Toddy Forth 79

TODDY SOFTWARE *2022*

KELLY ABRANTES MURTA

ZX81 Toddy Forth-79

Version 1.13

Guia do Usuário

Segunda Edição

Toddy Software 2022

In memoriam do pequeno Toddy,
foram inúmeros momentos felizes
que guardaremos para sempre em
nossas memórias!

Toddy de Pedra de Guaratiba
(02/03/2008 - 20/06/2019)

PREFÁCIO

Forth é uma linguagem de programação criada por Charles Moore na década de 1970 que se tornou relativamente popular em microcomputadores na década de 1980. Entre os motivos dessa popularidade pode-se citar a economia de memória e velocidade de processamento, principalmente quando comparado ao BASIC, linguagem predominante em microcomputadores pessoais.

O primeiro contato que tive com Forth foi em 1984 através do compilador para ZX81 publicado por Thomas Löw na edição 39 da revista brasileira Micro Sistemas. Já acostumado com a lentidão do ZX BASIC, fiquei muito impressionado com a velocidade do compilador Forth, mas para um adolescente morando em uma pequena cidade do interior do Brasil era extremamente difícil obter mais informações sobre a linguagem, então neste momento não avancei mais.

Algum tempo depois, com o surgimento do interesse pela retrocomputação (especialmente o ZX81) foi uma questão de tempo até que minha atenção se voltasse novamente para o Forth, desta vez com muita informação disponível na Internet. Foi quando encontrei uma listagem em assembler Z80 do FigForth e comparando-a com o Forth de Thomas Löw as coisas começaram a fazer sentido. E quando mais tarde descobri o Camel Forth de Brad Rodriguez, consegui incorporar ao Forth minimalista de Löw várias das técnicas recém-aprendidas, o que resultou no lançamento de Toddy Forth em 2011.

Ao longo do tempo que se seguiu, vários novos lançamentos foram compartilhados com a comunidade do forum **Sinclair ZX World**[1] e críticas bem fundamentadas foram apresentadas, o Toddy Forth ainda não era um produto consistente e havia lacunas a serem preenchidas. E foi isso que comecei a fazer em meados de 2019 quando decidi por uma revisão completa do Toddy Forth para adequá-lo a um padrão estabelecido, optando pelo Forth 79-Standard. Assim chegamos ao ZX81 Toddy Forth-79, um sistema Forth completo para o microcomputador ZX81 e cujos recursos e usabilidade serão apresentados nas próximas páginas.

Mas primeiro gostaria de agradecer ao Thomas Löw por ter me apresentado ao Forth; Brad Rodriguez, por seu Camel Forth e pela série de artigos "Moving Forth", essenciais para a compreensão do "interior" do Forth; Lennart C. Benschop pelo o Editor, Double Extension, Floating Point Extension e outros conjuntos de palavras, extraídos de seu excelente Forth-83 para ZX Spectrum; Coos Haak Utrecht, pelo Assembler Extension. E um agradecimento muito especial ao Fred (Moggy), pelas críticas, sugestões, testes, relatórios de bugs e revisão do texto. Seu entusiasmo com Toddy Forth foi um grande incentivo para continuar o desenvolvimento.

Boa leitura e divirta-se usando o Toddy Forth-79!

Kelly A. Murta
Outubro, 2022

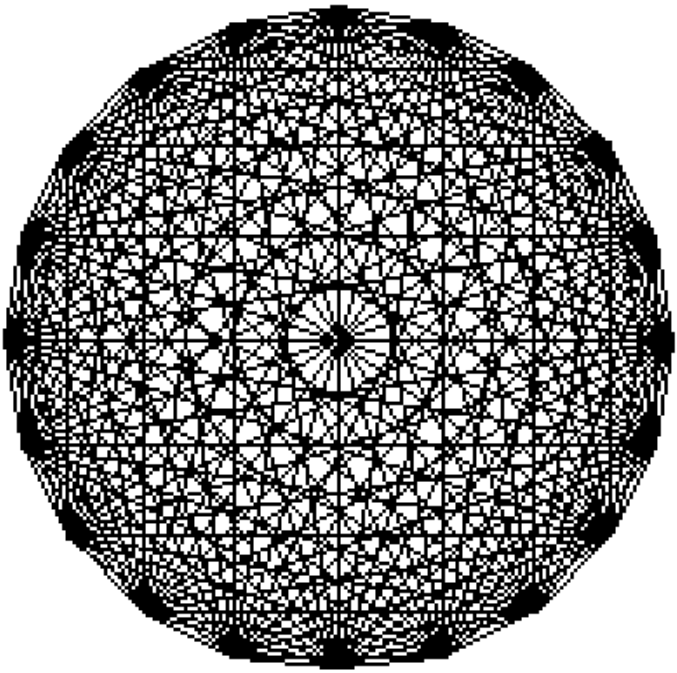
1 https://sinclairzxworld.com/

O QUE HÁ DE NOVO NA VERSÃO 1.13

O Toddy Forth-79 passou por uma grande mudança desde o lançamento da versão 1.03 que o torna praticamente um novo produto:

