DIGI-LOG

DIGBASIC

ELITE

CAPÍTULO 3

CAPÍTULO 4

CAPÍTULO 5

FAZENDO AS LIGAÇÕES DO SEU MICROCOMPUTADOR DGT-100

Tire, com cuidado, o microcomputador e seus acessórios de dentro da embalagem.

Verifique as tensões dos três aparelhos e certifique se as chaves seletoras estão de acôrdo com a tensão local.
No computador e no vídeo esta chave está localizada na parte traseira, enquanto que no gravador cassete está localizada embaixo do mesmo.

ATENÇÃO: O POSICIONAMENTO ERRADO DA CHAVE PODE CAUSAR DANOS IRREPARÁVEIS AOS EQUIPAMENTOS.

CONEXÃO DO CASSETE AO MICROCOMPUTADOR

1 - Identifique os cinco cabos que acompanham o computador. Destes cinco cabos, quatro são iguais e um diferente.
O cabo diferente é aquele com plug vermelho em uma das extremidades. Este é o cabo do remoto e a extremidade vermelha deve ser conectada ao **REM** do gravador, enquanto que a outra extremidade deste cabo deve ser conectada ao **REM 1** do computador.

2 - Conecte qualquer um dos outros cabos entre o **MIC 1** do computador e o MIC do gravador.

3 - Um outro cabo deve ser conectado entre **EAR 1** do computador e o **MONITOR** do gravador.

OBS:
Se você tem dois gravadores, faça a conexão do segundo gravador da mesma maneira acima descrita, só que você deve usar MIC 2, REM 2 e EAR 2 para as conexões ao computador.

NÃO DESCONECTE O CABO DE ALIMENTAÇÃO DO GRAVADOR COM OS CABOS LIGADOS AO MICROCOMPUTADOR.

CONEXÃO DO VÍDEO AO MICROCOMPUTADOR

1 - Conecte um dos cabos que sobrou, entre TV do computador e o jack frontal do vídeo. Este jack está localizado ao lado dos três controles da TV: volume, brilho e tonalidade.

2 - Conecte o último cabo entre SOM do computador e o jack que está localizado na lateral direita da TV.

USANDO O GRAVADOR CASSETE

1 - Para ler um programa - isto é, passar para o computador um programa que está gravado na fita - coloque o volume em 5 e a tonalidade em 8.
Posicione a fita no início do programa que deseja ler.
Aperte a tecla **PLAY** do gravador.
Escreva **CLOAD** no computador e aperte a tecla **RETURN**. Assim que você apertar **RETURN**, o gravador começará a rodar e dois asteriscos devem aparecer no canto superior direito do vídeo, sendo que o asterisco da direita deve piscar indicando que o computador está lendo algo.
Quando o computador acaba de ler o programa inteiro, o gravador desligará automaticamente e a mensagem **READY** aparecerá no vídeo.
Neste ponto o programa já está na memória do computador e você pode rodá-lo escrevendo **RUN** e apertando a seguir a tecla **RETURN**.

2 - Se o programa for em linguagem de máquina, consulte o comando **SYSTEM** neste manual.

3 - Se os asteriscos não piscarem ou não aparecerem no vídeo, tente novamente com um volume e/ou tonalidade diferente.

4 - Use o contador digital do cassete para facilitar a localização de programas.

5 - Para usar REW (Rewind) e FF (Fast Forward) você deve desconectar o plug do remoto.

6 - Para gravar um programa - isto é, passar para a fita um programa que está na memória do computador - coloque a fita no cassete e aperte simultaneamente as teclas **PLAY** e **RECORD**.
Escreva então **CSAVE** seguido de uma letra entre aspas, por exemplo **CSAVE"A"**, e aperte a tecla **RETURN**.
Você ouvirá o que está sendo gravado através do som da TV.
Quando o computador acabar de gravar o programa na fita, a mensagem **READY** aparecerá no vídeo e você terá o programa guardado na fita.
O programa continua ainda na memória do computador.
É bom costume fazer 2 ou 3 gravações do mesmo programa, pois em caso de defeito localizado na fita você terá outra cópia do programa.

7 - Para verificar se uma fita contém material gravado, você pode escutá-la desconectando os plugs MONITOR e REM do gravador.

8 - Não deixe as teclas **PLAY e/ou RECORD** pressionadas desnecessariamente.

9 - Antes de ler um programa certifique-se que a fita está posicionada em uma parte onde não há gravação, pois se o computador começar a ler um programa no meio de um precedente, você perderá o controle do computador. Neste caso você deve apertar o botão **RESET** do computador.

1Ø - Quando o botão **RESET** é pressionado, você acessa o programa monitor DIGBUG. Para retornar ao Basic aperte a tecla "Q".

NOTA:
A numeração indicada pelo CONTADOR de seu GRAVADOR pode ser diferente de numeração indicada por OUTRO GRAVADOR.
Portanto você deve anotar a numeração que é indicada no seu contador.
Este fato é devido a caracteristica mecânica do contador, o que o torna impreciso.

ALGUNS MACETES SOBRE O USO DO CASSETE

A interface para cassete do seu microcomputador DGT-1ØØ, é um método seguro e rápido para o armazenamento de programas.
No entanto, diversos fatores podem afetar o desempenho do sistema de cassete. Entre estes fatores estão o próprio gravador, a fita que se usa, o volume e a tonalidade.

GRAVADOR

Para se manter o gravador em boas condições de uso, é necessário alguma manutenção de rotina.
Com o uso, o cabeçote torna-se sujo e magnetizado. A sujeira provoca uma perda de até 5Ø% do volume e a magnetização causa distorção. Por isso, é recomendada a limpeza e a desmagnetização periódica do cassete. Lembre-se que o cabo MIC deve sempre ser conectado, pois ele contém o fio terra.

FITA

Recomendamos o uso de uma fita cassete com uma boa resposta de freqüência e uma boa qualidade de mecânica, isto é, a fita deve rodar livremente. Guarde a fita dentro da caixa e longe do calor e de campos elétrico e magnético.

VOLUME E TONALIDADE

Um volume baixo causa a perda de informação, enquanto que um volume alto causa distorção. Ambas as situações causam erro durante a leitura de uma fita.
O volume e a tonalidade recomendados são os seguintes:

	PROGRAMAS GRAVADOS NO SEU GRAVADOR		PROGRAMAS GRAVADOS EM OUTRO GRAVADOR	
	VOLUME	TONALIDADE	VOLUME	TONALIDADE
BASIC	5 - 6	8.5 - 9.5	8.5-9.5	5 - 6
SYSTEM	5 - 6	8 - 9.5	5 - 5.5	8 - 8.5

Estes dados são experimentais e você pode encontrar uma faixa que melhor se adapte ao seu gravador.
Quando o computador começar a ler uma informação, dois asteriscos deverão aparecer no canto superior direito do vídeo. O asterisco da direita deverá piscar até o fim do programa.
Se os asteriscos não aparecerem ou então não piscar isto é sinal de que a informação não está sendo lida. Neste caso você deverá então, tentar um novo volume e/ou tonalidade.

Use o botão **RESET** para retomar o controle do computador caso tenha problemas durante a leitura de uma fita.

Quando o botão **RESET** é pressionado, você acessa o programa monitor DIGBUG. Para retornar ao Basic aperte a tecla "Q".

OBSERVAÇÕES FINAIS

1 - Uma boa prática é sempre gravar o mesmo programa 2 ou até 3 vezes. Isto diminui suas chances de perder um programa.

2 - Quando fizer a leitura ou gravação de um programa em basic, use o comando **CLOAD**? para comparar a fita com aquilo que está na memória do computador.

INTRODUÇÃO

INICIAÇÃO

Assim que você ligar o seu microcomputador, a pergunta **PROTEGER ?** aparecerá no vídeo. Simplesmente aperte a tecla **RETURN** e o vídeo então apresentará:

```
DIGBASIC II

READY
>-
```

Isto quer dizer que o computador está pronto para fazer qualquer coisa que você desejar. Sempre que você der um comando ao computador, você deve terminar o comando com **RETURN** . Quando você aperta a tecla **RETURN** , você está dizendo ao computador para analisar e agir de acôrdo com aquilo que você acaba de escrever.

Por exemplo, escreva o seguinte:

```
PRINT MEM   RETURN   - Aqui você está pedindo ao computador
                       para imprimir, no vídeo, a quantidade
                       de memória disponível para programação.
```

O computador responderá:

```
15570

READY
>-
```

PROGRAMAS

Um programa de computador é uma série de instruções que você guarda no computador e depois manda que ele as execute através do comando **RUN.**

MODOS DE OPERAÇÃO

Existem quatro modos de operação do computador:

-COMANDO
-EXECUÇÃO
-EDIÇÃO
-MONITOR

1- MODO DE COMANDO

Você está no modo de comando sempre que o computador apresenta >- no vídeo. Isto é, você está no comando do computador e ele está esperando alguma instrução de sua parte para poder agir. Os comandos que você dá ao computador são executados assim que a tecla RETURN é apertada.
Aquele **PRINT MEM** que você escreveu só foi possível porque você estava no modo de comando.

2- MODO DE EXECUÇÃO

Você cria um programa enquanto está no modo de comando e quando deseja executá-lo, você deve passar para o modo de execução, através do comando **RUN.**

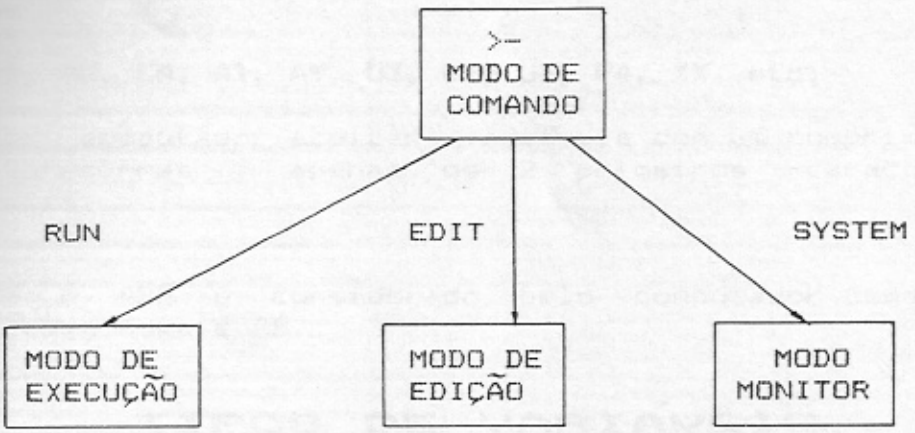
3- MODO DE EDIÇÃO

Este é o modo que lhe ajuda a fazer modificações e reparos em programas já criados. Você passa para este modo através do comando **EDIT.**

4- MODO MONITOR

Este modo lhe permite carregar programas em linguagem de máquina na memória do computador. Você passa para este modo através do comando **SYSTEM.**

Podemos resumir os modos de operação da seguinte forma: Sempre que aparece no vídeo os caracteres >- você está no modo de comando. Através de diversos comandos você passa para outros modos da seguinte forma:



MODO DE CALCULADORA

Você também pode ter o seu computador funcionando como uma calculadora.

Exemplo:
Se você deseja somar 179 com 253 , escreva o seguinte:

```
PRINT 179 + 253  RETURN
```

A ordem que o computador faz as operações é a seguinte: os parênteses são executados primeiramente. Dentro e fora dos parênteses é obedecido a seguinte ordem: exponenciação; negação; multiplicação; divisão, (da esquerda para a direita); maior, menor e igual (esquerda para direita); NOT, AND, OR.

Se você prefere que o resultado seja atribuído a uma variável, você deve escrever:

```
A = 179 + 253   RETURN
```

Neste caso o computador fez a conta e o resultado foi atribuído a variável A, ao invés de apresentá-lo no vídeo.Se você quer ver o resultado escreva:

```
PRINT A   RETURN
```

A letra A é uma variável que pode ser usada em qualquer outra situação.

Exemplo:
```
PRINT  A + Z
```

Assim como você atribuiu a letra A a operação feita, você poderia ter usado qualquer letra (A ate Z) e ainda poderia usá-la seguida de qualquer outra letra ou número. Portanto você poderia ter usado qualquer uma das 900 combinações possíveis. Eis alguns exemplos:

A, AA, AB, AC, CA, A1, A9, 03, FT, UZ, P4, XX, etc.

Apesar do computador aceitar variáveis com um comprimento maior que 2 caracteres , apenas os 2 primeiros caracteres são considerados.

Exemplo:
ABC, ABD e ABS é considerado pelo computador como uma única variável cujo nome é AB.

TIPOS DE VARIÁVEIS

Foi tratado acima dos nomes que você pode dar para as variáveis. Com certos caracteres suplementares você atribui a variável um tipo específico. Estes tipos são: inteiro, precisão simples, precisão dupla e string.

- String é um tipo de variável que além de guardar números pode guardar qualquer caracter alfanumérico. Você diz que a variável é uma string quando acrescenta o sufixo $ ao nome da variável.

Exemplo:
A$ = "COMPUTADOR"

Quando você quizer imprimir esta string, simplesmente escreva:
PRINT A$ RETURN

- Você diz que uma variável é inteira (números inteiros de -32769 a 32767) quando usa o sufixo %.

Exemplo:
A% = 45
B7% = 1350

- Com o sufixo ! ou E você tem variáveis de precisão simples (6 algarismos significativos).

A! = 3.14159
BF! = 7.23 E -3

- O sufixo # ou D declara que uma variável e de precisão dupla (16 algarismos significativos).

A# = 3.141592653589
GC# = 5.3475143311 D + 37

OBS. - Se você não colocar nenhum sufixo, a variável é considerada de precisão simples.

CAPÍTULO 1

FUNÇÕES MATEMÁTICAS

Neste capítulo vamos descrever as operações básicas de seu computador assim como as funções matemáticas especiais.
O computador usa os seguintes caracteres para as operações básicas:
+ - adição
- - subtração
* - multiplicação
/ - divisão
↑ - exponencial

Exemplos:

```
PRINT  4 + 2
PRINT  4 * 3 ↑ (1/5)
```

As funções especiais são:

ABS	INT
ATN	LOG
CDBL	RANDOM
CINT	RND
COS	SGN
CSNG	SIN
EXP	SQR
FIX	TAN

Em todas estas funções, o argumento deve estar incluido entre parênteses e pode ser tanto uma variável ou um número.

Exemplo:
Se A = 1.57 então SIN (1.57) e SIN (A) resulta numa mesma resposta do computador.

ABS (arg)

Esta função lhe fornece o valor absoluto do argumento.

ATN (arg)

Fornece o arco-tangente do argumento. Todas as funções trigonométricas fornecem e usam radianos.

