será setado no encerramento.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

Os registros AF, BC, DE e HL poderão ser alterados por esta chamada.

Nota: As interrupções são ativadas por esta chamada.

TAPOOF

Executado por meio de &H00F0.

Esta chamada interrompe a escrita no cassete pelo computador.

PARÂMETROS DE ENTRADA

Nenhum parâmetro de entrada será necessário.

PARÂMETROS DE SAÍDA

Nenhum parâmetro é devolvido.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

Nenhum registro ou posição de memória é afetado.

STMOTR 00F3

Executado por meio de CALL &H00F3.

Esta chamada liga, desliga ou altera para o estado oposto o motor do cassete.

PARÂMETROS DE ENTRADA

Caso o acumulador contenha 1, o motor do cassete será ligado; se for 0, será desligado; e se o acumulador contiver 255, o estado do motor do cassete será alterado, isto é, se estiver ligado, será desligado, e se estiver desligado, será ligado.

PARÂMETROS DE SAÍDA

Nenhum parâmetro será devolvido.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

Apenas o acumulador e flags são afetados por esta chamada.

CAPÍTULO 58

PONTOS DE ENTRADA DA BIOS RELACIONADOS AO SOM

Os pontos de entrada seguintes são utilizados para controlar o gerador de som programável (PSG).

GICINI 0090

Executado por meio de CALL &H0090.

Esta chamada inicializa o gerador de som programável esvaziando as filas ("queues") sonoras e desliga qualquer som que estiver sendo gerado neste momento.

PARÂMETROS DE ENTRADA

Nenhum parâmetro de entrada será necessário, mas as interrupções têm de ser desativadas antes de chamar esta rotina.

PARÂMETROS DE SAÍDA

Nenhum parâmetro será devolvido.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

O conteúdo de todos os registros é preservado, mas as posições de memória MUSICF, PLYCNT, VCBA até VCBA+4, VCBB até VCBB+4 e VCBC até VCBC+4, são colocadas em zero.

WRTPSG 0093

Executado por meio de &H0093.

Esta chamada escreve um valor em um dos registros do gerador de som programável.

PARÂMETROS DE ENTRADA

O acumulador deve conter o número de registro no qual se vai escrever, que deve estar na faixa 0-13. O dado que vai ser escrito deve ser colocado em E.

PARÂMETROS DE SAÍDA

Não há nenhum parâmetro de saída.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

Nenhum registro ou posição de memória é alterado.

Nota: As interrupções são desativadas no começo desta chamada, e ativadas no fim.

RDPSG 0096

Executado por meio de CALL &H0096.

Esta chamada lê o conteúdo de um dos registros do gerador de som programável.

PARÂMETROS DE ENTRADA

O acumulador deve conter o número do registro que vai ser lido, e que deve estar na faixa 0-13.

PARÂMETROS DE SAÍDA

O conteúdo do registro é devolvido no acumulador.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

Apenas o acumulador é afetado por esta chamada.

Nota: As interrupções são desativadas no começo desta chamada, e ativadas no fim.

STRTMS 0099

Executado por meio de CALL &H0099.

Esta rotina dá início à tarefa da música de fundo, caso haja.

PARÂMETROS DE ENTRADA

Nenhum parâmetro precisa ser passado para esta rotina.

PARÂMETROS DE SAÍDA

No término, o acumulador conterá zero caso o buffer sonoro esteja vazio.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

Os registros AF, HL e posições de memória PLYCNT e MUSICF poderão ser alterados por esta chamada.

Nota: A tarefa do fundo musical não iniciará até a próxima interrupção de 60 Hz.

CAPÍTULO 59

PONTOS DE ENTRADA DA BIOS ASSOCIADOS AO VDP

Os pontos de entrada da BIOS descritos neste capítulo fornecem o controle completo sobre o processador de vídeo do computador MSX.

DISSCR 0041

Executado por meio de CALL &H0041.

Desativa o modo de tela. Esta chamada fará com que o modo de tela tome a cor da borda. Toda saída será enviada a tela mas não será visível até que ENASCR (&H0044) seja chamado. Alterar o modo também reativará o modo de tela.

PARÂMETROS DE ENTRADA

Nenhum.

PARÂMETROS DE SAÍDA

Nenhum.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

Os registros AF e BC são modificados por esta chamada, e as interrupções são ativadas.

UTILIZAÇÕES POSSÍVEIS

Esta chamada tem utilidade no BASIC, via a função USR, para embranquecer o modo de tela. Poderia ser utilizada para estabelecer um quadro com o modo de tela desativado; depois, ativando-o (ENASCR) teríamos a impressão de uma plotagem instantânea. Exemplo:

```
10 DEF USR0 = &H41
20 DEF USR1 =&H44
30 CLS
40 REM DESATIVA O MODO DE TELA
50 X=USR0 (0)
60 INPUT PW$
70 REM ATIVA O MODO DE TELA
80 X=USR1 (0)
```

ENASCR 0044

Executado por meio de CALL &H0044.

Este endereço é chamado para ativar a tela, depois que tiver sido desativada por DISSCR.

PARÂMETROS DE ENTRADA

Nenhum parâmetro de entrada será necessário.

PARÂMETROS DE SAÍDA

Nenhuma saída é produzida.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

Os registros AF e BC são alterados por esta rotina, e as interrupções são ativadas.

UTILIZAÇÕES POSSÍVEIS

Ativar o modo de tela depois de DISSCR. Veja DISSCR para ter um exemplo de uso.

WRTVDP 0047

Executado por meio de &H0047.

Esta chamada escreve em um registro do processador de display de vídeo. Para detalhes sobre os efeitos de escrever no VDP veja Parte 2, Capítulo 44.

PARÂMETROS DE ENTRADA

O número de registro do VDP para ser acessado deve ser especificado no registro C e o dado a ser escrito em B.

PARÂMETROS DE SAÍDA

Nenhum parâmetro de saída é devolvido, mas a posição de memória RGOSAV+C contém o valor inicialmente especificado em B.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

Os registros AF e BC, e a posição de memória RGxSAV são alterados por esta chamada, onde x é o registro em que se escreve. Por exemplo, se C contiver 5 e B contiver 0, então, ao sair da rotina a posição de memória RG5SAV conterá 0. Também as interrupções são ativadas com esta chamada.

UTILIZAÇÕES POSSÍVEIS

Este é um ponto de entrada muito poderoso, permitindo um controle sobre o VDP. É recomendável que esta rotina seja utilizada para escrever nos registros VDP, uma

vez que cópias deles sejam automaticamente mantidos. Quando um valor é colocado em um registro, não há meio de dizer qual é o valor, uma vez que estes registros são "apenas de escrita".

RDVDP 013E

Executado por meio de CALL &H013E.

Esta rotina é utilizada para ler o registrador de status do processador de display de vídeo.

PARÂMETROS DE ENTRADA

Nenhum parâmetro de entrada será necessário.

PARÂMETROS DE SAÍDA

O acumulador contém uma cópia do registrador de status do VDP no encerramento.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

Apenas o acumulador é afetado por esta chamada.

Nota: O estado das interrupções não é afetado por esta chamada.

RDVRM 004A

Executado por meio de &H004A.

Esta chamada lê uma posição de memória na RAM de vídeo, controlado pelo processador de vídeo.

PARÂMETROS DE ENTRADA

O par de registros HL deve conter o endereço da RAM de vídeo (VRAM) a ser acessado.

PARÂMETROS DE SAÍDA

Ao voltar, A contém o conteúdo da memória de vídeo apontado por HL. E ainda, as interrupções são ativadas por esta rotina. Nota: A situação dos flags ao deixar esta rotina não reflete o conteúdo do acumulador.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

Apenas A e os flags são alterados por esta rotina.