- O suporte integrado para a Chroma-81 foi movido para uma extensão externa.
- Uma nova rotina de teclado, agora independente do driver de vídeo e funcional com teclados mecânicos externos.
- A combinação SHIFT+BREAK é capaz de parar a maioria dos programas, seja em Forth ou em código de máquina.
- Reorganização das variáveis do sistema e do usuário para torná-las compatíveis com o sistema multitarefa adotado.
- Palavra **PAUSE** renomeada para **WAIT**.
- Removida a variável **(WAIT)** e adicionada a nova palavra **PAUSE** com função similar.
- A palavra **REVERSE** foi renomeada para **INVERSE**.
- A palavra **CH128** foi renomeada para **CHR$128**.
- Removidas as palavras **USER CLEAR** e **COPY**.
- Removida a variável **'ERRNUM** e criada a palavra **ERRNUM** como vetor de execução.
- **KEY EMIT AT PAGE ABORT** recriadas como vetores de execução.
- A palavra **SAVE** foi renomeada para **FSAVE**.
- Adionadas as palavras: **FLOAD RAND RND INKEY PLOT MEM EXEC: IS VALUE TO +TO**.
- Extensa reorganização do código.
- Adicionados novos conjuntos de palavras de extensão para gráficos de alta resolução, multitarefa e suporte a interface Chroma-81.
- Adicionadas ao Editor as funções de Copiar Linha.

Índice



Capítulo 1

1 O ZX81 TODDY FORTH-79

1.1 INTRODUÇÃO

O Toddy Forth-79 (chamado TF79 no restante deste documento) é um sistema Forth completo para os microcomputadores ZX81. É baseado nas especificações do Forth-79 Standard, com o REQUIRED WORD SET totalmente implementado, bem como algumas palavras do EXTENSION WORD SET e do REFERENCE WORD SET. O DOUBLE NUMBER EXTENSION WORD SET e o ASSEMBLER EXTENSION WORD SET estão disponíveis para serem carregados conforme necessário. Há também extensões para suportar números de ponto flutuante, ZX Printer, multitarefa, gráficos de alta resolução e recursos de cores da Chroma-81, entre outros. Várias palavras específicas para o hardware do ZX81 também estão disponíveis.

O TF79 é orientado a blocos, o que significa que o código-fonte é armazenado em blocos de 1024 bytes. Esses blocos (também chamados de Telas) são armazenados no disco de RAM. Um ou mais desses blocos podem ser salvos como um único arquivo no cartão SD, para posteriormente serem recarregados no disco de RAM.

Antes de prosseguir, gostaria de salientar que este trabalho pretende ser um guia do usuário para o Toddy Forth-79, descrevendo seus recursos, funcionalidades e características técnicas. Não se destina a ensinar programação em Forth, para isso já existem excelentes documentações disponíveis na internet. Eu recomendo começar com os dois livros a seguir:

- *The Complete Forth, by Alan Winfield*
- *Starting Forth (First Edition), by Leo Brodie*

Obs.: *A maioria das versões da web do livro de Brodie foi modernizada para o padrão ansi a pedido do próprio Brodie e não é tão boa para o Forth-79, portanto, deve-se procurar o próprio livro original ou a versão original em PDF nos links abaixo.*

https://ia803101.us.archive.org/6/items/winfield_alan_the_complete_forth/winfield_alan_the_complete_forth.pdf

https://www.forth.com/wp-content/uploads/2018/01/Starting-FORTH.pdf

1.2 REQUISITOS DE HARDWARE

O TF79 não suporta o uso de fitas cassete, por isso é necessário ter uma interface Zxpand (seja a versão clássica ou plus) conectada ao ZX81 para permitir a carga do sistema e

também para carregar e salvar arquivos, além de fornecer a memória RAM necessária (32Kb).

O kernel do sistema é alocado na parte inferior da RAM, entre os endereços 8192 e 16383. A área de memória de 16384 a 32767 é dedicada às variáveis do sistema, arquivo de tela, dicionário do usuário, pilhas, buffers de terminal e buffer de tela. A memória de 32768 a 40959 é utilizada como disco de RAM para armazenamento de telas Forth, com capacidade para armazenar até 8 telas.

A partir da versão 1.12, o suporte para Chroma-81 foi removido do kernel, sendo fornecido através de uma extensão externa que pode ser carregada quando necessário.