Exemplo:
PRINT ATN (1)

CDBL (arg)

Converte o argumento em um valor com precisão dupla.

```
PRINT  CDBL (123%)/93%
```

CINT (arg)

Converte para inteiro, fornecendo o maior número inteiro logo abaixo do argumento.

```
PRINT  CINT (4.231)  fornece 4
PRINT  CINT (-4.231)  fornece -5
```

COS (arg)

Fornece o cosseno do argumento. Lembre-se que o argumento deve estar em radianos.

```
PRINT  COS (Ø.312)
```

CSNS (arg)

Converte o argumento para precisão simples.

```
PRINT  CSNG (.123456789Ø12)  fornece  .123457
```

EXP (arg)

Fornece e ↑ (arg) , sendo 'e' a base dos logarítimos naturais.

```
PRINT  EXP (1)  -  fornece o número 'e' .
```

FIX (arg)

Fornece a parte inteira do argumento.

```
PRINT  FIX (4.3)   fornece 4
PRINT  FIX (-4.3)  fornece -4
```

INT (arg)

Fornece o inteiro do argumento, usando o maior número inteiro que não seja maior que o argumento.

```
PRINT  INT (4.3)    fornece 4
PRINT  INT (-4.3)   fornece -5
```

LOG (arg)

Fornece o logarítimo natural do argumento, isto é, ln (arg).

RND (arg)

Fornece um número inteiro aleatório, usando como semente o número gerado quando se executou pela última vez a função f **RANDOM.**
Exceção: RND (Ø) fornece um número em precisão simples entre Ø e 1.

SGN (arg)

Fornece o sinal do argumento. Fornece -1 para arg negativo, Ø para arg. zero e 1 para arg positivo.

Exemplo:

Se A = -3 e B = 5 então
PRINT SGN (A * B) fornece -1

SIN (arg)

Fornece o seno do argumento. Lembre-se que o argumento deve ser em radianos.

SQR (arg)

Fornece a raiz quadrada do argumento. Se A = 3 e B = 4 , então:

PRINT SQR (A↑2 + B↑2) fornece 5

TAN (arg)

Fornece a tangente do argumento.

CAPÍTULO 2

COMANDOS E INSTRUÇÕES

Existe uma pequena diferença entre um comando e uma instrução.
Comandos são executados assim que você os escreve e aperta a tecla RETURN .
Instruções são colocadas em programas e são executados quando você dá o comando **RUN**.
Portanto as instruções são executadas sómente quando você está no modo de execução e os comandos são executados assim que você aperta a tecla RETURN.
Todas as definições dadas a seguir podem ser usadas como comandos ou como instruções com as seguintes exceções:

CONT - só pode ser usado como comando.
INPUT, INPUT # e PRINT # - só podem ser usados como instrução.

NEW

Este comando apaga qualquer programa que exista na memória do computador, deixando todas as linhas disponíveis para programação.

LIST

Este comando faz uma listagem do programa que está na memória do computador. Você pode listar todo o programa, apenas uma parte ou então apenas uma linha.
Para você fazer uma pausa durante a listagem aperte as teclas **SHIFT e @** simultaneamente. Quando você apertar qualquer outra tecla a listagem continuará.

Exemplos:

```
LIST    ==> lista todo o programa que está na memória.
LIST 10Ø   ==> lista apenas a linha 10Ø.
LIST 10Ø - 19Ø   ==> lista as linhas de 10Ø a 19Ø.
LIST -10Ø   ==> lista todas as linhas até 10Ø.
LIST 10Ø-   ==> lista todas as linhas a partir de 10Ø.
```

DELETE

Se você deseja apagar apenas uma parte do programa use o camando DELETE ao invés de **NEW** que apaga todo o programa.

Exemplos:

```
DELETE 5Ø    ==> apaga a linha 5Ø do programa.
DELETE -5Ø   ==> apaga todas as linhas do início até a linha 5Ø.
DELETE 3Ø - 5Ø  ==> apaga todas as linhas entre 3Ø e 5Ø.
```

RUN

Este comando faz com que o computador execute o programa que está na memória. Quando o comando **RUN** é executado, todas as variáveis numéricas e os strings são zeradas.

Exemplo:

```
RUN  ==> a execução de programa começa na linha com o número
mais baixo.
RUN 1ØØØ  ==> a execução começa na linha 1ØØØ.
```

Se você deseja começar a execução de um programa sem zerar todas as variáveis e strings use o comando **GOTO.**

AUTO

Este comando acessa a numeração automática de linhas de programação. Isto significa que você não precisa bater os números das linhas de programação, pois isto é feito automaticamente. Cada vez que você terminar de bater uma linha, o computador acessará a próxima linha.

Exemplo:

```
numeração automática das linhas -
AUTO                        1Ø,2Ø,3Ø...
AUTO 1Ø,5                   1Ø,15,2Ø,25...
AUTO 1,1                    1,2,3,4...
AUTO 2ØØ                    2ØØ,21Ø,22Ø...
AUTO 1,1ØØ                  1,1Ø1,2Ø1,3Ø1...
```

Para terminar a função **AUTO** , aperte a tecla **BREAK.**
Se após o número da linha aparecer um * , isto quer dizer que aquela linha já está usada. Se você não deseja reprogramar esta linha, aperte **BREAK** .

PRINT

Este comando manda para o vídeo, os ítens que o seguem. Estes ítens podem ser:

1) Mensagem entre aspas

Exemplo:

```
10 PRINT  "computador"
```

2) Strings

Exemplo:

```
10 A$  = "computador"
20  PRINT  A$
```

3) Números

Exemplo:

```
10  PRINT  25
```

4) Variáveis

```
10 A = 123
20 PRINT A
```

5) Ou ainda uma combinação de todos os ítens acima.

```
10  PRINT  "computador" , A$, 25, A
```

Note que os ítens acima foram separados por vírgula. A vírgula instrui o computador, para avançar até a próxima zona de tabulação automática antes de mandar os ítens para o vídeo.
O vídeo contém quatro zonas de tabulação. Você pode visualizar estas zonas com o seguinte programa:

```
10  PRINT  "zona 1", "zona 2", "zona 3", "zona 4"
```

Se você separar os ítens com ponto e vírgula, os ítens são escritos sem espaço entre eles.

Exemplo:

```
10  PRINT  "COMPUTADOR";"DGT-100", "COMPUTADOR"; " DGT-100"
```

Observe os exemplos abaixo:

```
10  PRINT  "zona 1",,, "zona 4"

10  PRINT  "COMPUTADOR"
20  PRINT  "DGT-100"

10  PRINT  "COMPUTADOR";
20  PRINT  " DGT-100"
```

Observe nos dois últimos exemplos que se a instrução terminar com pontuação (vírgula ou ponto e vírgula), o computador guarda esta posição e a considera no próximo PRINT.
Se a instrução terminar sem pontuação, o computador executa um "line feed" automático, isto é, passa para o início da próxima linha.

INPUT

Este comando faz com que o computador paralise a execução do programa e fique esperando uma dado do teclado. Este dado pode ser um número ou uma string.

Exemplo:

```
10 PRINT "Qual o seu nome"
20 INPUT A$
30 PRINT "OLA ";A$
```

Você pode condensar este programa da seguinte forma:

```
10 INPUT "Qual o seu nome";A$
20 PRINT "OLA ";A$
```

Note que após a execução do comando **INPUT** o computador imprime um ? - no vídeo.
Observe o exemplo a seguir:

```
10 INPUT A$,B,T$,C
```

Esta instrução pede que você entre quatro ítens na seguinte ordem: uma string, um número, outra string e outro número.
Você pode fornecer todos os ítens de uma vez (separados por vírgula) ou um de cada vez.
Se você fornecer mais ítens que o pedido a mensagem **EXCESSO DE DADOS** aparecerá. Se você fornecer uma string quando o computador está esperando um número, a mensagem **'REDO'** aparecerá e lhe dará chance de corrigir o erro.
Se você simplesmente apertar a tecla **RETURN** as variáveis assumirão o valor que elas continham antes da execução da instrução **INPUT.**

GOTO

Este comando altera a execução normal do programa que é da linha de número menor para a linha de número maior.

Exemplo:

```
10 PRINT "ESTE E";
20 GOTO 50
30 PRINT "COMANDO GOTO"
40 GOTO 70
50 PRINT " UM EXEMPLO DO ";
60 GOTO 30
70 END
```

O comando **GOTO** pode ser usado no modo de comando quando você quer iniciar um programa sem "zerar" todas as variáveis e strings.

ON...N...GOTO

Este é um **GOTO** de multiplas transferências. Neste comando o computador transfere a execução do programa para a n-esima linha especificada após **GOTO**.

Exemplo:

```
1Ø INPUT "Bata um numero de 1 a 4"; N
2Ø ON N GOTO 3Ø,5Ø,7Ø,9Ø
3Ø PRINT "Você apertou o numero 1"
4Ø GOTO 1Ø
5Ø PRINT "Você apertou o numero 2"
6Ø GOTO 1Ø
7Ø PRINT "Você apertou o numero 3"
8Ø GOTO 1Ø
9Ø PRINT "Você apertou o numero 4"
1ØØ GOTO 1Ø
```

Note no exemplo acima que se n e 1 a execução vai para a linha 3Ø. Se n = 2 a execução pula para a linha 5Ø, que é a segunda linha especificada no **GOTO**. Se n = 3 a execução pula para a linha 7Ø, que é a terceira linha especificada no **GOTO** e assim sucessivamente.

FOR...TO...STEP

NEXT

Você pode criar um loop sem fim com o seguinte programa:

```
1Ø PRINT "Loop sem fim"
2Ø GOTO 1Ø
```

Para você terminar este loop você deve apertar a tecla **BREAK**.

Agora, se você deseja executar um loop apenas um determinado número de vezes, você pode usar o comando FOR...TO...STEP NEXT

Exemplo:

```
1Ø FOR N = 1 TO 1Ø STEP1
2Ø PRINT "Esta mensagem sera impressa 1Ø vezes"
3Ø NEXT N
```

Este loop funciona da seguinte forma: a primeira vez que o comando **FOR** é executada , a variável (no exemplo acima usamos N) é setada no valor inicial (no exemplo usamos 1).
A execução do programa continua até que se encontre a instrução **NEXT**. Neste ponto, a variável é incrementada no valor especificado por **STEP** (no exemplo usamos 1), e comparada com o valor final da variável especificada após TO (no nosso caso é 1Ø). Neste ponto podem acontecer duas coisas: a variável incrementada é maior que o valor final ou então é menor ou igual a este valor.

Se fôr maior que o valor final, o loop é fechado e a execução do programa continua com a instrução após **NEXT.**
Se a variável não exceder o valor final, a execução do programa continua na instrução após FOR...TO...STEP.

Exemplos:

Usando outras variáveis e valores podemos imprimir aquela mensagem por 10 vezes.

```
10 FOR X = 12 TO 120 STEP 12
20 PRINT "Esta mensagem sera impressa 10 vezes"
30 NEXT X
```

Observe:

```
10 FOR K = 7 TO 16
20 PRINT K
30 NEXT K
```

Se o valor do **STEP** for 1, não é preciso especificá-lo. Note como o valor K é incrementado.

```
ou ainda
10 FOR K = 16 TO 7 STEP-1
20 PRINT K;
30 NEXT K
```

O valor do **STEP** pode ser negativo. Neste caso a variável é decrementada cada vez que a instrução **NEXT** é executada.

```
10 FOR Z = 0 TO 1 STEP .3
20 PRINT Z
30 NEXTZ
```

Após Z = .9, Z é incrementado de .3 o que resulta em Z = 1.2, que é maior que o valor final especificado para Z.

```
10 A = 5 : B = 15 : C = 3
20 FOR X = A TO B STEP C
30 PRINT X
40 NEXT X
```

Os valores podem ser variáveis.

```
10 FOR A = 1 TO 3
20 PRINT "Loop 1"
30 FOR B = 1 TO 4
40 PRINT "loop 2"
50 NEXT B
60 NEXT A
```

Você pode ter vários loops, sendo um dentro do outro. Para cada valor de A, o loop 2 é executado totalmente.Você pode condensar as linhas 50 e 60 em apenas uma:

```
50 NEXT B,A
```

IF...THEN...

Quando o computador encontra esta instrução ele testa a expressão que está entre **IF e THEN.**
Esta expressão pode ser verdadeira ou falsa. Se for verdadeira o computador executa a instrução que vem depois do **THEN.** Se a expressão for falsa o computador ignora a instrução depois do **THEN**, e executa a próxima linha de programação.

Exemplos:

```
10 IF  N = 3  THEN PRINT "N E IGUAL A TRES"

10  IF  N > 3  AND  N < 5  THEN PRINT "O VALOR DE N ESTA ENTRE 3 E
5"
```

IF...THEN...ELSE...

A instrução que vem depois de **ELSE** é a instrução que o computador deve executar caso a expressão seja falsa. Portanto se no comando **IF...THEN**, você não usa **ELSE**, e a expressão for falsa, o computador executa a próxima linha de programação. Se você usar **ELSE**, o computador executa a instrução que vem após **ELSE**. Estude o exemplo abaixo:

```
10 INPUT "APERTE A TECLA DE NÚMERO 3"; N
20 IF N = 3 THEN PRINT  "MUITO  OBRIGADO"  ELSE  PRINT  "VOCE  NAO
APERTOU O NUMERO 3"
30 GOTO 10
```

REM

O computador ignora tudo que vem depois do comando **REM.** Este comando é usado para documentar um programa.

Exemplo:

```
10 REM Este programa calcula a raiz quadrada de um número.
20 INPUT A
30 PRINT SQR (A)
40 GOTO 20
```

DEFINT, DEFSNG, DEFDBL, DEFSTR

Na introdução deste manual dissemos que o computador usa quatro tipos de variáveis: inteiro, precisão simples, precisão dupla e string.
Com o uso dos sufixos % , ! ou E , # ou D e $ você diz que as variáveis são inteiras, precisão simples, precisão dupla ou strings, respectivamente.
Mas você tem outra maneira de declarar o tipo de uma variável:

DEFINT diz que uma variável ou conjunto de variáveis são números inteiros. Há duas vantagens em usar números inteiros : você gasta menos memórias e as operações aritméticas são executadas mais rapidamente com números inteiros.

DEFSNG declara que uma variável ou conjunto de variáveis são números de precisão simples.

DEFDBL declara que uma variável ou conjunto de variáveis são números de precisão dupla.

DEFSTR declara que uma variável ou conjunto de variáveis devem ser tratados como strings.