WRTVRM 0004D

Executado por meio de &H004D.

Esta chamada escreve o conteúdo do acumulador para a posição na RAM de vídeo.

PARÂMETROS DE ENTRADA

O par de registros HL deve conter o endereço na RAM de vídeo em que se vai escrever, e A o valor a ser escrito.

PARÂMETROS DE SAÍDA

Nenhum.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

O acumulador e flags são alterados por esta rotina e as interrupções ativadas.

UTILIZAÇÕES POSSÍVEIS

Acessar a RAM de vídeo.

SETRD 0050

Executado por meio de &H0050.

Esta chamada estabelece o processador do display de vídeo para uma operação de leitura.

PARÂMETROS DE ENTRADA

HL deve conter o endereço a ser lido.

PARÂMETROS DE SAÍDA

Nenhum.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

O registro A e flags são afetados, e as interrupções são ativadas por esta chamada.

UTILIZAÇÕES POSSÍVEIS

Antes de ler a VRAM, o VDP tem de ser inicializado para uma operação de leitura. Esta rotina é chamada de RDVRM e LDIRMV, de modo que não é necessário utilizar este ponto de entrada a não ser que uma leitura feita diretamente do VDP deva ser executada.

SETWRT 0053

Executado por meio de CALL &H0053.

Esta chamada estabelece o processador de display de vídeo para um operação de escrita.

PARÂMETROS DE ENTRADA

O par de registros HL deve conter o endereço da RAM VDP em que se vai escrever.

PARÂMETROS DE SAÍDA

Nenhum.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

O acumulador e os flags são afetados, e as interrupções ativadas por esta rotina.

UTILIZAÇÕES POSSÍVEIS

Da mesma forma que com SETRD, esta rotina não precisará nunca ser utilizada a não ser que o usuário queira acessar VDP diretamente, e uma vez que é chamada por sub-rotinas WRTVRM, FILVRM e LDIRVM.

FILVRM 0056

Executado por meio de &H0056.

Esta chamada preenche uma seção da RAM de vídeo, controlada pelo processador de display de vídeo, com um único valor.

PARÂMETROS DE ENTRADA

O endereço do primeiro byte no bloco a ser preenchido deve ser fornecido em HL, o comprimento do bloco em BC e o valor a ser escrito em A.

PARÂMETROS DE SAÍDA

Nenhum parâmetro é devolvido por esta rotina.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

AF e BC são alterados e as interrupções ativadas por esta chamada.

UTILIZAÇÕES POSSÍVEIS

É útil para preencher rapidamente uma parte do modo de tela com uma única cor. É utilizado pelo PAINT do BASIC.

LDIRMV 0059

Executado por meio de CALL &H0059.

Esta chamada move um bloco de memória da VRAM para a RAM principal.

PARÂMETROS DE ENTRADA

O endereço da VRAM deve ser especificado em HL, o comprimento do bloco em BC e o endereço de destino na RAM principal em DE.

PARÂMETROS DE SAÍDA

Um bloco de memória conforme especificado acima por DE e BC é devolvido por esta rotina.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

AF, BC, DE e o bloco de destino são afetados e as interrupções ativadas.

UTILIZAÇÕES POSSÍVEIS

Esta chamada é útil se uma quantidade de processamento é requerido no modo de tela, uma vez que é muito mais rápido carregar o modo de tela na memória, processá-lo e depois colocá-lo de volta na RAM de vídeo utilizando LDIRVM. Outra utilização é para gravar displays do modo de tela copiando-o na memória principal, e depois gravar os dados. O display pode ser restaurado utilizando LDIRVM.

Nota: Não confunda esta rotina com LDIRVM: LDIRMV move um bloco da RAM de vídeo.

LDIRVM 005C

Executado por meio de &H005C.

Esta rotina copia um bloco de memória na RAM controlada pelo processador de vídeo (VRAM), a partir do RAM de Z80.

PARÂMETROS DE ENTRADA

O endereço do bloco deve ser especificado em HL, o destino na RAM de vídeo em DE, e o comprimento, em BC.

PARÂMETROS DE SAÍDA

Nenhum parâmetro é devolvido por esta rotina.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

AF, BC e DE são alterados, e as interrupções ativadas por esta chamada.

Nota: Não confunda esta chamada com LDIRMV (LDIRVM move um bloco de memória para a RAM de vídeo).

CHGMOD 005F

Executado por meio de CALL &H005F.

Esta chamada coloca o VDP em um de seus quatro modos de operação, isto é, modo texto, modo gráfico I (32 colunas), modo gráfico II ou modo policromático.

PARÂMETROS DE ENTRADA

O acumulador deverá conter o modo requerido, conforme segue:

A=0 Modo texto (40 colunas).
A=1 Modo Gráfico I (32 colunas).
A=2 Modo Gráfico II (Alta resolução).
A=3 Modo Policromático.

PARÂMETROS DE SAÍDA

Ao encerrar, SCRMOD contém o número do novo modo.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

Os registros alterados são AF, BC, DE e HL. Posições de memória possivelmente afetadas por esta chamada são LINLEN, NAMBAS, CGPBAS, SCRMOD e OLDSCR.

Nota: Esta sub-rotina chama um dos INITXT, INIT32, INIGP ou INIMULT, dependendo do valor contido no acumulador. Ao encerrar, as interrupções são ativadas.

CHGCLR 0062

Executado por meio de &H0062.

Esta chamada altera a cor do texto, do fundo e da borda, caso o VDP esteja no modo texto (40 ou 32 colunas); ou altera a cor da borda caso um modo gráfico esteja em operação.

PARÂMETROS DE ENTRADA

A cor do texto é pega do FORCLR.

A cor do fundo é pega do BAKCLR.

A cor da borda é pega do BDRCLR.

PARÂMETROS DE SAÍDA

Não há parâmetros de saída.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

AF, BC e HL são afetados por esta chamada.

CLRSPR 0069

Executado por meio de CALL &H0069.

Esta chamada inicializa todos os sprites. O modelo é estabelecido em transparente (zero), as cores dos sprites são colocadas na cor atual do texto, as posições verticais são colocadas em 20 (fora do modo de tela); os nomes dos sprites são colocados no número de plano do sprite (isto é, o nome do sprite no plano 0 é colocado no código 0 do ASCII etc.).

PARÂMETROS DE ENTRADA

Nenhum parâmetro de entrada é requerido.

PARÂMETROS DE SAÍDA

Nenhum parâmetro é devolvido por esta chamada.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

Os registros AF, BC, DE e HL são afetados pela chamada.

Nota: Esta rotina não é chamada ao alterar o modo.

INITXT 006C

Executado por meio de CALL &H006C.

Este ponto de entrada inicializa as posições de memória que descrevem o modo de tela, para modo texto, depois chama SETTXT.

PARÂMETROS DE ENTRADA

Posições de memória TXTNAM, TXTCGP e LINL40 contêm os parâmetros de entrada.

PARÂMETROS DE SAÍDA

Ao terminar, SCRMOD = 0, OLDSCR = 0, NAMBAS = TXTNAM, CGPBAS = TXTCGP e LINLEN = LINL40.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

Os registros AF, BC, DE, HL, posições de memória RG0SAV, RG1SAV, RG2SAV, RG3SAV, RG4SAV, RG5SAV, RG6SAV e as posições de memória listadas na seção de parâmetros de saída poderão ser afetados por esta chamada.

Nota: As interrupções são ativadas por esta chamada.