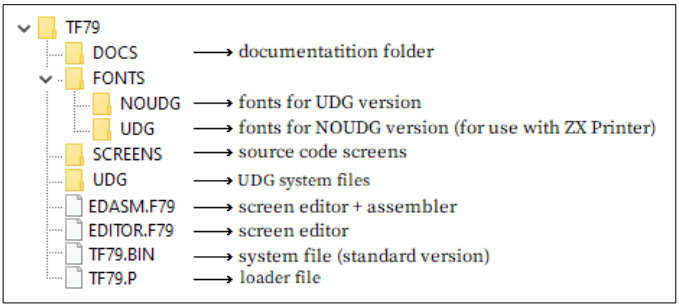
Os geradores de som compatíveis com o ZON X-81 continuam a ser suportados, permitindo a programação direta do PSG para gerar sons complexos.

O TF79 está disponível nas versões padrão e UDG. A versão padrão requer apenas a ZXpand e usa caracteres convencionais para representar caracteres ASCII não disponíveis no ZX81. A versão UDG requer, além da ZXpand, hardware que permita a redefinição de caracteres (como a placa UDG4ZXPAND criada por Andy Rea ou a interface Chroma-81), disponibilizando assim todos os caracteres ASCII utilizados por Forth.

1.3 INSTALAÇÃO

O TF79 básico é composto de dois arquivos, TF79.BIN que é o kernel do sistema a ser carregado na RAM no endereço 8192, e o TF79.P que é o carregador do sistema. Além disso, arquivos com extensão .F79 podem coexistir, eles são equivalentes ao TF79.P original com o vocabulário de usuário incluído e salvos com um nome próprio, como EDITOR.F79, por exemplo.

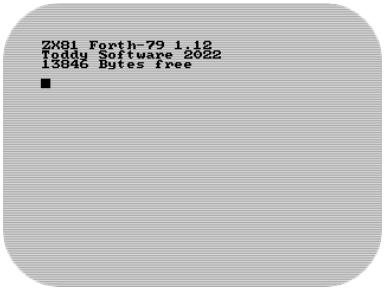
O sistema original vem com a seguinte estrutura de arquivos e pastas:



Copie as pastas relevantes para um cartão SD e insira-o na ZXpand.

Para carregar o sistema, vá até a pasta onde você salvou os arquivos e digite a partir do BASIC:

```
LOAD "TF79"
```



A tela inicial será apresentada com a identificação do sistema, informações de memória disponível e o prompt com o cursor piscando. Você também ouvirá um bipe audível se tiver um gerador de som conectado.

Digite **VLIST** e pressione NEWLINE para ver a lista de palavras incluídas no TF79 (pressione e segure a tecla SPACE para pausar a listagem ou SHIFT+SPACE para interromper a execução).

Obs.: *O kernel sempre será carregado do diretório atual. Por exemplo, se você deseja usar o Editor com a versão UDG, pode proceder da seguinte forma, supondo que esteja no diretório /TF79:*

```
CAT ">UDG"
LOAD "../EDITOR.F79"
```

A primeira sentença vai para o diretório /TF79/UDG; o segundo carrega o Editor do diretório superior e então carrega o kernel (TF79.BIN) do diretório atual.

Também é possível carregar um arquivo .F79 diretamente do ambiente Forth utilizando a palavra **FLOAD***, neste caso o TF79.BIN não será recarregado:*

```
FLOAD EDITOR.F79
```

1.4 O CONJUNTO DE CARACTERES

O TF79 trabalha com caracteres ASCII que são convertidos internamente em códigos do ZX81 quando enviados ao terminal. O 79-Standard define o conjunto de caracteres ASCII (código 32 a 127) como padrão, mas o TF79 adiciona um subconjunto de caracteres extra a isso (códigos 128 a 160).

Na versão NOUDG, os caracteres ausentes no ZX81 são representados pelos seguintes caracteres invertidos:

char	ASCII		char	ZX CODE	HOW TO TYPE IT
!	21	→		8D	SHIFT 1
@	40	→		8C	SHIFT 2
#	23	→		95	SHIFT 3
%	25	→		99	SHIFT 4
[	5B	→		90	SHIFT Q
]	5D	→		91	SHIFT W
&	26	→		97	SHIFT E
©	7F	→		9A	–
'	27	→		8F	SHIFT T
_	5F	→		96	SHIFT Y
~	7E	→		94	SHIFT A
|	7C	→		8E	SHIFT S
\	5C	→		98	SHIFT D
{	7B	→		93	SHIFT F
}	7D	→		92	SHIFT G
^	5E	→		8B	SHIFT H

1.4.1 A PALAVRA CHR$128

A versão UDG usa por padrão o modo CHR$128 para o gerador de caracteres. O CHR$128 usa códigos de caracteres ZX de 128 a 191 para fornecer 64 novos caracteres, totalizando 128 caracteres únicos disponíveis que podem ser redefinidos à vontade. A pasta FONTS contém alguns conjuntos de caracteres alternativos que podem ser carregados no endereço 15360 ($3C00) com

15360 BLOAD ***font.fn***.

Para visualizar o conjunto completo de caracteres do TF79, insira a seguinte definição:

```
: CHARS CR 160 32 DO I EMIT LOOP ;
```

e digite CHARS para executá-lo.