Exemplo:

Suponha que num determinado programa você precisa que as variáveis de A até F sejam inteiras, as variáveis G e H sejam de precisão simples, as variáveis de I até M sejam de precisão dupla e que as variáveis P, R e X sejam strings. Portanto o programa deve iniciar da seguinte maneira :

```
10 DEFINT A-F
20 DEFSNG G,H
30 DEFDBL I-M
40 DEFSTR P,R,X
```

NOTA: Você pode colocar várias instruções numa mesma linha, desde que o comprimento total da linha não ultrapasse 255 caracteres.
Para você colocar várias instruções numa mesma linha, basta separar as instruções com : .
No exemplo acima, você pode colocar :

```
10 DEFINT A-F : DEFSNG G,H : DEFDBL I-M : DEFSTR P,R,X
```

DATA

Geralmente há uma série de dados constantes que são usados durante a execução de um programa. Estes dados são armazenados no programa através do comando **DATA.** Os dados são acessados seqüencialmente através do comando **READ.** Os dados são separados por vírgula e podem ser números ou strings.

READ

Este comando lê os dados armazenados com o comando DATA. Os dados são lidos seqüencialmente, começando com o primeiro ítem do comando **DATA** e terminando com o último ítem do comando DATA. Os dados do comando **DATA** devem ser coerentes com os tipos de variáveis que o comando **READ** lê.

Exemplo:

```
10  READ A,B$,C$,D
20  DATA 13, PROGRAMA, EXEMPLO, 130
```

Após a execução deste programa teremos as seguintes variáveis :

```
A = 13
B$ = COMPUTADOR
C$ = EXEMPLO
D = 130
```

```
10  PRINT "NOME", "SALARIO"
20  READ N$, S
30  IF N$ = "FIM" THEN PRINT "FIM DA LISTA" : END
40  PRINT N$,S
50  GOTO 20
60  DATA JOSE, 10.000, MARIA, 15.000, ANTONIO
70  DATA 8.000, MARCOS, 20.000, FIM,0
```

RESTORE

Este comando faz com que o próximo dado lido pelo comando READ seja o primeiro dado do primeiro comando **DATA.** Isto lhe permite reutilizar os dados do comando **DATA.**

Exemplo:

```
10 READ A
20 RESTORE
30 READ B
40 PRINT A,B
50 DATA 10,40
```

Execute este programa e note que A e B tem o valor 10. Isto porque o comando **RESTORE** na linha 20 faz com que o comando **READ B** da linha 30 leia o valor 10 para B.

GOSUB E RETURN

Este comando transfere a execução do programa para uma subrotina especificada pelo número da linha que vem após **GOSUB.**
O comando **RETURN** faz com que o computador retorne da subrotina, executando a linha imediatamente após **GOSUB.**

Exemplo:

```
10 GOSUB 100
20 PRINT "RETORNO DA SUBROTINA" : END
100 PRINT "INICIO DA SUBROTINA DA LINHA 100"
110 RETURN
```

END

Este comando termina a execução do programa.
O comando **END** é geralmente usado para forçar o término do programa em um ponto que não seja o fim do programa.

STOP

Este comando interrompe a execução do programa, simulando a tecla **BREAK** .
Usa-se **STOP** como um recurso para debugar programas. Durante o break você pode examinar e mudar variáveis.
O comando **CONT** pode, então, ser usado para continuar a execução do programa (se o programa for alterado durante o break, o comando **CONT** não pode ser usado).

CONT

Se você usa o comando **STOP**, ou então aperta a tecla **BREAK** durante a execução de um programa, a execução é interrompida. Durante esta interrupção você pode verificar e mudar valores de variáveis e depois recomeçar a execução do programa (a partir do ponto onde foi interrompido) com o comando **CONT.**

OBS:
O comando **CONT** não pode ser usado após a edição de uma linha.

DIM

Um arranjo ou "array" é simplesmente uma lista ordenada de ítens. O comando **DIM** permite que se diga ao computador o tamanho do arranjo.

Exemplo:

```
10 DIM A(6),B(2,4),C$(15)
```

Esta instrução abre 3 arranjos : o arranjo A, com uma dimensão e elementos de Ø a 6; o arranjo B, com duas dimensões e elementos de Ø,Ø a 2,4; e o arranjo de strings C$, com uma dimensão e elementos de Ø a 15.

```
10 INPUT "Qual é o numero de nomes";N
20 DIM A(N,2)
```

Pode-se especificar o tamanho do arranjo através de um número ou de uma variável.
Se a instrução **DIM** não for mencionada, um tamanho de 11 (elementos de Ø a 1Ø) e presuposto para cada dimensão.

SET RESET

O seu microcomputador pode desenhar uma grande variedade de figuras no vídeo.
Existem dois comandos que lhe permitem desenhar:

SET - acende um ponto no vídeo.
RESET - apaga um ponto no vídeo.

Para fins gráficos o vídeo é dividido em 128 pontos horizontais por 48 pontos verticais. Os pontos horizontais são numerados da esquerda para a direita (de Ø até 127). Os pontos verticais são numerados de cima para baixo (de Ø até 47).

Exemplo:

```
10 SET(64,24)
```

Esta instrução acende um ponto no meio do vídeo.Note que este comando requer dois argumentos: o primeiro é a coordenada horizontal e o segundo é a coordenada vertical.
Genericamente: SET(x,y)

```
20 RESET(64,24)
```

Simplesmente apaga o ponto que foi acendido com **SET**.

Os argumentos x e y podem ser números, variáveis ou expressões.

Exemplo:

```
10 INPUT X,Y
20 SET(X,Y)
30 GOTO10
```

Este programa acende o ponto escolhido na linha 10.

Veja se você descobre o que os dois programas abaixo fazem antes de rodá-los.

```
10 SET(60,20)
20 GOSUB 100
30 RESET(60,20)
40 GOSUB 100
50 GOTO10
100 FOR R=0TO300
110 NEXTR
120 RETURN
```

```
10 FOR X=0TO127
20 FOR Y=0TO47
30 SET (X,Y)
40 NEXTY
50 NEXTX
60 GOTO 60
```

CLS

Este comando é um **RESET** total, isto é, é como um **RESET** no vídeo inteiro.
CLS é usado quando se quer limpar o vídeo, antes de apresentar um resultado.

POINT

Testa se um determinado ponto do vídeo está apagado ou aceso. Se o ponto estiver aceso, então a instrução **POINT** fornece -1 (Binário verdadeiro). Se o ponto estiver apagado, **POINT** fornece 0 (Binário falso).

POS

Indica a posição do cursor na linha (de 0 até 63).
É necessário um argumento qualquer que não é utilizado.

Exemplo:

```
10 PRINT POS (0)
```

PEEK, POKE

Estes comandos permitem a manipulação direta de posições de memórias.
Com **PEEK** você verifica o conteúdo de qualquer posição da memória. O valor fornecido por **PEEK** é em decimal.

Exemplo:

```
10 A = PEEK (18135)
```

Após a execução desta instrução, a variável A contém o mesmo valor que está armazenado na memória cujo endereço é 18135.
O comando **POKE** permite que seja colocado um valor numa localização especifica de memória. **POKE** requer dois argumetos: o endereço em decimal e um valor (de Ø até 255) também em decimal.
Antes de usar o comando **POKE**, saiba exatamente em que lugar da memória você está colocando os dados. Cuidado para não usar POKE em áreas críticas, como o stack.

Exemplo:

```
10  PRINT PEEK (18850)
20  POKE 18850, 7
30  PRINT PEEK (18850)
```

Na linha 1Ø apresenta-se no vídeo o dado que está na memória cujo endereço é 18850. Na linha 2Ø colocamos o número 7 nesta localização e na linha 3Ø verificamos se realmente o 7 está armazenado.

OBS.:
Para usar **PEEK e POKE** em endereços acima de 32767, use a seguinte fórmula:

- (65536 - endereço desejado)

Exemplo:

Para usar **PEEK** no endereço 32900 use:
PEEK (-32636)

OPERADORES LÓGICOS: AND OR NOT

Os operadores lógicos são usados para manipulações de bits e operações Booleanas.
AND, OR e NOT convertem seus argumentos em números inteiros (com sinal) em complemento de dois de 16 bits. Isto dá uma faixa de -32768 até +32767. Após esta conversão é feita a operação lógica e apresentado um número na mesma faixa.

As operações são realizadas bit-a-bit.
Exemplos:

3 OR 4 = 7
3 é 11 em binário e 4 é 100 em binário. O resultado de uma operação OR entre estes dois números é 111, que é 7 em decimal.

17 AND 17 = 17
17 é 11111 e 17 é 10001. 11111 AND 10001 resulta 10001, que é 17 em decimal.

-1 AND 6 = 6
-1 em binário é 1111111111111111 e 6 é 110, o resultado é 110 que é 6 em decimal.

8 AND 4 = 0
8 é 1000, 4 é 100 portanto o resultado é 0

6 OR 2 = 6
6 é 110, 2 é 10. Uma operação lógica OR entre dois binários resulta em 110, que é 6 em decimal.

-1 OR -2 = -1
O decimal -1 é 1111111111111111 em binário e -2 é 1111111111111110. Um OR entre estes dois números resulta em 1111111111111111 que é -1.

NOT 0 = -1
A negação de 0 em 16 bits é 1111111111111111, que é -1.

NOT -1 = 0
-1 é 1111111111111111 que negado resulta em 0

NOT X = -(X + 1)

Os operadores lógicos podem também ser usados para avaliar expressões.
Exemplo:

10 IF A > 3 AND A < 7 THEN PRINT "A é maior que 3 e menor que 7"

INP

Lê um byte de uma porta de entrada. Pode-se ter até 256 portas de entrada. A porta número 255 já é usada pelo computador para a interface de cassete.

Exemplo:

```
10 PRINT INP (121)
```

Esta instrução lê o byte que está na porta 121 e imprime o valor (em decimal) no vídeo.

OUT

Manda um byte para uma porta de saída. São requeridos 2 argumentos neste comando: o número da porta e o byte que deve ser mandado.

Exemplo.

```
10 OUT 140, 37
```

O valor 37 é enviado à porta 140.

OBS.: Tanto INP quanto OUT são instruções que dependem muito do hardware. A interface de uma porta, com aplicações específicas, deve ser projetada pelo usuário.

MEM

Este comando informa a quantidade de memórias que estão disponíveis ao usuário.

Exemplo:

PRINT **MEM** Fornece a quantidade de memórias não usadas e não protegidas.

TRON

Este comando permite que você acompanhe, linha por linha , a execução de um programa. Cada vez que o programa passa para uma nova linha, o número desta linha é impressa no vídeo.

Exemplo:

```
10 PRINT  "Linha 1Ø"
20 PRINT  "Linha 2Ø"
30 GOTO 4Ø
40 PRINT  "Linha 4Ø"
50 END
```

Antes de rodar o programa escreva **TRON** e aperte RETURN.

TROFF

Use TROFF para terminar a função **TRON.**
TRON e **TROFF** quando usados dentro de programa, ajuda a verificar se uma determinada linha é executada.

Exemplo:
```
100  TRON
110  A = A/123
120  TROFF
```
Sempre que a linha 11Ø for executada um (11Ø) aparecerá no vídeo.

CSAVE

Este comando grava na fita K7 o programa que está na memória.
O gravador deve estar devidamente conectado. com a fita e os botões **PLAY e RECORD** apertados simultaneamente.
Você deve especificar um nome para o programa a ser gravado.

Exemplo:

CSAVE "A"
Grava-se na fita o programa que está na memória com o nome A.

CLOAD

Este comando permite que você carregue o computador com programas em Basic da fita cassete. Para carregar um programa da fita para o computador faça o seguinte :
1 - Coloque a fita na posição desejada usando o contador digital do gravador.
2 - Coloque o gravador em PLAY.
3 - Ajuste o volume e a tonalidade.
4 - Escreva **CLOAD** e aperte RETURN

Então, o computador ligará o gravador e carregará o primeiro programa que encontrar.
Enquanto estiver carregando, dois asteriscos aparecerão no vídeo e o da direita piscará para indicar que algo está sendo carregado.
Se este asterisco da direita não piscar, volte a fita e tente novamente com um novo ajuste de volume e/ou tonalidade

Você pode também especificar o nome do programa a ser carregado. Por exemplo, **CLOAD "A"** fará com que o computado ignore todos os programas que tem nome diferente de A até encontrar um programa cujo nome é A.
Enquanto o computador procura pelo programa escolhido, os nomes dos programas que ele encontrar aparecerão no vídeo juntamente com um asterisco.

CLOAD?

Permite que você compare o programa que está gravado na fita com o programa que está armazenado na memória do computador. Este comando é muito útil para lhe dar certeza que um programa foi gravado perfeitamente.
Durante **CLOAD?**, o programa da fita é comparado com o programa da memória byte por byte.
Se houver alguma diferença, a mensagem **NÃO** aparecerá no vídeo.

PROTEGER?

Quando se liga o computador esta mensagem aparece no vídeo.
Esta é a oportunidade que se tem para reservar um espaço na memória para armazenamento de programas em linguagem de máquina.
Com este comando você protege uma área da memória de ser usada pelo Basic.
Quando você fornece um endereço, você está protegendo a memória a partir daquele endereço até o fim da memória disponível.
Por exemplo, se você responde 25000 você está protegendo da memória 25000 até o fim.
Você deve colocar um número maior que 17430.

SYSTEM

Este comando permite que você carregue o computador com programas em linguagem de máquina da fita cassete. Para carregar um programa em linguagem de máquina, faça o seguinte :

1 - Escreva **SYSTEM** e aperte RETURN .

2 - Um * ? aparecerá no vídeo. Coloque ,então. o nome do programa a ser carregado e aperte RETURN .

3 - Após a leitura do programa um outro * ? aparecerá no vídeo. Coloque um / seguido do endereço de entrada do programa. Ou então, coloque um / e aperte RETURN . Neste caso a execução do programa se iniciará no endereço especificado pelo programa objeto.

Se algum erro ocorrer durante a leitura da fita um C aparecerá junto ao asterisco da direita.

USR

Este é o comando que lhe permite unir o Basic com a linguagem de máquina. Do Basic você chama uma rotina em linguagem de máquina com a instrução **USR**. Da linguagem de máquina você retorna ao Basic com a instrução **RET**.

Para se fazer um programa em linguagem de máquina é necessário o conhecimento de Assembly e familiaridade com as instruções do microprocessador Z-80.

A primeira coisa que se deve fazer, quando se vai programar em linguagem de máquina, é proteger um espaço na memória RAM para armazenamento de seu programa. Se você não proteger um espaço na memória, o Basic pode "mexer" no espaço onde seu programa está armazenado e destrui-lo.
Você protege a memória quando você liga o computador e a mensagem **PROTEGER?** aparece no vídeo. Você deve proteger a partir do endereço que precede o início de seu programa em linguagem de máquina.
Depois você deve carregar o programa. Você tem duas opções para carregar o programa em linguagem de máquina : com o comando **SYSTEM**, se a fita foi criada com o programa DIGBUG ou então você pode usar o comando POKE para colocá-lo na memória.
Além disto você deve dizer ao Basic o endereço onde seu programa em linguagem de máquina se inicia.
Para isto você deve colocar o endereço nas memórias 16526 e 16527.
O byte menos significativo do endereço na memória 16526 e o byte mais significativo do endereço na memória 16527.