INIT 32 006F

Executado por meio de CALL &H006F.

Este ponto de entrada inicializa as posições de memória que descrevem o modo de tela para o modo gráfico I (texto de 32 colunas), depois chama SETT32.

PARÂMETROS DE ENTRADA

As posições de memória T32NAM, T32CGP, T32COL, T32ATR, T32PAT e LINL32 contém os parâmetros de entrada para esta rotina.

PARÂMETROS DE SAÍDA

Ao terminar, SCRMOD = 1, OLDSCR = 1, NAMBAS = T32 NAM, CGPBAS = CGPBAS, PATBAS = T32PAT, ATRBAS = T32ATR e LINLEN = LINL32.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

Os registros AF, BC, DE, HL, posições de memória RG0SAV, RG1SAV, RG2SAV, RG3SAV, RG4SAV, RG5SAV, RG6SAV e as posições de memória relacionadas na seção de parâmetros de saída poderão ser afetadas por esta chamada.

Nota: As interrupções são ativadas por esta chamada.

INIGRP 0072

Executado por meio de CALL &H0072.

Este ponto de entrada inicializa as posições de memória para o modo gráfico II (modo de alta resolução), depois chama SETGRP.

PARÂMETROS DE ENTRADA

As posições de memória GRPNAM, GRPCGP, GRPCOL, GRPATR e GRPPAT contêm os parâmetros de entrada utilizados nesta chamada.

PARÂMETROS DE SAÍDA

Ao terminar, SCRMOD = 2, PATBAS = GRPPAT e ATRBAS = GRPATR.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

Os registros AF, BC, DE, HL e posições de memória RG0SAV, RG1SAV, RG2SAV, RG3SAV, RG4SAV, RG5SAV, RG6SAV e as posições de memória relacionadas na seção de parâmetros de saída poderão ser afetados por esta chamada.

Nota: As interrupções são ativadas por esta chamada.

INIMLT 0075

Executado por meio de CALL &H0075.

Esta chamada inicializa as diversas posições de memória para o modo policromático, depois chama SETMLT.

PARÂMETROS DE ENTRADA

As posições de memória MLTNAM, MLTCGP, MLTCOL, MLTATR e MLTPAT contêm os parâmetros de entrada.

PARÂMETROS DE SAÍDA

Ao encerrar, SCRMOD = 3, PATBAS = MLTPAT e ATRBAS = MLTATR.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

Os registros AF, BC, DE, HL, posições de memória RG0SAV, RG1SAV, RG2SAV, RG3SAV, RG4SAV, RG5SAV, RG6SAV e as posições de memória relacionadas na seção de parâmetros de saída poderão ser afetados por esta chamada.

Nota: As interrupções são ativadas por esta chamada.

SETTXT 0078

Executado por meio de CALL &H0078.

Esta chamada prepara os registros do processador de display de vídeo para o modo texto (40 colunas).

PARÂMETROS DE ENTRADA

Nenhum parâmetro de entrada será requerido.

PARÂMETROS DE SAÍDA

Não há nenhum parâmetro de saída.

PARÂMETROS E POSIÇÕES DE MEMÓRIA ALTERADOS

Os registros AF, BC, DE, HL e posições de memória RG0SAV, RG1SAV, RG2SAV, RG3SAV, RG4SAV, RG5SAV, RG6SAV são afetados por esta chamada.

Nota: Chamar esta rotina não é suficiente para mudar o modo; ela estabelece os registros VDP mas não inicializa a RAM de vídeo. As interrupções são ativadas por esta rotina.

SETT32 007B

Esta chamada é executada por meio de CALL &H007B.

Esta chamada prepara os registros do processador de display de vídeo para o modo gráfico I (texto de 32 colunas).

PARÂMETROS DE ENTRADA

Nenhum parâmetro de entrada é necessário.

PARÂMETROS DE SAÍDA

Não há nenhum parâmetro de saída.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

Os registros AF, BC, DE, HL e posições de memória RG0SAV, RG1SAV, RG2SAV, RG3SAV, RG4SAV, RG5SAV, RG6SAV são afetados por esta chamada.

Nota: Chamar esta rotina não basta para alterar o modo; ela prepara os registros do VDP mas não inicializa a RAM de vídeo. As interrupções são ativadas durante a execução desta rotina.

SETGRP 007E

Executado por meio de CALL &H007E.

Esta chamada estabelece os registros para o processador de display de vídeo para o modo gráfico II (modo de alta resolução).

PARÂMETROS DE ENTRADA

Nenhum parâmetro de entrada é necessário.

PARÂMETROS DE SAÍDA

Não há nenhum parâmetro de saída.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

Os registros AF, BC, DE, HL e posições de memória RG0SAV, RG1SAV, RG2SAV, RG3SAV, RG4SAV, RG5SAV, RG6SAV são afetados por esta chamada.

Nota: Chamar esta rotina não é suficiente para alterar o modo; ela prepara os registros do PDV mas não inicializa a RAM de vídeo; as interrupções são ativadas durante esta rotina.

SETMLT 0081

Executado por meio de CALL &H0081.

Esta chamada prepara os registros do processador de display de vídeo para o modo policromático.

PARÂMETROS DE ENTRADA

Nenhum parâmetro de entrada é necessário.

PARÂMETROS DE SAÍDA

Não há parâmetros de saída.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

Os registros AF, BC, DE, HL e posições de memória RG0SAV, RG1SAV, RG2SAV, RG3SAV, RG4SAV, RG5SAV, RG6SAV são afetados por esta chamada.

Nota: Chamar esta rotina não é suficiente para alterar o modo; ela prepara os registros do VDP mas não inicializa a RAM de vídeo. As interrupções são ativadas durante a execução desta rotina.

CALPAT 0084

Executado por meio de CALL &H0084.

Esta chamada devolve o endereço de um padrão de sprite na RAM de vídeo.

PARÂMETROS DE ENTRADA

O número de sprite (0-31) deveria ser colocado no acumulador.

PARÂMETROS DE SAÍDA

O endereço do sprite na RAM de vídeo é devolvido no par de registros HL.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

Os registros AF, DE e HL são afetados por esta chamada.

Nota: O estado de interrupção não é alterado por esta chamada.

CALATR 0087

Executado por meio de CALL &H0087.

Esta rotina devolve o endereço na RAM de vídeo da tabela de atributo de um sprite.

PARÂMETROS DE ENTRADA

O número do sprite deve ser especificado no acumulador.

PARÂMETROS DE SAÍDA

Ao encerrar, o par de registros HL contém o endereço na RAM de vídeo da tabela de atributos do sprite desejado.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

DE, HL e os flags são alterados por esta rotina.

Nota: As interrupções não são afetadas.

GSPSIZ 008A

Executado por meio de CALL &H008A.

Esta chamada devolve o tamanho atual do sprite em termos de número de bytes ocupados por cada sprite (8 ou 32).

PARÂMETROS DE ENTRADA

Nenhum parâmetro de entrada é requerido.

PARÂMETROS DE SAÍDA

O número de bytes por sprite é devolvido no acumulador.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

O acumulador e flags são afetados por esta rotina.

Nota: Caso um sprite de 16x16 estiver em uso, o flag de carry será setado na saída; caso contrário, é resetado. As interrupções não são afetadas por esta chamada.

GRPPRT 008D

Executado por meio de CALL &H008D.

Esta chamada é utilizada para imprimir um caractere no modo gráfico de tela (modo de alta resolução) na posição atual do cursor gráfico.

PARÂMETROS DE ENTRADA

O código de caractere a ser impresso deve ser colocado no acumulador.