Versão padrão

Versão UDG

O modo de operação do gerador de caracteres pode ser alterado com a palavra **CHR$128**:

0 CHR$128 ---> desabilita CHR$128, retornando ao modo de operação tradicional
1 CHR$128 ---> ativa o modo CHR$128

O resultado é instantâneo, experimente para entender melhor a diferença entre os dois modos.

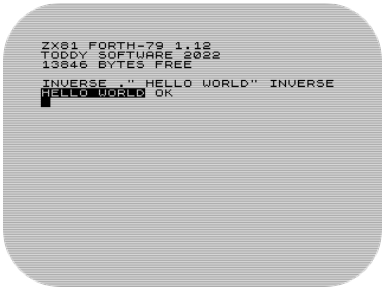
Obs:

1. **CHR$128** *só funciona com dispositivos UDG que usam o bit 0 do registrador I do Z80 para acessar os 64 caracteres extras. O UDG4ZXPAND não responde e a comutação deve ser feita fisicamente através de um jumper na placa.*
2. *Os caracteres ASCII do TF79 têm codificação diferente dependendo da versão usada: padrão ou UDG. Daí a necessidade de duas versões para cada conjunto de caracteres nas pastas FONTS/UDG e FONTS/NOUDG. Você pode estar se perguntando por que esses conjuntos de caracteres são necessários para a versão padrão? Simples, o módulo de impressão utiliza o conjunto de caracteres armazenado no endereço 15360, então você pode alterá-lo para ter impressões com caracteres diferentes. E a palavra CHR$128 também funciona com a ZX Printer.*

1.4.2 A PALAVRA INVERSE

A rotina de impressão de caracteres faz um *ou-exclusivo* do código do caractere com o conteúdo do endereço 16430 (**INVCHR**) antes de enviá-lo para a tela. A palavra **INVERSE** muda o conteúdo deste endereço para 0 ou 128, possibilitando alternar a impressão entre fundo branco/caractere preto e caractere preto/fundo branco. Experimente o exemplo:

INVERSE ." HELLO WORLD" INVERSE



A palavra **INVERSE** é mais adequada para uso com a versão padrão do TF79, mas também pode ser usado na versão UDG desde que o modo CHR$128 esteja desabilitado. Seu uso no modo CHR$128 não faz sentido.

1.5 O TECLADO

O TF79 vem com uma nova rotina de teclado com repetição automática e feedback audível para pressionamento de tecla (requer ZON X-81 ou PSG compatível). A combinação de teclas SHIFT+SPACE é constantemente monitorada e seu uso permite abortar a execução de qualquer programa, mas somente se estiver rodando em modo SLOW. Este recurso pode ser alterado usando as palavras **STOPON** / **STOPOFF**.

Os caracteres ASCII padrão são mapeados para o teclado conforme mostrado abaixo:

Os símbolos são acessíveis pressionando simultaneamente a tecla SHIFT. O SHIFT-0 apaga o último caractere digitado e o SHIFT-5 apaga toda a linha que está sendo digitada. A tecla NEWLINE informa que a linha digitada foi finalizada e os comandos digitados devem ser executados (estes funcionam de forma diferente dentro do Editor de Tela, como será visto mais adiante).

A combinação de teclas SHIFT-9 alterna o CAPS LOCK para letras minúsculas. Com a versão NOUDG ou se o modo CHR$128 estiver inativo, as letras minúsculas serão representadas por letras maiúsculas invertidas (letra branca, fundo preto).

O caractere © e o subconjunto extra (códigos 128 a 160) estão disponíveis apenas através de seus respectivos códigos de caracteres para uso com as palavras **EMIT** e **TYPE**.

O TF79 possui as palavras **KEY** e **INKEY** para leitura do teclado. **INKEY** lê o teclado e deixa na pilha o código ASCII da tecla pressionada ou 0 se nenhuma tecla foi pressionada. **KEY** espera até que uma tecla válida seja pressionada e deixa seu código ASCII na pilha.

Com a palavra **P@** é possível ler diretamente a porta FEh que dá acesso à matriz do teclado. Segue a representação do teclado com cada endereço associado a cinco teclas e cada tecla associada a um bit de D0 a D5 do byte lido da porta FEh.