Exemplo:

Suponhamos que seu programa se inicia na memória 31300.
Quando você liga o computador responda **PROTEGER?** com 31299.
31300 é 7A44 em hexadecimal.
O byte menos significativo do endereço é 44 em hexadecimal, que é 68 em decimal. Portanto faça: POKE 16526,68. O byte mais significativo do endereço é 7A em hexadecimal, que é 122 em decimal. Portanto faça: **POKE** 16527,122.
Estas duas instruções **POKE** dizem ao Basic que seu programa em linguagem de máquina se inicia na memória 31300.

No Basic para você chamar o programa em linguagem de máquina use uma instrução do tipo A = USR (B) onde B é um argumento qualquer. Quando o Basic encontra a instrução A = USR (B), a execução do programa se iniciará no endereço que está armazenado nas memórias 16526 e 16527.

Para retornar ao Basic use a instrução **RET** do Assembly.

Para passar um dado para a linguagem de máquina, inclua a seguinte instrução no início de seu programa em linguagem de máquina:

CALL ØA7FH

Isto carrega o argumento B no registrador HL com um número inteiro sinalizado de dois bytes.

Para retornar um dado da linguagem de máquina para o Basic, carregue o registrador HL com o dado (inteiro de dois bytes com sinal) e coloque a seguinte instrução no fim de sua rotina:

JP ØA9AH

O controle passará para o Basic e o registrador HL substituirá **USR(B)**.
Por exemplo, se você chamou a rotina em linguagem de máquina com:

A=USR(B)

Então a variável A terá o valor do dado que estava armazenado no registrador HL.

CLEAR

Se este comando for usado com um argumento, por exemplo, CLEAR 300, esta instrução faz com que o computador reserve n bytes (no exemplo 300) para armazenamento de strings. Além disto, todas as variáveis são zeradas.

Exemplo:

```
10 CLEAR 1000
```

Reserva 1000 bytes de memória para armazenamento de string.

Se você reservar a quantidade exata de memória para string, o seu programa pode fazer um uso mais eficiente da memória.
Quando se liga o computador um **CLEAR 50** é executado automaticamente.

VARPTR

Este comando fornece o endereço de onde está armazenada uma variável. O dado é sempre armazenado usando notação de complemento de dois.
Se você usar **VARPTR** (uma variável inteira) o computador fornecerá um endereço X tal que: o conteúdo do endereço X, simbolicamente (X), é o byte menos significativo do número inteiro. (X+1), ou seja, o endereço imediatamente superior, contém o byte mais significativo do número inteiro (Ø= o número é positivo; 1= o número é negativo).

Se for **VARPTR** (variável de precisão simples), o computador fornece um endereço X tal que:
(X) = byte menos significativo.
(X+1) = próximo byte mais significativo.
(X+2) = byte mais significativo. O bit mais significativo é o sinal do número.
(X+3) = expoente do número.

Se for **VARPTR** (variável de precisão dupla):
(K) = byte menos significativo.
(K+1)...(K+5) = próximo byte mais significativo.
(K+6) = byte mais significativo, com o bit mais significativo sendo o sinal.
(K+7) = expoente do número.

Exemplo:

Você pode verificar estes valores usando o comando **PEEK(X)** , sendo X o endereço de onde você quer verificar o conteúdo.

```
10 INPUT "Coloque um numero inteiro";A%
20 EN = VARPTR (A%)
30 PRINT "O byte menos significativo e";PEEK(EN)
40 PRINT "O byte mais significativo e";PEEK(EN+1)
50 PRINT "O numero e";PEEK(EN+1)*256+PEEK(EN)
```

A multiplicação por 256 é para deslocar o número de 8 casas binárias, já que ele é o byte mais significativo.

Números de precisão simples ou dupla são armazenados na forma exponencial normalizada. Então 128 é somado ao expoente. O bit mais significativo do byte mais significativo é usado como sinal.

Exemplo:
A! = 2 será armazenado da seguinte forma:
2 é 10 em binário que normalizado é .1 x 2!2. Portanto o exponente será 128 + 2 = 130. O número a ser armazenado é:

.10000000 00000000 00000000
byte + significativo ..... byte menos significativo

Portanto o byte mais significativo é: 10000000
No entanto, o bit mais significativo é trocado por zero, por ser um número positivo. Logo, A! é armazenado desta maneira:

Expoente (X+3)	(X+2)	(X+1)	(X)
130	0	0	0

A! = -5
5 = 101 = .101 x 2!3
Expoente : 128 + 3 = 131
Byte mais significativo : 10100000 = 160
assim,

(X+3)	(X+2)	(X+1)	(X)
131	160	0	0

A! = -19
19 = 10011 = .10011 x 2!5
Expoente : 128 + 5 = 133
Byte mais significativo : 10011000 = 152
Assim,

(X+3)	(X+2)	(X+1)	(X)
133	152	0	0

Se for **VARPTR** (de uma string) teremos :
(X) = comprimento da string
(X+1) = byte menos significativo do endereço onde se inicia a string
(X+2) = byte mais significativo do endereço onde se inicia a string

Para todos os tipos de variáveis acima relacionados temos :
(X-1) e (X-2) armazenará o nome da variável,
(X-3) conterá o código do tipo de variável. Para números inteiros este código é 02; para números com precisão simples é 04; dupla precisão é 08 e string é 03.

Se for **VARPTR** (variável de um array) o computador fornecerá o endereço para o primeiro byte daquele elemento no array. O elemento consistirá de 2 bytes se for um array de números inteiros; 4 bytes se for um array de números de precisão simples; 8 bytes se for um array de números de precisão dupla; 3 bytes se for string. O primeiro elemento no array é precidido por:

1 - Uma sequência de dois bytes por dimensão, cada par de 2 bytes indicando o tamanho de cada dimensão.
2 - Um byte que indica o número de dimensões.
3 - Dois bytes indicando o número total de elementos no array.
4 - Dois bytes indicando o nome do array.
5 - Um byte que indica o tipo do array (02 = inteiro; 04 = precisão simples; 08 = precisão dupla ; 03 = string).

ERROR

Este comando permite a simulação de um erro durante a execução de um programa.
A maior utilização desta instrução é para testes de rotinas que são atingidos pela instrução **ON ERROR GOTO.**

Exemplo:

```
10 ERROR 2
```

Rode este programa e no vídeo aparecerá **SN ERRO** pois 2 é código para **SN ERRO.**

ON ERROR GOTO

Normalmente, se um erro acontece durante a execução de um programa a execução do programa é interrompido.
Mas se você prever um certo tipo de erro, você pode fazer uma rotina que evita a interrupção do programa.

Exemplo:

```
10 ON ERROR GOTO 100
20 INPUT " Coloque um numero qualquer"; N
30 PRINT "O inverso deste numero e"; 1/N
40 GOTO 20
100 PRINT "Infinito"
110 RESUME 20
```

Note que a rotina que manipula o erro deve terminar com a instrução **RESUME.**

RESUME

Termina a rotina de manipulação do erro, especificando a linha onde a execução do programa deve continuar.
No exemplo acima, após o erro, a execução do programa continua na linha 20.

Exemplo:

```
10 ON ERROR GOTO 100
20 READ A
30 DATA 10,12
40 PRINT A;
50 GOTO 20
100 RESTORE
110 RESUME
```

RESUME sem o número da linha indica que o computador deve recomeçar na linha que o erro aconteceu.

```
10 ON ERROR GOTO 100
20 X = RND (5) -1
30 N = 1/X
40 GOTO 20
100 N = 0
110 RESUME NEXT
```

RESUME NEXT indica que o computador deve recomeçar na linha imediatamente após onde ocorreu o erro.
Neste exemplo, toda vez que X for zero, N é considerado zero.

ERL

Fornece o número da linha na qual um erro ocorreu.

Exemplo:

```
10 ON ERROR GOTO 100
20 INPUT "Coloque um numero qualquer";N
30 PRINT "O inverso deste numero e";1/N
40 PRINT "A raiz quadrada deste número e" SQR (N)
50 GOTO 20
100 IF ERL = 30 THEN 120  ELSE IF ERL = 40 THEN 140
110 PRINT "O erro aconteceu fora da linha 30 e 40" : END
120 PRINT "Infinito"
130 RESUME 40
140 PRINT "Imaginaria = ";
150 N = -N
160 PRINT SQR (N)
170 RESUME 50
```

Rode o programa e coloque diversos números, inclusive zero e números negativos.

ERR

Este comando é similar ao **ERL**, exceto que **ERR** fornece um valor que está relacionado com o código do erro, através da seguinte fórmula :

ERR/2+1 = código de erro

Exemplo:

No exemplo anterior troque alinha 1ØØ e 11Ø por:

```
1ØØ IF ERR/2+1 = 11 THEN 12Ø ELSE IF ERR/2+1 = 5 THEN 14Ø
11Ø PRINT "Erro inesperado" : END
```

Note que este programa alerta para qualquer tipo de erro, mesmo que aconteça um outro tipo de erro na linha 3Ø ou 4Ø. Por exemplo, um **SN ERRO** na linha 3Ø não será interpretado como um **/Ø ERROR**, como acontece no exemplo anterior.

PRINT @

Se você deseja mandar um ítem para o vídeo, numa localização determinada, use o comando **PRINT @.**
Com o comando **PRINT** você escreve no vídeo na posição em que o cursor se encontra. Com **PRINT @** você escreve em qualquer localização desejada; independente da posição do cursor.
O vídeo é dividido em 1Ø24 posições (de Ø até 1Ø23). **PRINT @** requer dois argumentos: um é a posição (deve ser um número entre zero e 1Ø23) e o outro é o que se deseja mandar para o vídeo.

Exemplo:

```
PRINT @ 48Ø, "n"
```

Este comando escreverá a letra n no meio do vídeo.

Um line-feed automático é executado sempre que se usar um **PRINT @** na última linha do vídeo. Para evitar isto, use um ponto e vírgula no fim do comando.

Exemplo:

```
1Ø PRINT @ 1Ø2Ø, "U";
```

Rode o programa abaixo e veja o resultado:

```
1Ø CLS
2Ø FOR P = 1 TO 1ØØ5
3Ø PRINT @ P, "COMPUTADOR DGT-1ØØ";
4Ø PRINT @ P - 1, " ";
5Ø NEXT
1ØØ GOTO 1ØØ
```

PRINT TAB

Este comando faz a tabulação do vídeo. Geralmente usa-se **PRINT TAB** para tabelas. **PRINT TAB** não move o cursor para a esquerda.

Exemplo:

```
10 PRINT TAB (5) "NOME"; TAB (20) "PROFISSAO"; TAB (35) "IDADE"
```

PRINT USING

Quando se deseja um formato especial, usa-se **PRINT USING.**
PRINT USING requer 2 argumentos: uma string e um valor, separados por ponto e vírgula.
Este comando imprime o valor, usando a especificação de formato contida na string. O valor pode ser um número ou uma string. As especificações de formato a serem usadas são:

Para números usamos # e para string usamos %...%.

- este sinal especifica a posição de cada digito do valor numérico.
O número de # que você usa ,define o campo numérico.

Exemplo:

PRINT USING "##.##"; 12.347 RETURN
12.35
O número de # depois do ponto, determina o tamanho da parte decimal.

PRINT USING "##.##"; 123.45 RETURN
% 123.45
Neste caso apareceu um % para indicar que o campo definido pela string e menor que o campo necessário para imprimir o valor numerico.

PRINT USING "###.###"; 12.3 RETURN
12.300
Se o campo especificado for maior que o campo necessário para imprimir o valor, nas posições a esquerda aparecem espaços e as posições a direita são completadas com zeros.

PRINT USING "#.##"; 5.678 RETURN
5.68
Note que há arredondamento.

PRINT USING "###,###"; 45678.12
45,678
Uma vírgula na especificação do campo, separa os milhares. Note que não foi especificado campo para imprimir decimais.

PRINT USING "###.###"; 12.34** RETURN.
**12.340
Dois asteriscos colocados no início da especificação do campo faz com que todos os lugares não usados, a esquerda do ponto, sejam completados com asteriscos. Os dois asteriscos estabelecem duas posições a mais.

PRINT USING "$$###.##"; 34.56
$34.56
Dois cifrões colocados no início da especificação do campo, faz aparecer um cifrão antes do número.

PRINT USING "$###.###"; 34.56**
***$ 34.560
Esta é uma combinação dos dois últimos especificadores.

PRINT USING "##.###↑↑↑↑"; 12,34
1.234 E + 01
↑↑↑↑ faz o número ser impresso em formato exponecial.

PRINT USING "+##.##"; 12.34
+12.34
Quando um sinal + é colocado no início de uma especificação, aparecerá um + para números positivos e um - para números negativos.

PRINT USING "-##.##"; -12.34
-12.34
Um sinal - faz com que apareça um - na frente de números negativos. Se o número for positivo, um espaço é imprimido.

Outra utilidade de **PRINT USING** é na especificação de campos para strings. O campo para string é especificado por % ... %. O comprimento do campo será 2 mais o número de espaços entre os dois % .

Exemplos:
PRINT USING "% %"; COMPUTADOR" RETURN (há 2 espaços entre os %)
COMP

PRINT USING ""; "COMPUTADOR" RETURN
CO

PRINT USING "↑"; "COMPUTADOR" RETURN
C
Se você deseja que apenas a primeira letra da string seja imprimida, então use ↑ .

PRINT USING "↑ ↑ !"; "MICRO"; "COMPUTADOR"; "DGT-100"
M C D
Usando-se mais de um ↑ , a primeira letra de cada string será imprimida com espaços correspondendo ao número de espaços entre os ↑ . Note, no exemplo, o espaço entre as letras M C e D, que corresponde ao espaço colocado entre os ↑ .

PRINT # -1 e INPUT # -1

Através destes dois comandos nós podemos guardar no cassete grandes quantidades de dados e depois recuperá-los para processamento, da mesma maneira que nós gravamos e carregamos os programas através dos comandos **CLOAD e CSAVE.**
Para dados nós temos os comandos **PRINT # -1 e INPUT # -1.**
Coloque o gravador para gravar e rode este pequeno programa:

```
10 A$ = "MICROCOMPUTADOR" : B = 1981 : C = 3.14159
20 PRINT # -1, A$, B, C
```

Este programa grava na fita cassete os valores de A$, de B e de C. Desta maneira você tem uma gravação permanente destas 3 variáveis. Estes valores podem ser lidos pelo computador através do comando **INPUT # -1.** Para ler estes valores, volte a fita, coloque o gravador em **PLAY** e rode o seguinte programa:

```
10 A$ = "" : B = 0 : C = 0
20 PRINT "Antes da leitura os valores sao : "; A$, B, C
30 INPUT # -1, A$, B, C
40 PRINT "Apos a leitura os valores sao : "; A$, B, C
```

Veja como o computador liga o gravador, lê os valores e depois desliga o gravador.