PARÂMETROS DE SAÍDA

Nenhum parâmetro é devolvido desta rotina.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

Nenhum registro ou posição de memória é afetado.

Nota: Esta rotina funciona apenas no modo gráfico II.

SCALXY 010E

Executado por meio de &H010E.

Esta chamada garante que um ponto, passado pelos registros, fique na tela. Em caso negativo, as coordenadas são "cortadas" e um erro será indicado. Cortar significa que x ou y são muito grandes; o valor máximo permitido é atribuido a eles e, em caso negativo, são colocados em zero. Ainda, se estiver no modo policromático, x e y são divididos por 4, uma vez que este modo tem 64 células horizontais e 48 verticais.

PARÂMETROS DE ENTRADA

BC deve conter a coordenada x, e DE a coordenada y.

PARÂMETROS DE SAÍDA

Ao terminar, BC e DE contêm as coordenadas "cortadas" de x e y respectivamente. Caso x ou y seja encontrado fora da faixa, o flag de carry será resetado; caso contrário, será setado.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

AF, BC e DE poderão ser alterados por esta rotina.

Nota: Pelo fato de que x e y estão em escala se o modo multicolorido estiver ativo (não os coloque você também em escala!).

As faixas de x e y, no modo gráfico II ou modo policromático, são 0-255 e 0-191, respectivamente.

MAPXYC 0111

Executado por meio de &H0111.

Esta rotina muito útil calcula o endereço na RAM de vídeo de um ponto no modo gráfico II ou no modo policromático. Devolve também a posição naquele byte dos bits representando o ponto.

PARÂMETROS DE ENTRADA

A posição (x,y) do ponto de interesse deve ser colocada em BC(x) e DE(y). Para o modo gráfico II, x tem de estar na faixa de 0-255, e y, 0-191. No modo policromático, x deve estar na faixa 0-63, e y na faixa 0-47. Esta rotina utiliza também SCRMOD para determinar o modo de tela, GRPCGP para encontrar o endereço inicial da tabela do gerador de padrões no modo gráfico II, e MLTCGP para encontrar o endereço inicial da tabela do gerador de padrões no modo policromático.

PARÂMETROS DE SAÍDA

O endereço na RAM de vídeo, do byte contendo o ponto, é devolvido em CLOC e o byte indicando os bits relevantes descrevendo aquele ponto é representado por um bit (texto = 1, fundo = 0). Este bit está na mesma posição que o bit de preparação em CMASK. Por exemplo, se o ponto for armazenado no bit 5 do byte apontado pelo CLOC, então CMASK conterá 00100000 em binário. Para o modo policromático, cada ponto é representado por quatro bits – o cor do ponto, e novamente a posição daquele quatro bits correspondem à posição do bit de preparação em CMASK. Por exemplo, se o ponto é

armazenado nos bits 0-3 do byte apontado pelo CLOC, então CMASK conterá 00001111 em binário.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

Os registros AF, D, HL e as posições de memória CLOC e CMASK são afetados por esta chamada.

Nota: Uma verificação de que o ponto realmente está no modo de tela não é executada por esta chamada. Para fazer esta verificação e aplicar uma escala para os valores de x e y no modo policromático, a sub-rotina SCLXY deve ser utilizada.

FETCHC 0114

Executado por meio de &H0114.

Esta rotina apenas lê o endereço do ponto atual e o padrão de máscara.

PARÂMETROS DE ENTRADA

CLOC deveria conter o endereço do ponto atual, e CMASK o padrão da máscara.

PARÂMETROS DE SAÍDA

Ao terminar, HL contém o contéudo de CLOC e A contém o valor armazenado em CMASK.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

HL e AF são os únicos registros alterados por esta chamada.

Nota: Esta chamada pode ser substituída por dois comandos Z80, isto é:

```
LD    A,(CMASK)
LD    HL, (CLOC)
```

STOREC 0117

Executado por meio de &H0117.

Esta rotina simplesmente armazena o endereço do ponto atual e o padrão de máscara na memória.

PARÂMETROS DE ENTRADA

HL deveria conter o endereço do ponto atual e A, o padrão da máscara.

PARÂMETROS DE SAÍDA

Ao terminar, CMASK conterá uma cópia do acumulador, e CLOC contém o valor contido em HL.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

As posições de memória CMASK e CLOC são afetadas por esta chamada.

Nota: Esta chamada pode ser substituída por dois comandos Z80, isto é.

```
LD    (CMASK),A
LD    (CLOC),HL
```

SETATR 011A

Executado por meio de &H011A.

Esta chamada estabelece o byte de atributo com o conteúdo do acumulador se o número no acumulador é menor que 16; caso contrário, o byte de atributo é inalterado e um erro é indicado.

PARÂMETROS DE ENTRADA

O acumulador deve conter o valor ao qual o byte de atributo deve ser estabelecido.

PARÂMETROS DE SAÍDA

Ao terminar, ATRBYT (o byte de atributo) contém o valor especificado no acumulador, desde que seja menor que 16.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

ATRBYT poderá ser alterado por esta chamada.

Nota: O byte atributo representa a cor de um ponto. É utilizado pela rotina SETC para estabelecer um ponto a esta cor.

READC 011D

Executado por meio de &H011D.

Esta chamada lê o atributo (cor) do ponto atual.

PARÂMETROS DE ENTRADA

O endereço do ponto atual e padrão de máscara deve ser armazenado em CLOC e CMASK, respectivamente.

PARÂMETROS DE SAÍDA

Ao terminar, a cor do ponto atual é armazenada no acumulador.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

Apenas o acumulador e flags são afetados por esta chamada.

SETC 0120

Executado por meio de &H0120.

Esta chamada estabelece o ponto atual na cor especificada. Caso esteja no modo policromático, este será o único efeito no display. Entretanto, no modo gráfico II, caso a cor especificada não seja a mesma que a do texto ou a do fundo, então a cor de texto dos pontos no byte contendo o ponto atual é alterada para a cor especificada.

PARÂMETROS DE ENTRADA

O endereço do ponto atual e padrão de máscara devem estar contidos em CLOC e CMASK, respectivamente, e a cor que será dada a este ponto deve ser colocada em ATRBYT.

PARÂMETROS DE SAÍDA

Nenhum parâmetro é devolvido por esta rotina.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

Apenas o acumulador e flags são afetados por esta chamada.

Nota: Se todos os pontos de um determinado byte são colocados na mesma cor, esta cor se torna a nova cor de fundo isto é, o byte na RAM de vídeo apontado pelo CLOC será zero.

RIGHTC 00FC

Executado por meio de CALL &H00FC.

Esta rotina movimenta o cursor gráfico um ponto para a direita.

PARÂMETROS DE ENTRADA

O endereço do byte contendo o ponto atual é armazenado em CLOC e o padrão da máscara do ponto em CMASK.

PARÂMETROS DE SAÍDA

O conteúdo de CLOC e CMASK terão sido atualizados à nova posição do cursor gráfico.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

AF e as posições de memória CLOC e CMASK poderão ser alterados por esta chamada.

Nota: Movimentando o cursor gráfico de modo a sair pelo lado direito do modo de tela, fará com que ele apareça no lado esquerdo, um ponto para baixo. Nenhuma verificação é ainda feita para garantir que o cursor gráfico não saia completamente do modo de tela, pelo canto inferior direito.

No modo policromático, o cursor gráfico é movido um bloco de 4 x 4 pontos por vez.

LEFTC 00FF

Executado por meio de CALL &H00FF.

Esta rotina movimenta o cursor gráfico um ponto para a esquerda.