Port Address	D0	D1	D2	D3	D4	D4	D3	D2	D1	D0	Port Address
FBFEh	1	2	3	4	5	6	7	8	9	0	FAFEh
FCFEh	Q	W	E	R	T	Y	U	I	O	P	F9FEh
FDFEh	A	S	D	F	G	H	J	K	L	NL	F8FEh
FEFEh	SH	Z	X	C	V	B	N	M	.	SP	F7FEh

P@ lê a porta endereçada pelo número na pilha, deixando o byte lido no TOS. Então, para saber se a tecla '0' foi pressionada, fazemos

```
64254 P@ \ read port FAFEh
1 AND    \ mask bit 0
```

que deixará 0 na pilha se a tecla foi pressionada ou 1 caso contrário.

Para evitar o bounce de tecla em teclados mecânicos externos, como o Memotech, a rotina de leitura do teclado implementa um tempo de bounce de aproximadamente 7 ms, o que é

adequado para a maioria dos casos. No entanto, se um tempo maior ainda for necessário, você pode alterar o valor do endereço 8732 (221Ch) que inicialmente contém o valor 30. Para um tempo de retorno de 9,6ms digite (cada unidade adicional adiciona aproximadamente 0,24ms):

```
40 8732 C!
```

Obs.: *Deve-se notar que teclados externos como os tipos Fuller ou Dk'tronics que se conectam à placa-mãe da mesma maneira que a membrana não sofrem problemas de debounce no mesmo grau que o Memotech que usa sua própria caixa de eletrônicos para se conectar ao computador.*

Da mesma forma, a taxa de repetição do teclado pode ser alterada mudando o conteúdo dos bytes nos endereços 8707 e 8725, respectivamente o tempo de espera para iniciar a repetição e o tempo de repetição. O valor padrão é 23 e 2 (eu sei, não há muito espaço para diminuir este último).

Depois de encontrar os valores adequados para você, as alterações podem ser salvas permanentemente digitando:

```
8192 8192 BSAVE >TF79.BIN
```

Isso substituirá o arquivo atual pela nova versão. Se você deseja preservar o arquivo atual, altere o caractere **>** para **+** no nome do arquivo (consulte o manual do ZXpand para obter detalhes).

1.5.1 O BUFFER DE ENTRADA DO TERMINAL

Com o TF79 em modo interativo, tudo que é digitado ecoa na tela e é simultaneamente armazenado no buffer de entrada do terminal (**TIB**) antes de ser interpretado, o que ocorre quando a tecla NEWLINE é pressionada.

O tamanho do **TIB** é de 128 bytes e quando ele estiver totalmente preenchido, o interpretador automaticamente assume o controle e começa a interpretar o que foi digitado. Se isso acontecer enquanto você estiver inserindo uma definição de dois pontos, é provável que a definição fique incompleta. Se a última palavra não tiver sido truncada, Forth permanecerá no modo de compilação e você poderá completar a definição digitando o restante e terminando-a com **;** .

Mas se a última palavra foi truncada, provavelmente não será encontrada pelo interpretador, gerando um aviso de erro e deixando uma definição incompleta no

dicionário. Para remover a última definição incompleta do dicionário, é necessário torná-la localizável usando a palavra **SMUDGE** e depois removê-la com **FORGET**:

SMUDGE FORGET <palavra-incompleta>

1.6 SUPORTE AO GERADOR DE SOM ZONX-81

O ZON X-81 foi um Gerador de Som Programável produzido pela BI-PAK Semiconductors, baseado no chip AY-3-8912 com 3 canais de som e controle total sobre pitch, volume, tones e ruído, todos com controle de envelope. Os detalhes técnicos da programação do PSG não serão discutidos aqui, para isso, consulte o manual do ZON X-81 e/ou a ficha técnica do chip AY, ambos disponíveis com uma rápida pesquisa no google.

TF79 tem duas palavras para acessar o PSG: **BELL** e **SOUND**.

BELL (--)
Emite um som de 1318 Hz com duração aproximada de 40ms. É utilizado na palavra **WARM**, executado na inicialização do TF e após a ocorrência de um erro.

SOUND (d r --)
Usado para programar o PSG, envia o número d para o registrador r do AY-3-8912.