Na linha 10 deste exemplo nos zeramos todas as variáveis. Se os valores não forem lidos corretamente, estas variáveis terão o valor zero.
A linha 20 imprime os valores antes da leitura.
A linha 30 lê os valores armazenados na fita.
A linha 40 imprime os valores após a leitura.

OBSERVAÇÕES IMPORTANTES

1) A instrução **INPUT # -1** deve ser idêntica a instrução **PRINT # -1** que criou os dados em termos do número e tipo dos ítens.
Por exemplo, se uma string é lida quando o computador está esperando um número, um **AI ERRO** aparecerá no vídeo. Se não houver dados suficientes um **FD ERRO** aparecerá no vídeo.

2) **PRINT#-1 e INPUT#-1** não podem ser usados no modo de comando, isto é, estas duas instruções sempre devem ser parte de um programa.

3) Para guardar e acessar dados no segundo gravador use **PRINT#-2 e INPUT#-2.**

Exemplo:

Vamos criar um arquivo com os seguintes campos: NOME, PROFISSÃO E IDADE. O nosso arquivo contará com os seguintes dados :

NOME	PROFISSÃO	IDADE
Paulo	Engenheiro	37
Suely	Jornalista	28
Maria Isabel	Médica	3Ø
Roberto	Programador	25

Então o nosso programa para construir o arquivo poderá ser assim:

```
1Ø READ N$, P$, I
2Ø IF N$ = "Fim" THEN END
3Ø PRINT # -1, N$, P$, I
4Ø GOTO 1Ø
5Ø DATA Paulo, Engenheiro, 37, Suely, Jornalista, 28, Maria Isabel, Medica, 3Ø, Roberto, Programador, 25, Fim, Fim,Ø
```

Coloque o gravador para gravar e rode o programa. Os três últimos dados servem apenas para indicar o fim do arquivo.

Agora podemos até desligar o computador, que o nosso arquivo está seguramente guardado na fita.

Vamos fazer um programa para recuperar os dados guardados no cassete :

```
1Ø INPUT # -1, A$, B$, C
2Ø IF A$ = "Fim" THEN END
3Ø PRINT A$, B$, C
4Ø GOTO 1Ø
```

NOTE:

1) Os dados foram lidos 3 de cada vez, para ser compatível com a instrução **PRINT # -1**, que os criou.

2) Não é necessário usar os mesmos nomes para ler os dados pois o nome da variável não é gravado. Apenas o valor é gravado.

Se você quiser aumentar o arquivo basta colocar mais instruções DATA no fim do programa. Não se esqueça que os 3 últimos ítens do último DATA deve ser Fim, Fim, Ø.

CAPÍTULO 3

STRINGS

A maior vantagem de um microcomputador sobre uma calculadora é a sua capacidade de manipulação com letras.
String é uma variável que pode conter números ou letras, com um comprimento máximo de 255 caracteres. Para declarar que uma variável é string, você deve acrescentar o símbolo $ ao nome da variável.

COMPARAÇÃO DE STRINGS

Você pode comparar strings, para, por exemplo, colocá-las em ordem alfabética.
Quando as strings são comparadas, todos os caracteres devem ser iguais (incluindo espaços) .

Exemplo:

```
1Ø A$ = "COMPUTADOR"
2Ø IF A$ = "COMPUTADOR" THEN PRINT "Teste Positivo"
```

Se você coloca um espaço antes ou depois da palavra COMPUTADOR na linha 2Ø, o teste falha.

As strings são comparadas caracter por caracter da esquerda para a direita.
Há ainda outros tipos de comparação que se pode fazer com strings: maior (">"); menor ("<"); maior ou igual (">="); menor ou igual ("<="); diferente ("<>").

Exemplo:

```
1Ø INPUT "Escreva COMPUTADOR"; A$
2Ø IF A$ <> "COMPUTADOR" PRINT "Você não escreveu certo"
```

CONCATENAÇÃO DE STRINGS

Você pode juntar duas ou mais strings em uma só, usando o símbolo + .

Exemplo:

```
1Ø A$ = "PRIMEIRO"
2Ø B$ = "SEGUNDO"
3Ø C$ = A$ + B$
4Ø PRINT A$, B$, C$
```

ASC (string)

Esta função fornece o código do primeiro caracter da string. A string argumento deve estar entre parenteses.

Exemplo:
```
10 PRINT ASC ("E")
20 A$ = "EXEMPLO"
30 PRINT ASC (A$)
```

CHR$ (expressão)

É o inverso de **ASC.** Ou seja, imprime um caracter cujo o código está entre parenteses.

Exemplo:
```
10 PRINT CHR$ (65)
20 N = 64
30 PRINT CHR$ (N + 1)
```
As linhas 10 e 30 imprimem o mesmo caracter.

É possível imprimir aspas, com o comando **CHR$** (34). Veja o anexo 1, onde se tem todos os códigos usados. Os códigos de 0 a 31 são códigos de controle. Por isso, quando se usa estes códigos com **CHR$**, você obtem a função efetuada por aquele código de controle.
Por exemplo se você executar um **PRINT CHR$** (23) você passa para o formato de vídeo com 32 caracteres por linha.

STR$ (expressão)

Converte uma expressão numérica em string. A expressão numérica deve estar entre parenteses.

Exemplo:
```
10 A$ = STR (3.14)
```

Após a execução desta instrução, a string A$ tem 5 caracteres, da seguinte forma: " 3.14" - um espaço, o caracter 3, o caracter ponto, o caracter 1 e o caracter 4. O espaço é colocado para permitir a colocação do sinal.

```
10 P = -3.145
20 B$ = STR$ (P)
```

A linha 20 atribui à B$ um string com seis caracteres desta forma: "-3.145" .

NOTA: No exemplo acima você pode efetuar operações aritméticas com a variável P, mas apenas manipulações de strings com a variável B$.

VAL (string)

É o inverso da função **STR$** ou seja, fornece um número representado pelos caracteres da string argumento.

Exemplo:

```
PRINT VAL ("152 Cruzeiros")   RETURN
152
```

Apenas os caracteres numéricos são considerados.

```
10 A$ = 15
20 B$ = 16
30 C = VAL (A$ + "." + B$)
```

Neste exemplo C terá o valor de 15.16

```
10 A$ = 15
20 B$ = 16
30 C = VAL (A$ + "E" + B$)
C tera o valor de 15  E 16
```

LEN (string)

Fornece o comprimento da string.

Exemplo:

```
10 A$ = "Computador"
20 PRINT " A palavra "; A$; " tem "; LEN (A$); " caracteres"
```

FRE

Fornece a quantidade de memória disponível para armazenamento de strings.
Um argumento qualquer deve ser incluido entre parênteses.

Exemplo:

```
PRINT FRE (U$)
```

LEFT$ (string,n)

Fornece os n primeiros caracteres da string.

Exemplo:

```
PRINT LEFT$ ("COMPUTADOR", 3) RETURN
```

RIGHT$ (string,n)

Fornece os n últimos caracteres da string.

Exemplo:

```
PRINT RIGTH$ ("COMPUTADOR",3)   RETURN
DOR
```

MID$ (string,c,n)

Fornece uma substring de comprimento n e começando na posição c da string.

Exemplo:

```
10 A$ = "Exemplo"
20 B$ = MID$ (A$,3,3)
```

Após a execução deste programa, B$ será igual "emp".

INKEY$

Esta função faz uma varredura no teclado e fornece uma string de um caracter, que é o caracter que foi teclado antes da varredura. Se nenhum caracter for teclado, a string terá o valor nulo. Devido a rapidez com que a função **INKEY$** é executada, ela é colocada geralmente dentro de um loop para que o teclado seja varrido diversas vezes.
Esta função é muito útil quando se deseja colocar os valores sem precisar de apertar a tecla RETURN.

Exemplo:

```
10 PRINT "Aperte uma letra"
20 A$ = INKEY$
30 IF A$ = "" THEN 20 ELSE PRINT "Voce apertou  a tecla "; A$
40 GOTO 10
```

STRING$ (n,caracter)

Fornece uma string composta de n caracteres.

Exemplo:

```
10 A$ = STRING$ (155,"A") ou 10 A$ = STRING$ (155,65)
A$ será uma string composta de 155 letras A

10 A$ = STRING$ (200,170)
A$ será uma string composta de 200 blocos gráficos cujo código é
170.
```

RESERVA DE MEMÓRIA

PARA ARMAZENAMENTO DE STRINGS

Quando você liga o computador 50 bytes de memória são reservados automáticamente para armazenamento de string. Para reservar mais ou menos que 50 bytes você deve usar CLEAR seguido do número desejado.

CAPÍTULO 4

EDIÇÃO

Você passa para o modo de edição, através do comando EDIT seguido do número de uma linha.
Este modo permite que se faça correções e modificações em linhas de programas, com relativa facilidade. Como exemplo de todos os comandos vamos usar a seguinte linha de programa:

```
20 FOR R = 0 TO 50 : PRINT "Exemplo" : NEXT
```

Portanto bata esta linha para acompanhar os exemplos.

EDIT

Este é o comando que coloca o computador no modo de edição. Você tem que especificar a linha que deseja editar.

Exemplo:

Escreva EDIT 20 e aperte RETURN. Então aparecerá :

```
20 -
```

Você está então no modo de edição e pode começar a editar a linha 20.

TECLA RETURN

Você termina a edição de uma linha quando a tecla RETURN é apertada.

TECLA DE ESPAÇO

Se você apertar a tecla de espaço, o cursor se moverá para a direita.

Exemplo:

Estando no modo de edição o vídeo apresenta:

```
20 -
```

Aperte a tecla de espaço uma vez e o vídeo apresenta:

```
20 F -
```

Apertando mais uma vez:

```
20 FO -
```

Portanto este comando permite que você alcance o lugar desejado para fazer a edição.
Se você desejar andar 7 posições para fazer uma edição você pode apertar a tecla de espaço 7 vezes ou então apertar a tecla 7 e a tecla de espaço.

TECLA <—

É' semelhante a tecla de espaço, só que o movimento é da direita para a esquerda, isto é , para trás.

TECLA L

Quando você aperta a tecla L o resto da linha, que está sendo editada, é listada no vídeo. O cursor executa um line-feed e o número da linha aparece novamente. Você continua no modo de edição.

Exemplo:

```
EDIT 20   RETURN
20 -
```

Aperte L (não é necessário apertar RETURN)

```
20 FOR R = 0 TO 50 : PRINT "Exemplo" : NEXT
20 -
```

TECLA I

Este comando permite inserir caracteres. Você sempre coloca caracteres onde o cursor se localiza.
Se você aperta a tecla <--- você deleta caracteres ao invés de inserí-los.

Exemplo:

Suponhamos que você deseja inserir o caracter 1 após a palavra "Exemplo" da linha 20.

```
EDIT 20   RETURN
20
```

Agora aperte a tecla de espaço quantas vezes forem necessárias para se alcançar a letra 'o' da palavra Exemplo. O vídeo mostrará :

```
20 FOR R = 0 TO 50 : PRINT "Exemplo-
```

Aperte a tecla I para entrar no modo de inserção. Você pode então colocar os caracteres desejados, no caso, o 1.

```
20 FOR R = 0 TO 50 : PRINT "Exemplo 1 -
```

Para você sair do modo de inserção veja o comando abaixo.

TECLA SHIFT—↑

Aperte SHIFT ↑ para sair de um dos seguintes modos: I, X ou H.
Após apertar SHIFT ↑ você ainda está no modo de edição e você pode, por exemplo, inserir caracteres em outras posições da linha.
No exemplo acima, após apertar SHIFT ↑ aperte a tecla L para listar a linha inteira. Se tudo está conforme você deseja, aperte a tecla RETURN para sair do modo de edição e voltar ao modo de comando.

TECLA D

Esta tecla deleta o caracter que está a direita do cursor. Os caracteres que forem deletados serão incluidos entre pontos de exclamação.

Exemplo:

Na linha 20 que estamos usando, suponhamos que você deseja trocar a palavra "Exemplo" pela palavra "Teste" .

EDIT 20 RETURN
20 -

Aperte a tecla de espaço até alcançar a primeira aspas :
20 FOR R = 0 TO 50 : PRINT "-

Aperte a tecla D sete vezes (ou então 7D).
Se você apertou 7D o vídeo mostrará :
20 FOR R = 0TO 50 : PRINT "!Exemplo! -

Agora você pode usar o comando I para inserir a palavra "Teste"

TECLA A

Esta tecla cancela todas as modificações feitas na linha, enquanto você está no modo de edição.
Por exemplo, se você deletou algum caracter e depois desistiu, aperte a tecla A para reiniciar a edição da linha.

NOTA - A tecla A só funciona enquanto você esta' editando a linha. Isto é , se você deletou algum caracter e apertou RETURN a modificação é irrecuperável.
Por isso é sempre aconselhável usar o comando L, para verificar se a linha está conforme desejado, antes de terminar a edição com a tecla RETURN .

- Lembre-se que você deve sair de qualquer subcomando (através de SHIFT ↑) para poder usar A.

TECLA X

Este comando move o cursor para o fim da linha e entra automaticamente no modo de inserção.

Exemplos:

Suponha que na linha 20 você prentenda acrescentar uma nova instrução.

EDIT 20
20 -
Aperte a tecla X
20 FOR R = 0 TO 50 : PRINT "Exemplo 1" : NEXT -
Você está no modo de inserção e pode colocar a instrução que deseja. Por exemplo, escreva: PRINT "FIM" no fim da linha. Agora saia do comando X (através de SHIFT ↑) e aperte L, para verificar a linha.

TECLA H

Quando você aperta esta tecla você deleta tudo que está a direita do cursor e entra automaticamente no modo de inserção.

TECLA C

Este comando permite que você troque caracteres começando na posição do cursor.
Se você aperta C sem um número precedente, o computador permite que você troque o caracter que está na posição do cursor.
Se você aperta C precedido de um número, o computador permite que você troque tantos caracteres quantos os especificados pelo número que precede C.
Depois que você trocou os caracteres, você retorna automaticamente para o modo de edição, isto é , não é necessário apertar SHIFT ↑ para sair deste comando.