PARÂMETROS DE ENTRADA

O endereço do byte contendo o ponto atual é armazenado em CLOC e o padrão de máscara do ponto em CMASK.

PARÂMETROS DE SAÍDA

O conteúdo de CLOC e CMASK terá sido atualizado à nova posição do cursor gráfico.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

AF e as posições de memória CLOC e CMASK serão alterados por esta chamada.

Nota: Movimentando o cursor gráfico de modo a sair pelo lado esquerdo do modo de tela, fará com que ele apareça no lado direito, um ponto para cima. Ainda aqui, nenhuma verificação é feita para garantir que o cursor gráfico não saia completamente do modo de tela, pelo canto superior esquerdo.

No modo policromático, o cursor gráfico é movido um bloco de 4 x 4 pontos.

UPC 0102

Executado por meio de CALL &H0102.

Esta rotina movimenta o cursor gráfico um ponto para cima se não estiver no topo do modo de tela; caso contrário, a sua posição permanece inalterada.

PARÂMETROS DE ENTRADA

O endereço do byte contendo o ponto atual é armazenado em CLOC e o padrão da máscara do ponto em CMASK.

PARÂMETROS DE SAÍDA

O conteúdo de CLOC e CMASK deve ter sido atualizado à nova posição do cursor gráfico.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

AF e as posições de memória CLOC e CMASK poderão ser alterados com esta chamada.

Nota: No modo policromátrico, o cursor gráfico é movimentado um bloco de 4 x 4 pontos.

TUPC 0105

Executado por meio de CALL &H0105.

Esta rotina movimenta o cursor gráfico um ponto para cima se não estiver no topo do modo de tela; caso contrário, sua posição permanece inalterada e o flag de carry é setado.

PARÂMETROS DE ENTRADA

O endereço do byte contendo o ponto atual é armazenado em CLOC e o padrão da máscara do ponto em CMASK.

PARÂMETROS DE SAÍDA

O conteúdo de CLOC e CMASK serão atualizados à nova posição do cursor gráfico.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

AF e as posições de memória CLOC e CMASK poderão ser alterados por esta chamada.

Nota: Esta chamada é virtualmente igual à UPC: a única diferença é que o flag de carry será setado se o cursor gráfico já estiver no topo do modo de tela; caso contrário, é resetado.

No modo policromático, o cursor gráfico é movido em um bloco de 4 x 4 pontos.

DOWNC 0108

Executado por meio de CALL &H0108.

Esta rotina moverá o cursor gráfico um ponto para baixo se não estiver no limite inferior do modo de tela, caso contrário, a sua posição permanecerá inalterada.

PARÂMETROS DE ENTRADA

O endereço do byte contendo o ponto atual é armazenado em CLOC e o padrão de máscara do ponto em CMASK.

PARÂMETROS DE SAÍDA

O conteúdo de CLOC e CMASK terá sido atualizado à nova posição do cursor gráfico.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

AF e as posições de memória CLOC e CMASK poderão ser alterados por esta chamada.

Nota: No modo policromátrico, o cursor gráfico é movido um bloco de 4 x 4 pontos.

TDOWNC 010B

Executado por meio de &H010B.

Esta rotina movimenta o cursor gráfico um ponto para baixo desde que não esteja no limite inferior do modo de tela; caso contrário, sua posição permanece inalterada e o flag de carry é setado.

PARÂMETROS DE ENTRADA

O endereço do byte contendo o ponto atual é armazenado em CLOC e o padrão de máscara do ponto em CMASK.

PARÂMETROS DE SAÍDA

O conteúdo de CLOC e CMASK terá sido atualizado à nova posição do cursor gráfico.

REGISTROS E POSIÇÕES DE MEMÓRIA ATUALIZADOS

AF e as posições de memória CLOC e CMASK poderão ser alterados por esta chamada.

Nota: Esta chamada é idêntica à DOWNC: a única diferença é que o flag de carry será setado se o cursor gráfico já estiver na parte inferior do modo de tela; caso contrário, será resetado.

No modo policromático, o cursor gráfico é movido um bloco de 4 x 4 pontos.

NSETCX 0123

Executado por meio de CALL &H0123.

Esta chamada estabelece um determinado número de pontos, à direita do cursor gráfico atual, com uma determinada cor.

PARÂMETROS DE ENTRADA

HL deve conter o número de pontos que são estabelecidos, ATRBYT a cor dos pontos, e o endereço e padrão de máscara do cursor gráfico deve estar armazenado em CLOC e CMASK, respectivamente.

PARÂMETROS DE SAÍDA

Nenhum parâmetro será devolvido.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

AF, BC, DE, HL e as posições de memória CLOC e CMASK poderão ser afetados por esta chamada.

Nota: Se estiver no modo policromático, a posição do cursor gráfico ficará um ponto à esquerda do último estabelecido, mas no modo gráfico II esta posição permanece inalterada.

As interrupções são ativadas por esta rotina.

GTASPC 0126

Executado por meio de CALL &H0126.

Esta rotina é utilizada pelo comando CIRCLE do interpretador BASIC, para tomar os parâmetros da razão de raios atual.

PARÂMETROS DE ENTRADA

ASPECT1 deve conter 256/ (razão de aspecto) e ASPCT2 deveria conter 256* (razão de aspecto).

PARÂMETROS DE SAÍDA

Ao encerrar, HL conterá os conteúdos de ASPCT2 e DE, ASPECT1.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

Apenas HL e DE são afetados por esta chamada.

PNTINT 0129

Executado por meio de CALL &H0129.

Esta é uma rotina de inicialização para o comando PAINT do BASIC. A cor de contorno do espaço a ser preenchido é verificada, para que esteja menor que 16.

PARÂMETROS DE ENTRADA

O acumulador deve conter a cor do contorno que vai ser preenchido.

PARÂMETROS DE SAÍDA

Caso o modo policromático seja atualmente ativo, então BRDATR conterá uma cópia do parâmetro ativo; caso contrário, conterá uma cópia do valor armazenado em ATRBYT. Ainda, caso o parâmetro de entrada seja maior que 15, então no modo policromático o flag de carry será setado no retorno desta rotina.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

O acumulador, flags e posição de memória BRDATR serão afetados por esta rotina.

Nota: A cor de contorno é utilizada apenas na rotina PAINT do modo policromático. No modo gráfico II, a rotina PAINT procura pela cor de texto.

SCANR 012C

Executado por meio de CALL &H012C.

Esta chamada varre um determinado número de pontos à direita do cursor gráfico, colocando-os em uma determinada cor, até que um ponto de uma determinada "cor de contorno" seja encontrado, ou que o lado direito do modo de tela seja alcançado, ou que o máximo número de pontos seja varrido.

PARÂMETROS DE ENTRADA

O número máximo de pontos a ser varrido deve ser armazenado em DE (até 256), a cor de contorno deve ser armazenado em BRDATR, e a cor em que os pontos devem ser colocados em ATRBYT.

PARÂMETROS DE SAÍDA

Nenhum parâmetro é devolvido.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

AF, BC, DE, HL e as posições de memória CLOC, CMASK e FILNAM até FILNAM+3 serão afetados por esta chamada.

Nota: Esta rotina é utilizada pelo comando PAINT. FILNAM até FILNAM+3 é utilizado como espaço de trabalho.

As interrupções são ativadas nesta chamada.

SCANL 012F

Executado por meio de CALL &H012F.

Esta chamada varre um determinado número de pontos à esquerda do cursor gráfico, colocando-os em uma determinada cor, até que um ponto de uma determinada "cor de contorno" seja encontrado, ou que o lado esquerdo do modo seja alcançado, ou que o número máximo de pontos seja varrido.