Seguem alguns exemplos de efeitos sonoros que podem ser gerados pelo PSG. Comece definindo as seguintes palavras auxiliares (não é necessário digitar os comentários entre parênteses ou após a barra invertida):

```
16434 CONSTANT SEED

: RANDOMIZE 16436 @ SEED ! ;

: RND ( n1 -- n2) \ generates a random number between 0 and n1-1
  SEED @ 31521 * 6927 + DUP SEED ! U* SWAP DROP ;

: DLY (n -- )      \ delay
  0 DO LOOP ;

: CLPSG 14 0 DO 0 I SOUND LOOP ;

: MSOUND ( dn rn ... d2 r2 d1 r1 n -- ) \ program simultaneously
  0 DO SOUND LOOP ;                     \ n registers
```

Obs.:

*Antes de executar qualquer um dos exemplos, é recomendável executar **CLPSG** para limpar os registros do PSG.*

Sons simples (digite no modo interpretativo):

```
31 6 55 7 16 8 2 12 14 13 5 MSOUND \ Train
100 0 62 2 45 4 56 7 8 8 8 9 8 10 7 MSOUND \ Discord
210 0 90 2 60 4 56 7 16 8 16 9 16 10 40 12 8 13 9 MSOUND \ Bell
```

Sons complexos:

```
: BIRDS
  RANDOMIZE
  BEGIN
   13 8 254 7 0 1 3 MSOUND 85 RND 15
   DO I 0 SOUND 20 DLY
   LOOP ?TERMINAL
  UNTIL ;

: ADREAM  \ American Dream
  62 7 SOUND
  BEGIN
   2 -1
   DO 1 I - 7 * 2+  I 1+ 7 * 1+
    DO I 8 SOUND  51 100
     DO I 0 SOUND 15 DLY -1 +LOOP
      J NEGATE
     +LOOP 2
   +LOOP ?TERMINAL
  UNTIL ;

: WHISTLING 15 8 62 7 2 MSOUND 193 48 DO I 0 SOUND 30 DLY LOOP 63
7 SOUND ;
: EXPLOSION 31 6 7 7 2 MSOUND 11 8 DO 16 I SOUND LOOP 0 13 56 12 2
MSOUND ;
: LASER 0 13  15 12 16 8 55 7 4 MSOUND 32 1 DO I 6 SOUND 50 DLY 3
+LOOP ;
```

1.7 VETORES DE EXECUÇÃO E VALUES

1.7.1 VETORES DE EXECUÇÃO

A ação de uma palavra do dicionário geralmente é fixada no momento de sua definição. A palavra pode ser redefinida posteriormente e, a partir de então, todas as novas referências à palavra usarão sua nova definição. No entanto, todas as referências anteriores ainda usarão o significado antigo. Através do uso de vetores de execução é possível alterar a definição de uma palavra de forma que todas as referências já compiladas também utilizem a nova versão.

No TF79 um vetor de execução pode ser criado com a palavra **EXEC:**. Por examplo:

```
EXEC: CAKE
```

cria o vetor de execução **CAKE**. Inicialmente isso não faz nada, mas a ação da palavra pode ser definida a qualquer momento com **IS**, como segue:

```
: CREAM-CAKE  CR ." YUMMY!" ;

' CREAM-CAKE  IS  CAKE
```

Agora, sempre que você executar a palavra **CAKE**, você receberá a resposta "**YUMMY!**". Agora tente:

```
: MUD-CAKE  CR ." YUCK!" ;

' MUD-CAKE  IS  CAKE
```

Agora, **CAKE** provoca a resposta "**YUCK!**". O poder dessas palavras é que, ao contrário de simplesmente redefinir uma palavra com o mesmo nome, a ação da palavra muda imediatamente em cada definição em que foi compilada.

O TF79 vem com palavras importantes definidas como vetores de execução, o que significa que substituindo suas próprias definições o comportamento do sistema pode ser facilmente modificado.

Estas são as palavras definidas como vetores de execução no TF79:

KEY	**PAGE**
EMIT	**ERRNUM**
AT	**ABORT**

Obs.: *Cada vetor de execução é uma definição de código que possui o seguinte conteúdo em seu campo de código:*

```
LD HL,exec_address
JP (HL)
```

Portanto, você deve evitar definir vetores de execução em endereços de memória acima de 32767.

1.7.2 VALUES

Um **VALUE** é um híbrido entre **VARIABLE** e **CONSTANT**. Nós definiimos um **VALUE** do mesmo modo como definimos uma **VARIABLE**:

```
VALUE THIRTEEN  ok
```

Isto criará o value **THIRTEEN** inicializando-o com 0.

Assim como fazemos com uma **VARIABLE** podemos alterar o valor de **THIRTEEN**, para isso usaremos a palavra **TO** da seguinte forma:

```
13 TO THIRTEEN ok
```

No entanto, invocamos o novo **VALUE** da mesma forma que o fazemos com um **CONSTANT**:

```
THIRTEEN . 13  ok

49 TO THIRTEEN  ok

THIRTEEN . 49  ok
```

Também podemos adicionar um número a um **VALUE** com a palavra **+TO**:

```
VALUE A  10 TO A  ok

5 +TO A  ok

A . 15  ok
```

As palavras **TO** e **+TO** também funcionam com definições de palavras, substituindo o **VALUE** que o segue pelo que está atualmente no TOS, então pode ser perigoso seguir **TO**/**+TO** com qualquer coisa que não seja um **VALUE**. É perfeitamente seguro usar **TO**/**+TO** com **VARIABLE**s e **CONSTANT**s.