Exemplos:

Suponha que no nosso exemplo você deseja trocar o 50 por 63.

```
EDIT 20
20 -
```

Aperte a tecla de espaço até o caracter que precede o 5

```
20 FOR R = 0 TO -
```

Agora aperte 2C, para dizer ao computador que você deseja trocar 2 caracteres. Coloque então, o número 63

```
20 FOR R = 0 TO 63 -
```

Quando você apertou 2C o computador deleta 2 caracteres e fica esperando 2 novos caracteres. Note que o número de caracteres deletados é sempre igual ao número de caracteres inseridos.
A tecla '<---' não funciona com o comando C. Portanto ela não deve ser usada. Se você cometeu um erro durante o comando C, edite a linha novamente para corrigí-la.

TECLA E

Termina a edição e mantém todas as alterações feitas. Antes de usar a tecla E você deve sair de qualquer outro comando através de SHIFT ↑ .

TECLA Q

Termina a edição e cancela todas as alterações feitas. Antes de usar a tecla Q você deve sair de qualquer outro comando através de SHIFT ↑ .

TECLA S

Formato para este comando: nSc
Este comando move o cursor até a n-esima ocorrência do caracter c.

Exemplo:

3SP - o computador move o cursor até a terceira ocorrência do caracter P
2S: - o computador move o cursor até a segunda ocorrência do caracter :
1SA ou SA - O computador move o cursor até a primeira ocorrência do caracter A. Note que, se você deseja a primeira ocorrência, o 1 é opcional.

Após você mover o cursor até o ponto desejado você pode executar qualquer outro subcomando.

TECLA K

Formato para esta comando: nKc
Este comando diz ao computador para deletar a partir do cursor até a n-esima ocorrência do caracter c, exclusive.

Exemplo:

2KP - o computador deleta tudo que está entre o cursor e a segunda ocorrência do caracter P
1K: ou K: - o computador deleta tudo que está entre o cursor e a primeira ocorrência do caracter :. Note que o 1 é opcional.

Supondo que a linha 20 esteja desta maneira:

```
20 FOR R = 0 TO 63 : PRINT "Exemplo 1" : NEXT : PRINT "Fim".
```

Se você deseja tirar o loop desta linha faça:

```
EDIT 20
20 -
```

Aperte 2KP

```
20 ! FOR R = 0 TO 63 : PRINT "Exemplo" : NEXT : ! PRINT "Fim"
```

Aperte L

```
20 PRINT "Fim"
```

NOTA - A letra P maiúscula é diferente da letra p minúscula. Por isso usamos 2KP. Se a palavra Exemplo estiver escrita em maiúsculas devemos usar 3KP.

CAPÍTULO 5

MENSAGENS DE ERRO

O computador fornece uma série de mensagens de erro, que lhe ajuda bastante a encontrar erros em programas.

Estas mensagens são:

NF - NEXT without FOR - Código 1
Quando se usa um NEXT sem um FOR correspondente.

SN - Syntaxe Erro - Código 2
Geralmente ocorre quando se faz um erro de pontuação, comandos escritos erradamente ou parenteses abertos.

RG - RETURN without GOSUB - Código 3
Quando o computador encontra um RETURN sem que um GOSUB fosse executado anteriormente.

FD - Fim de Dados - Código 4
Quando um READ ou INPUT # é executado e não há dados suficientes.

PI - Paramétro Incompatível - Código 5
Quando executa-se uma operação usando paramétros ilegais como, por exemplo, raiz quadrada de número negativo.
Este erro também ocorre se você usa a instrução USR sem antes dizer ao Basic o início do programa em linguagem de máquina.

OV - Overflow - Código 6
Quando o número usado ou obtido é muito grande.

FM - Fim da Memória - Código 7
Quando toda a memória dispónível já foi usada ou reservada.

LI - Linha Indefinida - Código 8
Quando há uma referência a uma linha que não existe.

IA - Índice além do dimensionado - Código 9
Quando se refere a um elemento de uma matriz (array) com o índice maior do que o dimensionado.

AR - Arranjo redimensionado - Código 10
Quando se redimensiona uma matriz (array) que já foi dimensionada com o comando DIM

/O - Divisão por zero - Código 11
Quando se faz uma divisão por zero, ou por um número menor que +/- 1.701411 E -38

CT - Comando Direto Ilegal - Código 12
Quando se tenta executar um INPUT, INPUT# ou PRINT# no modo de comando.

VI - Variável incompatível - Código 13
Quando se atribui uma string a uma variável numérica e vice-versa.

OS - Overflow de espaço para string - Código 14
A quantidade de espaço reservado para string foi excedido.

SG - String muito grande - Código 15
Quando o comprimento de uma string excede 255 caracteres.

FC - Fórmula de string muito complexa - Código 16
Quando uma operação com string é muito complexa. Se isto acontecer, divida a operação em operações intermediárias mais simples.

IC - Impossível continuar - Código 17
Quando se tenta executar um CONT, sendo que não houve uma interrupção ou então o programa terminou ou foi editado.

NR - Não há RESUME - Código 18
Quando o fim do programa é alcançado numa rotina de manipulação de erros.

RE - RESUME sem erro - Código 19
Quando a instrução RESUME é encontrada antes que um ON ERROR GOTO seja executado.

CN - Código não existente - Código 20
Quando se tenta simular um erro, através do comando ERROR, com um código não existente.

FO - Falta do operando - Código 21
Quando se tenta executar uma operação, sem que seja fornecido os operandos necessários.

AI - Arquivo incorreto - Código 22
Os dados de uma fonte externa (por exemplo, o cassete) não estão corretos ou em sequência diferente.

CD - Comando de disco - Código 23
Quando se executa um comando que só é possível em computadores que tem unidade de disco.

APÊNDICE 1

CÓDIGOS ASCII

CÓDIGO DEC.	CÓDIGO HEX.	TECLAS *	PRINT CHR$(Código)
0	00	-	Sem efeito
1	01	BREAK	Sem efeito
		CONTROL A	
2	02	CONTROL B	Sem efeito
3	03	CONTROL C	Sem efeito
4	04	CONTROL D	Sem efeito
5	05	CONTROL E	Sem efeito
6	06	CONTROL F	Sem efeito
7	07	CONTROL G	Sem efeito
8	08	<---	Move o cursor para trás e
		CONTROL H	apaga o caracter
9	09	--->	Sem efeito
		CONTROL I	
10	0A	↓	Move o cursor para o início
		CONTROL J	da próxima linha e apaga
11	0B	CONTROL K	Idem
12	0C	CONTROL L	Idem
13	0D	CONTROL M	Idem
14	0E	CONTROL N	Idem
15	0F	CONTROL O	Sem efeito
16	10	CONTROL P	Sem efeito
17	11	CONTROL Q	Sem efeito
18	12	CONTROL R	Sem efeito
19	13	CONTROL S	Sem efeito
20	14	CONTROL T	Sem efeito
21	15	CONTROL U	Sem efeito
22	16	CONTROL V	Sem efeito
23	17	CONTROL W	Muda para 32 caracteres/linha
24	18	SHIFT <---	Move o cursor para trás sem
		CONTROL X	apagar o caracter
25	19	SHIFT --->	Avança o cursor
		CONTROL Y	
26	1A	CONTROL Z	Move o cursor para baixo
27	1B	SHIFT ↑	Move o cursor para cima
28	1C	-	Coloca o cursor no canto superior esquerdo e passa para 64 car/linha
29	1D	-	Move o cursor p/ início da linha
30	1E	-	Apaga até o fim da linha
31	1F	CLR	Apaga até o fim da tela
32	20	ESPAÇO	Espaço
33	21	!	!
34	22	"	"
35	23	#	#
36	24	$	$
37	25	%	%
38	26	&	&
39	27	'	'

40	28	(	(
41	29	)	)
42	2A	*	*
43	2B	+	+
44	2C	,	,
45	2D	-	-
46	2E	.	.
47	2F	/	/
48	30	0	0
49	31	1	1
50	32	2	2
51	33	3	3
52	34	4	4
53	35	5	5
54	36	6	6
55	37	7	7
56	38	8	8
57	39	9	9
58	3A	:	:
59	3B	;	;
60	3C	<	<
61	3D	=	=
62	3E	>	>
63	3F	?	?
64	40	@	@
65	41	A	A
66	42	B	B
67	43	C	C
68	44	D	D
69	45	E	E
70	36	F	F
71	47	G	G
72	48	H	H
73	49	I	I
74	4A	J	J
75	4B	K	K
76	4C	L	L
77	4D	M	M
78	4E	N	N
79	4F	O	O
80	50	P	P
81	51	Q	Q
82	52	R	R
83	53	S	S
84	54	T	T
85	55	U	U
86	56	V	V
87	57	W	W
88	58	X	X
89	59	Y	Y
90	5A	Z	Z
91	5B	!	!
92	5C	-	!
93	5D	-	<--
94	5E	-	-->
95	5F	Ç	Ç
96	60	¢	¢
97	61	A	a
98	62	B	b
99	63	C	c

100	64	D	d
101	65	E	e
102	66	F	f
103	67	G	g
104	68	H	h
105	69	I	i
106	6A	J	j
107	6B	K	k
108	6C	L	l
109	6D	M	m
110	6E	N	n
111	6F	O	o
112	70	P	p
113	71	Q	q
114	72	R	r
115	73	S	s
116	74	T	t
117	75	U	u
118	76	V	v
119	77	W	w
120	78	X	x
121	79	Y	y
122	7A	Z	z
123	7B	-	
124	7C	-	
125	7D	-	
126	7E	-	
127	7F	Ç	ç

Os códigos de 128 a 191 são caracteres gráficos.

Os códigos de 192 a 255 são de compressão de espaço

* Algumas dessas teclas só são válidas quando usadas com <INKEY$>.
Quando em <INPUT> ou no modo de comando, podem ser ignoradas ou ter outro efeito.

APÊNDICE 2

PALAVRAS RESERVADAS

As palavras a seguir são reservadas para uso exclusivo pelo DIGBASIC e não devem ser usadas como nome de variáveis. Será obtido um SN ERRO caso sejam usadas como variáveis.

ABS
AND
ASC
ATN
AUTO
CDBL
CHR$
CINT
CLEAR
CLOCK
CLOSE
CLS
CMD
CONT
COS
CSNG
CVD
CVI
CVS
DATA
DEFDBL
DEFFN
DEFINT
DEFSNG
DEFUSR
DEFSTR
DELETE
DIM
EDIT
ELSE
END
EOF
ERL
ERR
ERROR
EXP
FIELD
FIX
FN
FOR
FORMAT
FRE
FREE
GET
GOSUB
GOTO
IF
INKEY$
INP
INPUT
INSTR
INT
KILL
LEFT$
LET
LSET
LEN
LEN
LINE
LIST
LOAD
LOC
LOF
LOG
MEM
MERGE
MID$
MKD$
MKI$
MKS$
NAME
NEW
NEXT
NOT
ON
OPEN
OR
OUT
PEEK
POINT
POKE
POS
POSN
PRINT
PUT
RANDOM
READ
REM
RENAME
RESET
RESTORE
RESUME
RETURN
RIGHT$
RND
RSET
SAVE
SET
SGN
SIN
SQR
STEP
STOP
STRING$
STR$
TAB
TAN
THEN
TIME$
TO
TROFF
TRON
USING
USR
VAL
VARPTR

APÊNDICE 3

QUANTIDADES DE MEMÓRIAS USADAS

Variáveis inteiras: 5 bytes (2 para o valor e 3 para o nome)

Variáveis de precisão simples: 7 bytes (4 para o valor e 3 para o nome)

Variáveis de precisão dupla: 11 bytes (8 para o valor e 3 para o nome)

Strings: 6 bytes no mínimo (3 para o nome, 3 para stack e pointer e 1 para cada caracter da string)

Array de variáveis: 12 bytes no mínimo (3 para o nome, 2 para o tamanho total, 1 para o número de dimensões, 2 para o tamanho de cada dimensão e 2,3,4 ou 8 (dependendo do tipo do array) para cada elemento no array)

Loop FOR-NEXT ativado: 16 bytes

GOSUB: 6 bytes

Cada nível de parenteses requer 4 bytes mais 12 bytes para cada valor temporário.

Cada linha de programação requer 5 bytes no mínimo: 2 para o número da linha, 2 para o pointer da linha e 1 de carriage return.

APÊNDICE 4

CÓDIGOS INTERNOS DO DIGBASIC

Os comandos de um programa em Basic são armazenados em códigos ao invés das próprias palavras com a finalidade de se economizar memórias.
Se você usar o comando PEEK na área de programação (início no endereço 17129 em decimal), você encontrará seu programa armazenado com os seguintes códigos:

CÓDIGO	COMANDO	CÓDIGO	COMANDO
129	FOR	130	RESET
131	SET	132	CLS
134	RANDOM	135	NEXT
136	DATA	137	INPUT
138	DIM	139	READ
140	LET	141	GOTO
142	RUN	143	IF
144	RESTORE	145	GOSUB
146	RETURN	147	REM
148	STOP	149	ELSE
150	TRON	151	TROFF
152	DEFSTR	153	DEFINT
154	DEFSNG	155	DEFDBL
157	EDIT	158	ERROR
159	RESUME	160	OUT
161	ON	173	SAVE
174	SYSTEM	177	POKE
178	PRINT	179	CONT
180	LIST	182	DELETE
183	AUTO	184	CLEAR
185	CLOAD	186	CSAVE
187	NEW	188	TAB
189	TO	191	USING
192	VARPTR	193	USR
194	ERL	195	ERR
196	STRING$	198	POINT
200	MEM	201	INKEY$
202	THEN	203	NOT
204	STEP	205	+
206	-	207	*
208	/	209	↑
210	AND	211	OR
212	>	213	=
214	<	215	SGN
216	INT	217	ABS
218	FRE	219	INP
220	POS	221	SQR

222	RND	223	LOG
224	EXP	225	COS
226	SIN	227	TAN
228	ATN	229	PEEK
239	CINT	240	CSGN
241	CDBL	242	FIX
243	LEN	244	STR$
245	VAL	246	ASC
247	CHR$	248	LEFT$
249	RIGHT$	250	MID$

APÊNDICE 5

MAPEAMENTO DE MEMÓRIA

ENDEREÇO DECIMAL	ENDEREÇO HEXA	CONTEÚDO
00000	0000	DIGBASIC e DIGBUG
14304	37E0	E/S mapeadas na memória
14336	3800	Teclado
15360	3C00	Vídeo
16384	4000	Vetores do DIGBASIC
16405	4016	DCB do teclado
16413	4020	DCB do vídeo
16421	4025	DCB da impressora
16429	402D	Reservado
16870	41EC	Buffer de E/S
17129	42E9	Texto de programa em Basic
		Variáveis
		Arrays
		Memórias livres
		Stack
		Armazenamento de strings
		Reservado com o PROTEGER?
32767	7FFF	Fim da memória (computadores com 16K)
65535	FFFF	Fim da memória (computadores com 48K)