PARÂMETROS DE ENTRADA

O número máximo de pontos a ser varrido deve ser armazenado em DE (até 256), a cor de contorno deve ser armazenado em BRDATR, e a cor em que os pontos devem ser colocados, em ATRBYT.

PARÂMETROS DE SAÍDA

Nenhum parâmetro é devolvido.

REGISTROS E POSIÇÕES DE MEMÓRIA ALTERADOS

AF, BC, DE, HL e posições de memória CLOC, CMASK e FILNAM até FILNAM+3 serão afetados por esta chamada.

Nota: Esta rotina é utilizada pelo comando PAINT. FILNAM até FILNAM+3 é utilizado como espaço de trabalho.

As interrupções são ativadas nesta chamada.

CAPÍTULO 60

O USO DE HOOKS

Os hooks fornecem uma maneira de interceptar o interpretador BASIC ou o sistema operacional em determinados pontos, permitindo que um processamento adicional, ou alternativo, seja executado. Vejamos um exemplo: considere a chamada CHPUT da BIOS (veja a Parte 2, Capítulo 31 (Vol. 1)). Esta rotina é utilizada para imprimir texto no modo de tela. Aqui está uma listagem dos primeiros poucos comandos da CHPUT:

```
PUSH   HL        ; GRAVA OS REGISTRADORES
PUSH   DE
PUSH   BC
PUSH   AF
CALL   &HFDA4  ; CHAMA O HOOK H.CHPU
```

Em primeiro lugar, os registros HL, BC, DE e AF são empurrados para a pilha. Depois o endereço de memória &HFDA4 é chamado. Normalmente, o conteúdo de &HFDA4 e os quatro bytes seguintes contêm um comando Z80 RET, passando imediatamente o controle ao comando que segue ao CALL. Entretanto, &HFDA4 é em RAM, de modo que o seu conteúdo pode ser alterado. Deixe que os 3 bytes, iniciando em &HFDA4, sejam alterados, de modo que o controle seja passado à outra sub-rotina em &HD000, isto é:

```
FDA4   C3   00   D0 = JP      &HD000
```

Faça também com que o código em &HD000 seja conforme indicado a seguir:

```
POP   HL      ; LIVRE-SE DO PRIMEIRO ITEM DA PILHA
                (isto é, VOLTE AO ENDEREÇO DO HOOK)
POP   AF      ; RESTAURE OS REGISTROS
POP   BC
POP   DE
POP   HL
RET           ;SAIA DA CHAMADA CHPUT DO BIOS
```

Agora o item do topo da pilha é 3 a mais que o endereço de onde &HFDA4 foi chamado, isto é, o endereço de retorno para CHPUT. Portanto, se este for removido da pilha, a pilha será restaurada ao estado em que se encontrava imediatamente antes de chamar &HFDA4. Ainda, se os próximos quatro itens forem retirados (POP) conforme mostrado acima, a pilha ficará como estava ao entrar com CHPUT, de modo que o comando RET saia da sub-rotina CHPUT. Conseqüentemente, o caractere que era para ser impresso não o será!

Este exemplo demonstra como o sistema operacional pode ser interceptado. Os cinco bytes, iniciando no endereço &HFDA4, são mostrados como um hook. Hooks semelhantes existem em muitos outros pontos do sistema operacional e no interpretador BASIC, e uma lista deles, em ordem alfabética, está relacionada a seguir:

NOME	ENDEREÇO	DETALHES
H-ATTR	FE1C	MSXSTS, no começo da rotina ATTR$ (atributo).
H-BAKU	FEAD	SPCDSK, na rotina BAKUPT (duplicação).
H-BINL	FE76	SPCDSK, na rotina BINLOD (carga binária).
H-BINS	FE71	SPCDSK, na rotina BINSAV (gravação binária).
H-BUFL	FF8E	BINTRP, na rotina BUFLIN (linha de buffer).
H-CHGE	FDC2	MSXIO, no início da rotina CHGET (pegar caractere).
H-CHPU	FDA4	MSXIO, no início da rotina CHPUT (colocar caractere).
H-CHRG	FF48	BINTRP.
H-CLEA	FEDO	BIMISC, na rotina CLEARC (liberar CLEAR).
H-CMD	FEOD	MSXSTS, no início da rotina CMD (comando).
H-COMP	FF57	BINTRP.
H-COPY	FE08	MSXSTS, no início da rotina COPY (copiar arquivos).
H-CRDO	FEE9	BIO, na rotina CRDO (CR1f DO).
H-CRUN	FF20	BINTRP.

NOME	ENDEREÇO	DETALHES
H-CRUS	FF25	BINTRP.
H-CVD	FE49	MSXSTS, no início da rotina CVD (converter db1).
H-CVI	FE3F	MSXSTS, no início da rotina CVI (converter int).
H-CVS	FE44	MSXSTS, no início da rotina CVS (converter sng).
H-DEGVN	FEC1	SPCDEV, na rotina DEVNAM (nome de dispositivo).
H-DGET	FE80	SPCDSK, na rotina DGET (pegar disco).
H-DIRD	FF11	BINTRP, na entrada DIRD (comando DO direto).
H-DOGR	FEF3	GENGRP, na rotina DOGRPH (fazer gráfico).
H-DSKC	FEEE	BIO, na rotina DSKCHI (entrada de caractere de disco).
H-DSKF	FE12	MSXSTS, no início da rotina DSKF (disco livre).
H-DSKI	FE17	MSXSTS, no início da rotina DSKI (entrada de disco).
H-DSKO	FDEF	MSXSTS, no início da rotina DSK0$ (saída de disco).
H-DSPC	FDA9	MSXIO, no início da rotina DSPCSR (cursor display).
H-DSPF	FDB3	MSXIO, no início da rotina DSPFNK (apresenta tecla de função).
H-EOF	FEA3	SPCDSK, na função EOF (fim de arquivo).
H-ERAC	FDAE	MSXIO, no início da rotina ERASCR (cancela cursor).
H-ERAF	FDB8	MSXIO, no início da rotina ERAFNK (cancelar a tecla de função).
H-ERRF	FF02	BINTRP.
H-ERRO	FFB1	BINTRP, na rotina ERROR.
H-ERRP	FEFD	BINTRP, na rotina ERRPRT (erro de impressão).
H-EVAL	FF70	BINTRP.
H-FIEL	FE2B	MSXSTS, no início da rotina FIELD (campo).
H-FILE	FE7B	SPCDSK, no comando FILES.
H-FILO	FE85	SPCDSK, na rotina FILOU1 (arquivo 1 saída).
H-FINE	FF1B	BINTRP.
H-FING	FF7A	BINTRP.
H-FINI	FF16	BINTRP.
H-FINP	FF5C	BINTRP.
H-FORM	FFAC	MSXIO, na rotina FORMAT (formatador de disco).
H-FPOS	FEA8	SPCDSK, na função FPOS (posição de arquivo).
H-FRET	FF9D	BISTRG, na rotina FRETM (livre para temporários).
H-FRME	FF66	BINTRP.
H-FRQI	FF93	BINTRP, na rotina FRQINT.
H-GEND	FEC6	SPCDEV, nO GENDSP (despachador geral de dispositivo).
H-GETP	FE4E	SPCDSK, na rotina GETPTR (pegar flag de arquivo).
H-GONE	FF43	BINTRP.
H-INDS	FE8A	SPCDSK, na rotina INDSKC (caractere de disco de entrada).
H-INIP	FDC7	MSXIO, no início da rotina INIPAT (inicializa padrão).
H-IBNLI	FDE5	MSXINL, no início da rotina LININ (linha de entrada).
H-IPL	FE03	MSXSTS, no início da rotina IPL (carregamento inicial do programa).
H-ISFL	FEDF	BIMISC, na rotina ISFLIO (é E/S arquivo).
H-ISMI	FF7F	BINTRP, na rotina ISMID$ (é MID$).
H-ISRE	FF2A	BINTRP.