Você pode ver exemplos de uso de vetores de execução e **VALUE**s nas definições de **LINE** e **CIRCLE** na próxima seção.

1.8 A PALAVRA PLOT

O TF79 possui a palavra **PLOT** que permite manipular elementos gráficos comumente chamados de pixels. Cada pixel corresponde a 1/4 de um caractere na tela e é especificado através de suas coordenadas (x,y), em uma tela de 64 pixels de largura e 48 pixels de altura. O pixel localizado na borda inferior esquerda da tela possui coordenadas (0,0) e o pixel localizado na borda superior direita possui coordenadas (63,47).

Uma vez definida a localização do pixel, você deve decidir o que fazer com ela. Existem quatro possibilidades, ou modos de plotagem:

0 - Faça-o branco (unplot)
1 - Faça-o preto (plot)
2 - Deixe-o como está (move)
3 - Altere-o (invert)

Então **PLOT** precisa de três números na pilha: (x-coordinates, y-coordinates, ploting-mode --).

Tente:

```
32 24 1 PLOT
```

que fará com que um pequeno quadrado preto apareça no meio da tela. A melhor coisa a fazer é experimentar várias coordenadas e modos de plotagem para se familiarizar.

Palavras para traçar linhas e círculos com PLOT são definidas a seguir:

```
\ BRESENHAM'S LINE ALGORITHM
\ FROM ROSETTACODE.ORG
EXEC: STEEP       \ NOOP or SWAP
EXEC: YSTEP       \ 1+ or 1-

VALUE Y         VALUE COLOR
VALUE DELTAX    VALUE DELTAY

: LINE  ( x0 y0 x1 y1 color -- )
  TO COLOR
  ROT SWAP              ( x0 x1 y0 y1)
  2DUP  - ABS >R 2OVER  - ABS R> <
  IF    ' SWAP          \ swap use of
                        \ x and y
  ELSE 2SWAP ' NOOP
  THEN  IS STEEP  ( y0 y1 x0 x1)
  2DUP >
  IF SWAP 2SWAP SWAP    \ ensure
                        \ x1 > x0
  ELSE  2SWAP
  THEN                ( x0 x1 y0 y1)
  2DUP >
  IF    1-   ELSE   ' 1+
  THEN IS YSTEP
  OVER - ABS    TO DELTAY TO Y
  SWAP 2DUP - DUP TO DELTAX
  2/ ROT 1+ ROT ( error x1+1 x0)
  DO   I Y STEEP COLOR PLOT
       DELTAY - DUP 0<
       IF Y YSTEP TO Y DELTAX +
       THEN
  LOOP  DROP ;
```

```
\ MIDPOINT CIRCLE ALGORITHM

VALUE X    VALUE Y     VALUE C
VALUE P    VALUE X1    VALUE Y1
VALUE PY   VALUE PXY

: CIRCLE  ( x y r c -- )
  TO C    TO X1     TO Y    TO X
     0    TO P    0 TO Y1
  BEGIN X1 Y1 < NOT WHILE
    P Y1 2* + 1+   TO PY
    PY X1 2* - 1+  TO PXY
    X X1 +   Y Y1 +   C PLOT
    X X1 -   Y Y1 +   C PLOT
    X X1 +   Y Y1 -   C PLOT
    X X1 -   Y Y1 -   C PLOT
    X Y1 +   Y X1 +   C PLOT
    X Y1 -   Y X1 +   C PLOT
    X Y1 +   Y X1 -   C PLOT
    X Y1 -   Y X1 -   C PLOT
    PY TO P    1 +TO Y1
    PXY ABS PY ABS < IF
        PXY TO P  -1 +TO X1 THEN
  REPEAT ;
```

Essas palavras estão disponíveis na pasta SCREENS, como arquivos LINE.BLK e CIRCLE.BLK. Carregue-os com

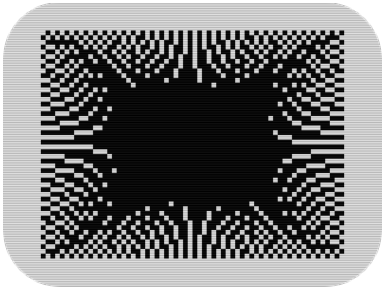
```
RUN SCREENS/LINE.BLK  ok
RUN SCREENS/CIRCLE.BLK  ok
```

Agora, tente este código:

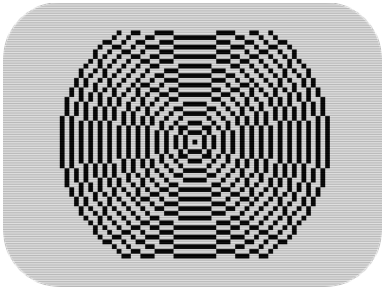
```
: MOIRE  ( step -- )
  PAGE 64 0
  DO I 0 63 I - 47 1 LINE
     I 47 < IF 0 I 63 47 I - 1 LINE THEN
  DUP +LOOP KEY 2DROP PAGE ;
```

e então digite **2 MOIRE** para ver o resultado.