RESUMO

CARACTERES E TECLAS ESPECIAIS

RETURN	Interpreta comando
CLEAR	Limpa o vídeo
BREAK	Termina a execução do programa
SHIFT @	Pausa na listagem e execução do programa
SHIFT -->	Converte vídeo para 32 caracteres por linha
-->	Move cursor para a próxima zona de tabulação
<--	Volta o cursor e deleta caracter
SHIFT <--	Apaga linha
↓	Line Feed
?	Abreviação para PRINT
'	Abreviação para REM
.	Linha apontada no momento. Usado com LIST,EDIT,DELETE etc
!	Declara que uma variável é de precisão simples
E	Declara que uma variável é de precisão simples (notação exponencial)
#	Declara que uma variável é de precisão dupla
D	Declara que uma variável é de precisão dupla(notação exponencial)
%	Declara que uma variável é inteira
$	Declara que uma variável é string

OPERADORES MATEMÁTICOS

+	Soma
-	Subtração
*	Multiplicação
/	Divisão
↑	Exponencial
>	Maior que
<	Menor que
=	Igual a
>= ou =<	Menor ou igual a
>= ou =>	Maior ou igual a
<> ou ><	Diferente

ORDEM DAS OPERAÇÕES

```
↑                    Exponencial
-                    Negação
* e /
+ e -
>, <, >=,etc
NOT
AND
OR
```

COMANDO	DESCRIÇÃO	EXEMPLO
ABS	Fornece valor absoluto	ABS(X/7)
AND	Operador lógico AND	952 AND 255
ASC(string)	Fornece código ASCII do primeiro caracter da string	ASC(A$) ASC(Teste)
ATN(arg)	Fornece o arco tangente do argumento	ATN(1.2)
AUTO	Numeração automática de linhas	AUTO AUTO 5,12
CDBL(ARG)	Coverte o argumento para presição dupla	CDBL(A+1/3#)
CHR$(exp)	Fornece uma string de um caracter cujo código é a exp	CHR$(65)
CINT(arg)	Converte o argumento para inteiro	CINT(B) CINT(D#+5)
CLEAR	Zera todas as variáveis	CLEAR
CLEAR n	Zera todas as variáveis e reserva n bytes para strings	CLEAR 500 CLEAR MEM/2
CLOAD	Carrega um programa em Basic da fita	CLOAD
CLOAD ?	Verifica o programa da memória com o da fita	CLOAD?
CLS	Limpa o vídeo	CLS
CONT	Continua execução de um programa após BREAK ou STOP	CONT
COS(ARG)	Fornece o coseno do arg Argumento deve ser em radiano	COS(3,14+A)
CSAVE	Guarda um programa em Basic na fita	CSAVE"Q"
CSNG(arg)	Converte o arg para precisão simples. Arg deve ser de precisão dupla	CSNG(A#)
DATA	Armazena dados a serem acessado por READ	DATA 13,5,Exemplo
DEFDBL	Define variáveis de precisão dupla	DEFDBL A DEFDBL A-H
DEFINT	Define variáveis inteiras	DEFINT A DEFINT A-H
DEFSNG	Define variáveis de precisão simples	DEFSNG A DEFSNG A-H
DEFSTR	Define variáveis strings	DEFSTR A DEFSTR A-H

DELETE	Deleta uma ou um conjunto de linhas	DELETE 10 DELETE 15-70
DIM	Dimensionamento de array	DIM A(12) DIM DS$(16,8)
EDIT	Entra no modo de edição	EDIT 10
END	Termina execução do programa e retorna ao modo de comando	END
ERL	Fornece o número da linha na qual ocorreu o erro	ERL
ERR	Fornece um valor relacionado com o código do erro	ERR/2 + 1
ERROR(cod)	Simula erro especificado pelo código	ERROR(2)
FIX(arg)	Fornece parte inteira do arg	FIX(X-Y)
FOR..TO..STEP	Cria um loop. STEP é opcional e se não fôr usado o incremento será 1 (um)	FORR=0TO5
FRE(string)	Fornece a quantidade de memória disponível para armazenamento de strings	FRE(A$)
GOSUB	Chamada de subrotina	GOSUB 230
GOTO	Altera sequência normal de execução do programa	GOTO 120
IF(exp)THEN.. ELSE	Testa a expressão e se verdadeira executa comandos após THEN. Se for falsa executa comandos após o ELSE	IF A=1 THEN PRINT"A = 1" PRINT "A <> 1"
INKEY$	Varredura do teclado	A$=INKEY$
INP	Lê byte na porta de entrada	A=INP(123)
INPUT	Suspende execução do programa e espera entrada pelo teclado	INPUT A$
INPUT#-1	Lê um dado do cassete	INPUT#-1,A$
INT(arg)	Fornece maior número inteiro que não seja maior que o arg	INT(-3.14)
LEFT$(string,n)	Fornece os n primeiros caracteres da string	LEFT$(a$,3)
LEN(string)	Fornece o comprimento da string	LEN(A$)
LIST	Lista o programa	LIST LIST 10-90
LOG(arg)	Fornece o logaritmo neperiano	LOG(3)
MEM	Fornece quantidade de memória disponíveis. Não inclui memórias livres que estejam reservadas para string	MEM
MID$(str,p,n)	Fornece uma substring de comprimento n e começando na posição p	MID$(A$,3,4)
NEW	Apaga o programa e zera variáveis	NEW
NOT	Operador lógico NOT	NOT 5
ON ERROR GOTO	Altera execução do programa caso um erro aconteça	ON ERROR GOTO 10
ON n GOTO	Altera execução do programa dependendo do valor de N	ON N GOTO 100,120,780
OR	Operador lógico OR	5 OR 7
OUT	Manda byte p/ porta de saída	OUT 123,78
PEEK	Verifica posição de memória	PEEK(17531)
POINT(x,y)	Verifica se um ponto no vídeo está acesso (1) ou não (0)	POINT(63,24)

PEEK	Verifica posição de memória	PEEK(17531)
POINT(x,y)	Verifica se um ponto no vídeo está acesso (1) ou não (0)	POINT(63,24)
POKE	Armazena byte na memória	POKE 18765,34
POS(arg)	Indica posição do cursor	POS(0)
PRINT	Imprime no vídeo a expressão que se segue.	PRINT "EXEMPLO"
	A vírgula move o cursor para a próxima zona de tabulação	PRINT A,B$
	O ponto-e-vírgula evita um CR (Carriage Return)	PRINT A;B$
PRINT#-1	Guarda um dado no cassete	PRINT#-1,A$
PRINT TAB	Avança cursor para a posição especificada	PRINT TAB(60)A$
PRINT USING	Formata impressão	PRINT USING P$;N
PRINT @ n	Imprime na posição do vídeo especificada por n (0<N<1024)	PRINT@480,"Ola'"
PROTEGER?	Reserva de espaço na memória para armazenamento de programas em linguagem de máquina	
RANDOM	Realimenta semente do gerador de números aleatórios	RANDOM
READ	Acessa dados armazenados no no comando DATA	READ A$
REM	Ignora o resto da linha	REM Rotina 3
RESET(x,y)	Apaga ponto especificado por x e y	RESET(64,23)
RESTORE	Reseta apontador do comando DATA	RESTORE
RESUME	Retorno de rotina de manipulação de erro	RESUME RESUME 100 RESUME NEXT
RETURN	Retorno de subrotina	RETURN
RIGHT$(str,n)	Fornece os últimos n caracteres da string	RIGHT$(A$,3)
RND(exp)	Fornece número aleatório	RND(A+B)
RUN	Executa programa iniciando na linha de número menor ou na linha especificada	RUN RUN 90
SET(x,y)	Acende ponto no vídeo especificado por x e y	SET(63,23)
SNG(arg)	Fornece -1 para arg negativo 0 para arg zero e 1 para arg positivo	SNG (SIN(1.2))
SIN(arg)	Fornece o seno do argumento Arg deve estar em radianos	SIN(.57)
SQR(arg)	Fornece raiz quadrada do arg	SQR(A*B-2)
STOP	Interrompe execução	STOP
STR$(exp)	Converte a expressão numérica em string	STR$(A+B)
STRING$(n,chr)	Fornece uma string de n caracteres	STRING$(128,"*")
SYSTEM	Entra no modo monitor	
TAN(arg)	Fornece a tangente do arg Argumento em radianos	TAN(.37)
TROFF	Termina acompanhamento da execução do programa	TROFF

TRON	Ativa acompanhamento da execução do programa	TRON
USR	Chamada de programa em linguagem de máquina	USR(0)
VAL(string)	Fornece um valor numérico que corresponde a string	VAL("1"+A$)
VARPTR(var)	Fornece o endereço de onde uma variável está armazenada	VARPTR(A$)

APÊNDICE 6

ALGUNS PROGRAMAS EM BASIC

A melhor maneira de se aprender a programar é programando. Neste apêndice damos alguns programas para você rodar no seu DGT-100.
A finalidade principal destes programas é didática e você deve estudar os programas antes de rodá-los.
Alguns dos programas são bastantes simples e outros são mais complexos, mas não deixe de pelo menos ter uma idéia do que o computador está fazendo.

MEDIDOR DE REFLEXO

Este programa mede o seu reflexo. Quando aparecer no vídeo a palavra JÁ, aperte a tecla BREAK imediatamente.
Quanto menor o número que por último aparecer no vídeo, mais rápido é o seu reflexo.

```
10 CLS
20 FOR X=0 TO 300 : NEXT
30 PRINT"Prepare-se"
40 FOR X=0 TO RND(3000) : NEXT
50 PRINT
60 PRINT
70 PRINT"JA'................."
80 FOR X=1 TO 20
90 PRINTX;
100 NEXT
```

TRACADOR DE RETAS

Este programa traça uma reta entre dois pontos quaisquer que você fornece pelo Teclado

```
10 INPUT"PRIMEIRO PONTO=";X1,Y1
20 INPUT"SEGUNDO ponto=";X2,Y2
30 CLS
40 A=(Y2-Y1)/(X2-X1)
50 B=Y1-A*X1
60 IFX2<X1THENSU=X1:X1=X2:X2=SU
70 FORX=X1TOX2
80 Y=A*X+B
90 SET(X,Y)
100 NEXT
110 FORC=0TO2000:NEXT
120 GOTO10
```

DESENHOS

```
10 CLEAR200:DIMP(29,1)
20 CLS:PRINTCHR$(23);TAB(10);"POLIGONOS"
30 PRINT:PRINT:PRINT"DE AS COORDENADAS DOS VERTICES
E O PROGRAMA OS UNIRA COM
SEGMENTOS DE RETA.
OS PONTOS SÃO UNIDOS NA MESMA
SEQUÊNCIA DE ENTRADA.
40 GOTO310
50 PRINT@512,:PRINT:PRINT"O NÚMERO DE PONTOS É  LIMITADO
EM UM MAXIMO DE 30. SE HOUVER
MENOS QUE 30 PONTOS, COLOQUE
UMA COORDENADA NEGATIVA APÓS
O ÚLTIMO.
60 FORI=1TO5000:NEXT:PRINT:PRINT:PRINT:PRINT:PRINT:PRINT:PRINT
65 PRINT:PRINT:PRINT:PRINT@320,STRING$(32,140)
70 PRINT:PRINT:FORP=0TO29
80 PRINT@512,:PRINT:PRINT:PRINT:PRINT:PRINT:PRINT@578," PONTO ";P+1
90 INPUT"     X ";P(P,0):IFP(P,0)>127THEN190
100 INPUT"     Y ";P(P,1):IFP(P,1)>47THEN190
110 IF(P(P,0)ORP(P,1))>=0NEXT
120 CLS:FORSG=0TO(P-2)
130 X1=P(SG,0):X2=P(SG+1,0):Y1=P(SG,1):Y2=P(SG+1,1)
140 IF(X1=X2)AND(Y1=Y2)THEN160ELSEDY=Y2-Y1:DX=X2-X1
150 IFABS(DX)>ABS(DY)GOSUB210ELSEGOSUB260
160 NEXTSG
170 A$=INKEY$:IFA$=""THEN170
180 IFA$="R"THEN120ELSE20
190 PRINT"PONTO FORA DA TELA
COLOQUE-O NOVAMENTE.
200 FORI=1TO1000:NEXT:GOTO80
210 A=DY/DX:B=Y1-A*X1
220 FORX=X1TOX2STEPSGN(X2-X1)
230 SET(X,FIX(A*X+B+.5))
240 NEXTX
250 RETURN
260 A=DX/DY:B=X1-A*Y1
270 FORY=Y1TOY2STEPSGN(Y2-Y1)
280 SET(FIX(A*Y+B+.5),Y)
290 NEXTY
300 RETURN
310 PRINT@576,"O COMPUTADOR PODE TAMBÉM CRIAR
UM DESENHO!
ESCOLHA O QUE DESEJAR:
   1 - DAR OS PONTOS
   2 - COMPUTADOR DESENHAR
320 INPUTI:IFI=1GOTO50
330 RANDOM
340 FORP=0TORND(19)+10
350 P(P,0)=RND(128-1):P(P,1)=RND(48-1)
360 NEXT:GOTO120
```

CUPIM

Veja os cupins "comendo" o vídeo.

```
10 CLS
20 FOR X=1 TO 127
30 FOR Y=1 TO 47
40 SET (X,Y)
50 NEXT
60 NEXT
70 X=RND(127)
80 Y=RND(47)
90 RESET(X,Y)
100 GOTO 70
```

CASTELO

Este programa certamente agradará as crianças da casa.
Aperte a tecla BREAK para terminar a execução.

```
10 INPUT"Coloque suas iniciais";A$
20 CLS
30 Z=74
40 FORY=17TO47
50 FORX=ZTO127
60 SET(X,Y)
70 NEXT
80 IFY<23THENZ=Z+2
90 NEXT
100 FOR X=75 TO 123 STEP 4
110 SET (X,16)
120 SET (X+1,16)
130 NEXT
140 Q=0
150 FOR X=95 TO 125 STEP 5
160 FOR Y=47 TO 35 STEP -1
170 RESET (X,Y)
180 NEXT
190 NEXT
200 FOR X=95 TO 125
210 RESET (X,34)
220 NEXT
230 PRINT@688,"Castelo ";A$;
240 FOR X=73 TO 100
250 SET(X,12)
260 SET (X,13)
270 NEXT
280 FOR X=85 TO 95
290 SET (X,14)
300 SET (X,15)
310 NEXT
320 RESET (90,13)
330 RESET (91,13)
350 FOR X=2 TO 14
360 FOR Y=40 TO 43
```

```
370 SET (X,Y)
380 NEXT
390 NEXT
400 FOR X=3 TO 13 STEP 2
410 RESET (X,41)
420 NEXT
430 RESET (7,43)
440 RESET (8,43)
450 FOR X=1 TO 200 : NEXT
460 RESET (73,12)
470 RESET (73,13)
480 RESET (74,13)
490 SET (101,12)
500 SET (101,13)
510 SET (102,12)
520 SET (102,13)
530 FOR X=0 TO 200 : NEXT
540 SET (74,12)
550 SET (74,13)
560 SET (73,12)
570 SET (73,13)
580 RESET (102,12)
590 RESET (102,13)
600 RESET (101,12)
610 RESET (101,13)
620 FOR X=71 TO 2 STEP -1
630 P=X-73
640 Y=P[2/150 +12
650 SET (X,Y)
660 SET (X-1,Y)
670 RESET (X+1,Q)
680 RESET (X,Q)
690 Q=Y
700 NEXT
710 PRINT@ 771,"BOOM...";
720 GOSUB1000
730 FOR X=1 TO 18
740 RESET (X,45)
750 RESET (X,36)
760 RESET (X,37)
770 NEXT
780 GOTO350
790 END
1000 FOR X=0 TO 1500 : NEXT
1010 RETURN
```

APÊNDICE 7

LENDO FITAS E DADOS GRAVADOS EM 500 BPS

O microcomputador DGT-100 possui um circuito interno que lhe permite ler fitas gravadas em 500 bps.