NOME	ENDEREÇO	DETALHES
H-KEYC	FDCC	MSXIO, no início da rotina KEYCOD (codificador de tecla).
H-KEYI	FD9A	MSXIO, no início do manipulador de interrupção.
H-KILL	FDFE	MSXSTS, no início da rotina KILL (matar arquivo).
H-KYEA	FDD1	MSXIO, no início da rotina KYEASY (chave fácil).
H-LIST	FF89	BINTRP, na rotina LIST.
H-LOC	FE99	SPCDSK, na função LOC (local).
H-LOF	FE9E	SPCDSK, na fnção LOF (comprimento de arquivo).
H-LOPD	FED5	BIMISC, na rotina LOPDFT (loop e default colocado).
H-LPTO	FFB6	MSXIO, na rotina LPTOUT (saída impressora de linha).
H-LPTS	FFBB	MSXIO, na rotina LPTSTT (status impressora de linha).
H-LSET	FE21	MSCSTS, no início da rotina LSET (colocação à esquerda).
H-MAIN	FFOC	BINTRP, na entrada MAIN.
H-MERG	FE67	SPCDSK, na rotina MERGE (arquivos de programa de fusão).
H-MKD$	FE3A	MSXSTS, no início da rotina MKD$ (duplicar).
H-MKI$	FE30	MSXSTS, no início da rotina MKI$ (fazer inteiro).
H-MKS$	FE35	MSXSTS, no início da rotina MKS$ (fazer único).
H-NAME	FDF9	MSXSTS, no início da rotina NAME (renomear).
H-NEWS	FF3E	BINTRP.
H-NMI	FDD6	MSXIO, no início da rotina NMI (interrupção não mascarada).
H-NODE	FEB7	SPCDEV, na rotina NODEVN (nenhum nome dispositivo).
H-NOFO	FE58	SPCDSK, na rotina NOFOR (nenhuma cláusula para).
H-NOTR	FF34	BINTRP.
H-NTFL	FE62	SPCDSK, na rotina NTFLO (não arquivo 0).
H-NTFN	FF2F	BINTRP.
H-NTPL	FF6B	BINTRP.
H-NULO	FE5D	SPCDSK, na rotina NULOPN (nenhum arquivo aberto).
H-OKNO	FF75	BINTRP.
H-ONGO	FDEA	MSXSTS, no início do procedimeno ONGOTP (na rotina de procedimento GOTO).
H-OUTD	FEE4	BIO, na rotina OUTDO (OUT DO).
H-PARD	FEB2	SPCDEV, na rotina PARDEV (passe nome dispositivo).
H-PHYD	FFA7	MSXIO, na rotina PHYDIO (E/S disco físico).
H-PINL	FDDB	MSXINL, no início da rotina PINLIN (linha de programa).
H-PLAY	FFC5	MSXSTS, na entrada para a declaração PLAY.
H-POSD	FEBC	SPCDEV, na rotina POSDSK (disco possível).
H-PRGE	FEF8	BINTRP, na rotina PRGEND (fim de programa).
H-PRTF	FF52	BINTRP.
H-PTRG	FFA2	BIPTRG, na rotina PTRGET (obter flag).
H-QINL	FDEO	MSXINL, no início da rotina QINLIN (ponto de interrogação e linha de entrada).
H-READ	FF07	BINTRP, na entrada pronta.
H-RETU	FF4D	BINTRP.
H-RSET	FE26	MSXSTS, no início da rotina RSET (colocar à direita).

NOME	ENDEREÇO	DETALHES
H-RSLF	FE8F	SPCDSK, reselecionar antigo acionador.
H-RUNG	FECB	BIMISC, na rotina RUNC (liberar RUN).
H-SAVD	FE94	SPCDSK, gravar acionador atual.
H-SAVE	FE6C	SPCDSK, para gravar rotina.
H-SCNE	FF98	BINTRP.
H-SCRE	FFCO	MSXSTS, na entrada da declaração SCREEN.
H-SETF	FE53	SPCDSK, na rotina SETFIL (estabelecer flag de arquivo).
H-SETS	FDF4	MSXSTS, no início da rotina SETS (colocar atributo).
H-SNGF	FF39	BINTRP.
H-STKE	FEDA	BIMISC, na rotina STKERR (erro de pilha).
H-TIMI	FD9F	MSXIO, no início do manipulador de interrupção.
H-TOTE	FDBD	MSXIO, no início da rotina TOTEXT (força o modo de tela ao modo texto).
H-TRMN	FF61	BINTRP.
H-WIDT	FF84	BINTRP, na rotina WIDTH (largura).

CAPÍTULO 61

A RAM DO SISTEMA

Esta seção contém uma listagem de todas as variáveis do sistema, juntamente com seu endereço e comprimento em bytes. O endereço é fornecido em hexadecimal e o comprimento em decimal. A RAM do sistema se estende de &HF380 até &HFFFF. As posições &HF380 até &HF399, inclusive, na verdade contêm sub-rotinas de código de máquina utilizadas para administrar os slots:

NOME
ENDEREÇO
COMP.
NOME
ENDEREÇO
COMP.

NOME	ENDEREÇO	COMPRIMENTO	NOME	ENDEREÇO	COMPRIMENTO
ARG	F847	16	BUFEND	FC18	0
ARYTA2	F7B5	2	BUFMIN	F55D	1
ARYTAB	F6C4	2	CAPST	FCAB	1
ASPCT1	F40B	2	CASPRV	FCB1	1
ASPCT2	F40D	2	CENCNT	F933	2
ASPECT	F931	2	CGPBAS	F924	2
ATRBAS	F928	2	CGPNT	F91F	3
ATRBYT	F3F2	1	CLIKFL	FBD9	1
AUTFLG	F6AA	1	CLIKSW	F3DB	1
AUTINC	F6AD	2	CLINEF	F935	1
AUTLIN	F6AB	2	CLMLST	F3B2	1
BAKCLR	F3EA	1	CLOC	F92A	2

NOME	ENDEREÇO	COMPRIMENTO	NOME	ENDEREÇO	COMPRIMENTO
BASROM	FBB1	1	CLPRIM	F38C	14
BDRCLR	F3EB	1	CMASK	F92C	1
BOTTOM	FC48	2	CNPNTS	F936	2
BRDATR	FCB2	1	CNSDFG	F3DE	1
BUF	F55E	258	CODSAV	FBCC	1
CONLO	F66A	8	FNKSWI	FBCD	1
CONSAV	F668	1	FORCLR	F3E9	1
CONTXT	F666	2	FRCNEW	F3F5	1
CONTYP	F669	1	FRETOP	F69B	2
CPCNT	F939	2	FSTPOS	FBCA	2
CPCNT8	F93B	2	FUNACT	F7BA	2
CPLOTF	F938	1	GETPNT	F3FA	2
CRCSUM	F93D	2	GRPACX	FCB7	2
CRTCNT	F3B1	1	GRPACY	FCB9	2
CS120	F3FC	10	GRPATR	F3CD	2
CSAVEA	F942	2	GRPCGP	F3CB	2
CSAVEM	F944	1	GRPCOL	F3C9	2
CSCLXY	F941	1	GRPHED	FCA6	1
CSRSW	FCA9	1	GRPNAM	F3C7	2
CSRX	F3DD	1	GRPPAT	F3CF	2
CSRY	F3DC	1	GXPOS	FCB3	2
CSTCNT	F93F	2	GYPOS	FCB5	2
CSTYLE	FCAA	1	HEADER	F40A	1
CURLIN	F41C	2	HIGH	F408	2
CXOFF	F945	2	HIMEM	FC4A	2
CYOFF	F947	2	HOLD	F83E	8
DAC	F7F6	16	HOLD2	F836	8
DATLIN	F6A3	2	HOLD8	F806	48
DATPTR	F6C8	2	INSFLG	FCA8	1
DECCNT	F7F4	1	INTCNT	FCA2	2
DECTM2	F7F2	2	INTFLG	FC9B	1
DECTMP	F7F0	2	INTVAL	FCA0	2
DEFTBL	F6CA	26	JIFFY	FC9E	2
DEVICE	FD99	1	KANAMD	FCAD	1
DIMFLG	F662	1	KANAST	FCAC	1
DONUM	F665	1	KBUF	F41F	318
DORES	F664	1	KEYBUF	FBF0	40
DOT	F6B5	2	LFPROG	F954	1
DRWANG	FCBD	1	LINL32	F3AF	1
DRWFLG	FCBB	1	LINL40	F3AE	1
DRWSCL	FCBC	1	LINLEN	F3B0	1
DSCTMP	F698	3	LINTTB	FBB2	24
ENDBUF	F660	1	LINWRK	FC18	40
ENDFOR	F6A1	2	LOHADR	F94B	2
ENDPRG	F40F	5	LOHCNT	F94D	2
ENSTOP	FBB0	1	LOHDIR	F94A	1
ERRFLG	F414	1	LOHMSK	F949	1
ERRLIN	F6B3	2	LOW	F406	2
ERRTXT	F6B7	2	LOWLIM	FCA4	1
ESCCNT	FCA7	1	LPTPOS	F415	1
EXPTBL	FCC1	4	MAXDEL	F92F	2
FBUFFR	F7C5	43	MAXFIL	F85F	1
FILNAM	F866	11	MAXUPD	F3EC	3

NOME	ENDEREÇO	COMPRIMENTO	NOME	ENDEREÇO	COMPRIMENTO
FILNM2	F871	11	MCLFLG	F958	1
FILTAB	F860	2	MCLLEN	FB3B	1
FLBMEM	FCAE	1	MCLPTR	FB3C	2
FLGINP	F6A6	1	MCLTAB	F956	2
FNKFLG	FBCE	10	MEMSIZ	F672	2
FNKSTR	F87F	160	MINDEL	F92D	2
MINUPD	F3EF	3	RG6SAV	F3E5	1
MLTATR	F3D7	2	RG7SAV	F3E6	1
MLTCGP	F3D5	2	RNDX	F857	8
MLTCOL	F3D3	2	RS2IQ	FAF5	64
MLTNAM	F3D1	2	RTPROG	F955	1
MLTPAT	F3D9	2	RTYCNT	FC9A	1
MOVCNT	F951	2	RUNBNF	FCBE	1
MUSICF	FB3F	1	RUNFLG	F866	0
NAMBAS	F922	2	SAVEND	F87D	2
NEWKEY	FBE5	11	SAVENT	FCBF	2
NLONLY	F87C	1	SAVSP	FB36	2
NOFUNS	F7B7	1	SAVSTK	F6B1	2
NTMSXP	F417	1	SAVTXT	F6AF	2
NULBUF	F862	2	SAVVOL	FB39	2
OLDKEY	FBDA	11	SCNCNT	F3F6	1
OLDLIN	F6BE	2	SCRMOD	FCAF	1
OLDSCR	FCB0	1	SKPCNT	F94F	2
OLDTXT	F6C0	2	SLTATR	FCC9	64
ONEFLG	F6BB	1	SLTTBL	FCC5	4
ONELIN	F6B9	2	SLTWRK	FD09	128
ONGSBF	FBD8	1	STATFL	F3E7	1
OPRTYP	F664	0	STKTOP	F674	2
PADX	FC9D	1	STREND	F6C6	2
PADY	FC9C	1	SUBFLG	F6A5	1
PARM1	F6E8	100	SWPTMP	F7BC	8
PARM2	F750	100	T32ATR	F3C3	2
PATBAS	F926	2	T32CGP	F3C1	2
PATWRK	FC40	8	T32COL	F3BF	2
PDIREC	F953	1	T32NAM	F3BD	2
PLYCNT	FB40	1	T32PAT	F3C5	2
PRMFLG	F7B4	1	TEMP	F6A7	2
PRMLEN	F6E6	2	TEMP2	F6BC	2
PRMLN2	F74E	2	TEMP3	F69D	2
PRMPRV	F74C	2	TEMP8	F69F	2
PRMSTK	F6E4	2	TEMP9	F7B8	2
PROCNM	FD89	16	TEMPPT	F678	2
PRSCNT	FB35	1	TEMPST	F67A	30
PRTFLG	F416	1	TRCFLG	F7C4	1
PTRFIL	F864	2	TRGFLG	F3E8	1
PTRFLG	F6A9	1	TRPTBL	FC4C	78
PUTPNT	F3F8	2	TTYPOS	F661	1
QUEBAK	F971	4	TXTATR	F3B9	2
QUETAB	F959	24	TXTCGP	F3B7	2
QUEUEN	FB3E	1	TXTCOL	F3B5	2
QUEUES	F3F3	2	TXTNAM	F3B3	2
RAWPRT	F418	1	TXTPAT	F3BB	2
RDPRIM	F380	5	TXTTAB	F676	2

NOME	ENDEREÇO	COMPRIMENTO	NOME	ENDEREÇO	COMPRIMENTO
REPCNT	F3F7	1	USFLG	F6A6	0
RG0SAV	F3DF	1	USRTAB	F39A	20
RG1SAV	F3E0	1	VALTYP	F663	1
RG2SAV	F3E1	1	VARTAB	F6C2	2
RG3SAV	F3E2	1	VCBA	FB41	37
RG4SAV	F3E3	1	VCBB	FB66	37
RG5SAV	F3E4	1	VCBC	FB8B	37
VLZADR	F419	2	VOICCQ	FA75	128
VLZDAT	F41B	1	VOICEN	FB38	1
VOICAQ	F975	128	WINWID	FCA5	1
VOICBQ	F9F5	128	WRPRIM	F385	7

ÍNDICE ANALÍTICO

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Composição e Arte-Final:
JAG Composição Editorial e Artes Gráficas Ltda.
Praça F. Roosevelt, 208 - 8º andar
Tel. (011) 255-5694 - São Paulo

Impressão e Acabamento
GRÁFICA E EDITORA FCA
com filmes fornecidos pelo editor.
AV. HUMBERTO DE ALENCAR CASTELO BRANCO, 3972 - TEL.: 419-0200
SÃO BERNARDO DO CAMPO - CEP 09700 - SP

OUTROS LIVROS NA ÁREA

Avalon Software – *O Livro Vermelho do MSX*

Burd/Moreira – *MSX – Jogos – volumes 1 a 3*

Burd/Moreira – *MSX – Comandos Básicos – Guia do Operador*

Burd – *Simulações no MSX*

Bussab – *MSX – Música*

Carvalho – *Assembler para o MSX*

Casari – *MSX com Disk Drive*

Hoffman – *MSX – Guia do Usuário*

Marriot – *MSX – O meu Primeiro Livro*

0-07-450286-7