- Se o programa for em BASIC, faça o seguinte:
  1 - Conecte o MIC do gravador ao MIC2 do computador

  2 - Conecte o MONITOR do gravador ao REM2 do computador. Como o segundo gravador nunca é acessado para leitura de programas, em 500 bps usamos o REM2 como entrada do circuito para fitas gravadas em 500 bps.

  3 - Escreva CLOAD#$, e aperte RETURN. Note que existe uma vírgula após o $. Proceda a partir deste ponto, como se fosse um programa em 2000 bps.

- Se o programa for em linguagem de máquina, faça as mesmas conexões descritas acima e:
  1 - Escreva **SYSTEM** e aperte RETURN

  2 - Um *? aparecerá no vídeo. Coloque então o nome do programa precedido de #$,. Note a vírgula depois do $. Aperte RETURN.

Exemplo:
#$,TESTE (programa chamado TESTE será lido em 500bps)

  3 - Proceda a partir deste momento, como se fosse um programa em 2000 bps.

- Se você for ler dados gravados em 500 bps, faça as mesmas conexões descritas acima e use o comando: INPUT#$,. Note a vírgula depois do $.

COMANDOS DE IMPRESSORA

Existem dois comandos especiais que lhe da acesso a impressora.

LLIST - Este comando e como o LIST (para vídeo) só que a listagem é feita pela impressora. Suas variações são:

LLIST: lista, na impressora, todo o programa na memória

LLIST 10Ø - Irá listar, na impressora, da linha 10Ø até o final do programa. LLIST 10Ø-20Ø: Irá listar, na impressora, da linha 10Ø a linha 20Ø do programa.

LLIST. Irá listar, na impressora, a linha que foi utilizada por último.

LLIST-10Ø: Irá listar, na impressora, todas as linhas do programa até a linha 10Ø, inclusive.

LPRINT: Este comando nos permite mandar para a impressora todas as informações que poderíamos mandar para o vídeo através do comando PRINT.

Suas variações são:
L PRINT variável ou "expressão" - Lista o valor da variável ou expressão entre aspas. Da mesma forma que o comando PRINT.

Exemplo:
```
1Ø A=3Ø
2Ø LPRINTA
3Ø LPRINT "teste"
```

L PRINT USING - Lista através da impressora do mesmo modo que comando PRINT USING faz pelo vídeo.

L PRINT TAB - Como PRINT TAB desloca o cursor para a posição idicada pelo número entre parênteses que segue o TAB. Como o PRINT TAB só aceita até 63, para tabelarmos colunas de número maior devemos usar o comando STRING$(n,32) após o último L PRINT TAB. Assim a impressora irá saltar n espaços (código 32) da última posição tabelada.

Exemplo:
```
1Ø L PRINT TAB(3) "Nome" TAB(4Ø) "ALTURA" STRING$(6Ø,32) "PESO"
```
Esta linha imprimirá a palavra Nome a partir da coluna 3, Altura a partir da 4Ø e Peso a partir da 1Ø6.

CÓDIGOS

Muitos códigos são usados para controlar a saída para impressora. Você pode usar L PRINT CHR$(n), onde n é um código de controle do DGT-100 para executar uma das funções mostradas abaixo:

Código	Função
10 ou 13	NOVA LINHA COM RETORNO DO CARRO DE IMPRESSÃO
11 ou 12	MOVE O CARRO DE IMPRESSÃO PARA O TOPO DA PÁGINA
138	RETORNO DO CARRO DE IMPRESSÃO COM NOVA LINHA

Exemplo:
10 L PRINT "LINHA UM"; CHR$(10); "LINHA DOIS"; CHR$(10); "LINHA TRES". produz:
LINHA UM
LINHA DOIS
LINHA TRES

OBS.:
Ao fim de uma linha um line feed (linha nova) irá sempre correr. A menos que se inclua um ponto e vírgula no final dos comandos L PRINT.
Veja a diferença entre os dois exemplos abaixo

```
10 L PRINT "LINHA UM";
20 L PRINT "LINHA DOIS";
30 L PRINT "LINHA TRES";
```

```
10 LPRINT "LINHA UM"
20 LPRINT "LINHA DOIS"
30 LPRINT "LINHA TRES"
```

O código para acessar o topo da página (11 ou 12) deve ser o primeiro ítem no comando L PRINT. Qualquer ítem anterior será ignorado.

Exemplo:
1 - L PRINT CHR$(11) "ESTA E A PRIMEIRA LINHA" e não

2 - L PRINT "ESTA E A PRIMEIRA LINHA" CHR$(11).
Pois em 2 a sentença não será impressa.

Estes códigos de topo de página (11 ou 12) são códigos passivos. O computador não "sente" quando se está no fim de uma página e automaticamente passa para a próxima. Você deve dizê-lo quando você quer que ele mude de página usando estes códigos.

Uma vez feito isso, o computador irá calcular o número de linha que precisa saltar para atingir a próxima página. Ele lê o número de linhas que já imprimiu nesta página no contador de linhas (endereço 16425). No endereço 16424, ele armazena o número de linhas permitido por página. Subtraindo o número de linhas impressas do número de permitidas, o computador calcula o número de linhas que deve deixar em branco. Então ele salta este número de linhas e passa para a próxima página.

Quando o computador é ligado, o formato de página é iniciado com 66 linhas por página.

Você pode mudar isso, pondo endereço 16424 o número de linhas por página que você desejar.

Exemplo:

POKE 16424, 40

Este comando muda o formato da página para 39 linhas por página, pois o valor é sempre igual ao número de páginas mais 1.

- No começo do seu programa você pode pôr o registrador de linhas em 1. Isto só é necessário uma vez, pois o computador o fará novamente quando mudar de página. Você pode fazer isso usando POKE 16425, 1

- Você deve fazer a escolha do modo de operação de sua impressora quanto ao número de linhas por páginas.

- Algumas impressoras respondem a códigos de controle adicionais. Consulte o manual de sua impressora e veja se existem e quais são suas funções.

Observando todas as instruções acima você irá explorar o máximo de seu conjunto DGT-100-Impressora.

ROTINAS DE SOM

Uma das caracteristicas de seu microcomputador DGT-1ØØ é a capacidade de gerar sons.

O DGT-1ØØ usa parte da porta de cassete para a geração de sons, pois está porta não é utilizada totalmente pelo cassete.

Podemos obter algum ruído com o programa abaixo:

```
1Ø OUT 255,Ø : OUT 255,1 : GOTO 1Ø
```

Esta é a máxima frequência que podemos conseguir com programas em basic, devido a velocidade de processamento do basic.

Então, para se conseguir um som mais atraente, devemos fazer uma rotina em linguagem de máquina, que é geralmente 1ØØ vezes mais rápido que o basic.

Faremos um programa que será parte em basic e parte em linguagem de máquina.

A rotina em linguagem de máquina será a seguinte:

```
1ØØ              CALL        ØA7FH
11Ø              LD          A,Ø1H
12Ø              LD          C,ØØH
13Ø              LD          DE,(4Ø3DH)
14Ø LP2          LD          B,L
15Ø              CPL
16Ø LP1          AND         Ø3H
17Ø              OR          E
18Ø              OUT         (ØFFH),A
19Ø              DEC         C
2ØØ              JR          Z,LP3
21Ø              DJNZ        LP1
22Ø              JR          LP2
23Ø LP3          DEC         H
24Ø              JR          NZ,LP1
25Ø              RET
26Ø              END
```

Este programa deverá ser assemblado usando o programa DGEDAS (editor assembler da Digitus) ou usando o apêndice A do manual do DIGBUG.

Uma das maneiras de se colocar este programa na memória do computador é através do comando POKE.

Para isto faremos o programa abaixo.

OBS.
Este programa requer que você responda a pergunta PROTEGER?, quando se liga o computador, com 32737.
Simplesmente ao invés de apertar a tecla RETURN quando o computador é ligado, bata o número 32737 e aperte RETURN.

```
10 FOR X = 32738 TO 32767
20 READ A
30 POKE X,A
40 NEXT
50 DATA 205,127,10,62,1,14,0,237,91,61,64,69,47,230,3,179,211,255,
13,40,41,16,246,24,242,37,32,241,201
```

Se você tem 48K de RAM responda PROTEGER? com 65505 e troque a linha 10 por:
```
10 FOR X = -30 TO -2
```

Agora faremos um programa que pegará os dois paramentros e os passará para a rotina em linguagem de máquina.

```
60 POKE 16526,127 : POKE 16527,226 : REM ** INFORMA AO BASIC ONDE
SE INICIA A ROTINA DE SOM **
70 INPUT "Qual a duração (de 1 a 127)";D
80 INPUT "Qual a frequência (de 1 a 255)";F
90 P = D*256 + 255 - F
100 A = USR(P)
110 GOTO 70
```

Para 48K de RAM troque a linha 60 por:
```
60 POKE 16526,255 : POKE 16527,226
```

Estude este último programa, adapte-o para suas necessidades e faça bom proveito da capacidade sonora de seu DGT-100.

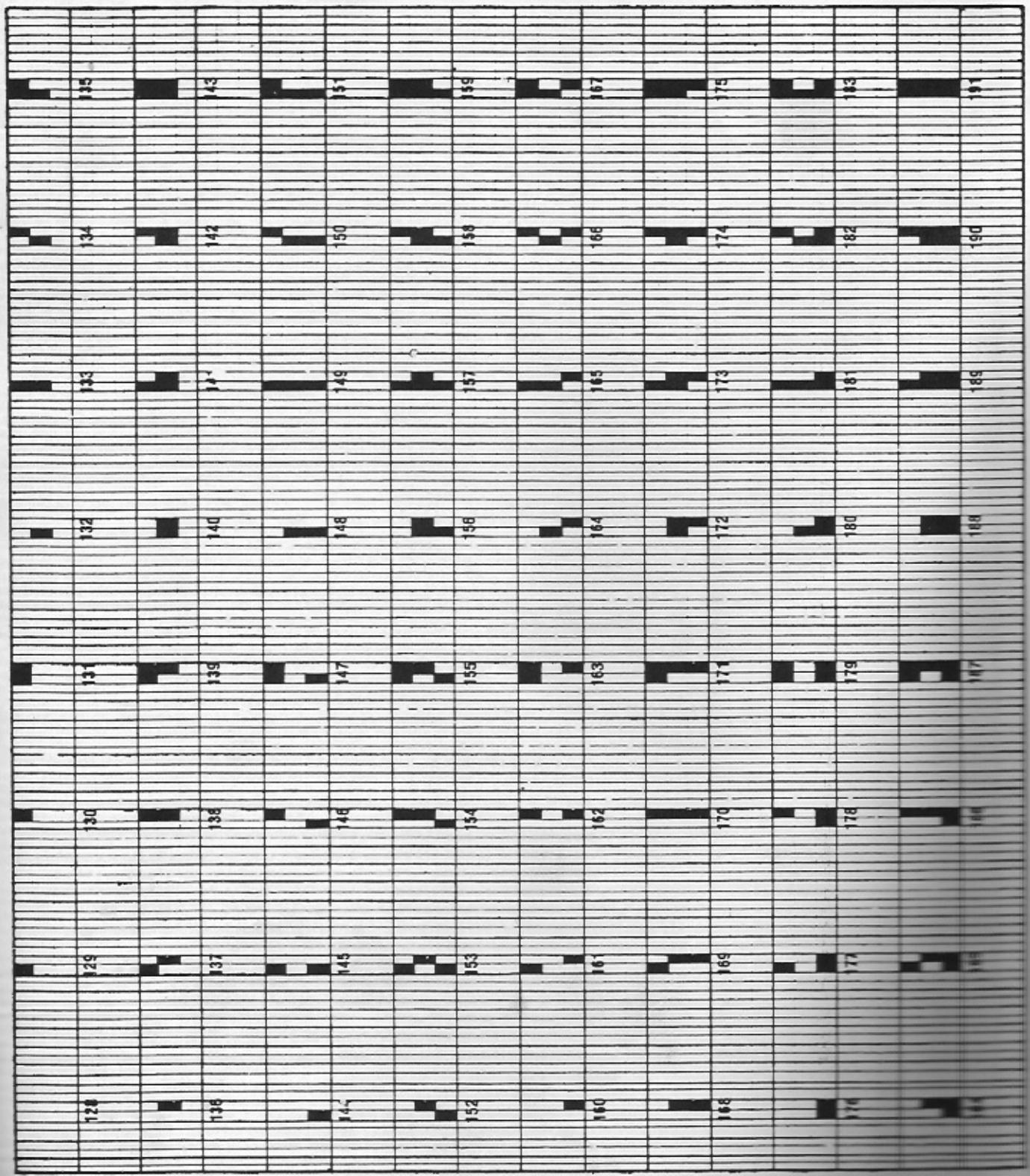
Exemplo:

Substitua a linha 70 do programa acima por:
```
70 A$=INKEY$ : IF A$="" THEN 70 ELSE A=USR(ASC(A$)) : GOTO 70
```

e tenha o seu DGT-100 funcionando como um "PIANO" digital.

APÊNDICE 8

CARACTERES GRÁFICOS



DIGITUS
Ind. Com. e Serv. de Eletrônica Ltda.

RUA GÁVEA, 150 - FONE: 332-8300
BELO HORIZONTE - MG