Tente também com outros números (3, 4, 5,…).



2 MOIRE



CIRCLES

Os círculos concêntricos na última imagem foram gerados pela seguinte palavra:

```
: CIRCLES  PAGE
  30 0 DO  32  24  I  1 CIRCLE
  2 +LOOP ;
```

Obs.:

1. *As últimas coordenadas (x,y) plotadas são armazenadas nos endereços 16438 e 16439, respectivamente. PLOT aceita coordenadas fora da tela (x>63 e/ou y>47), o que permite desenhar linhas e círculos que vão além dos limites da tela, mas valores acima de 255 produzirão resultados errôneos.*
2. *Na versão UDG* **PLOT** *só funciona corretamente com o modo CHR$128 habilitado.*

1.9 GERADOR DE NÚMEROS PSEUDO-ALEATÓRIOS

Algo que frequentemente é muito útil em determinadas aplicações é um gerador de números aleatórios, como por exemplo, para simular o lançamento de dados. Entretanto é muito difícil produzir números aleatórios utilizando um computador de 8 bits, mas relativamente fácil produzir uma sequência de números gerados por uma operação matemática que pareçam ser aleatórios. Esses são comumente conhecidos como geradores de números pseudo-aleatórios (GNPA).

O TF79 traz embutido um GNPA rápido e que utiliza a técnica 'xor-shift' para produzir uma sequência de números com características suficientes para serem considerados aleatórios para a maioria das necessidades. Esse gerador produz uma sequência de números a partir de um número inicial conhecido com semente.

O TF79 usa as palavras **RAND** e **RND** para gerar números pseudo-aleatórios. **RAND** armazena o número do topo da pilha na variável de sistema SEED (16434) para ser usado por **RND**, mas se esse número for nulo, a semente será obtida da variável de sistema FRAMES (16436). **RND** gera um número situado entre 0 e o número no TOS – 1. Experimente o exemplo abaixo:

```
: MEASLES  PAGE
  BEGIN
    64 RND 48 RND 1 PLOT
  AGAIN ;
0 RAND  MEASLES
```

Para uma mesma raiz, a sequência de números gerados por **RND** será sempre a mesma.

1.10 CONDIÇÕES DE ERRO E SEUS EFEITOS

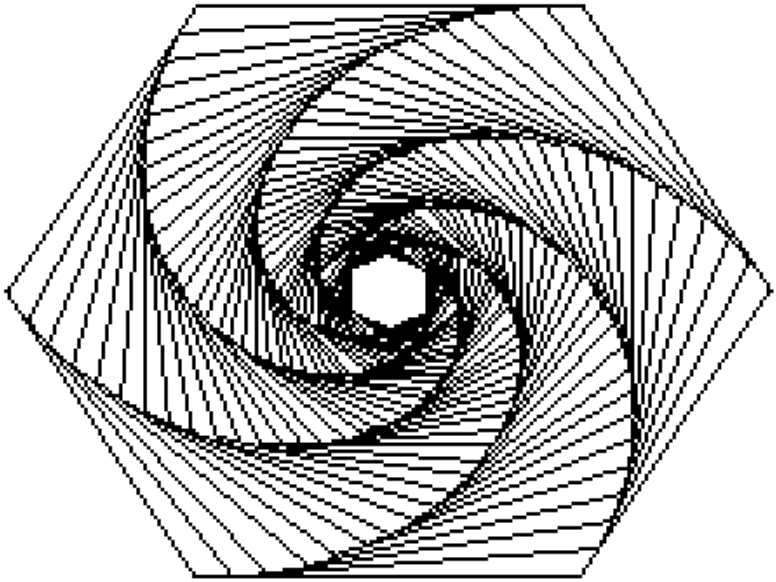
Quando um erro for detectado, ocorrerá o seguinte:

- Uma mensagem de erro será exibida.
- O nome da palavra Forth que causou o erro será exibido.
- **ABORT** será executado, retornando o sistema ao interpretador e esvaziando as pilhas.
- Se o erro ocorrer durante o carregamento de uma tela, a localização do erro será colocada na pilha para que ao digitar **WHERE** o Editor seja iniciado com o cursor posicionado à direita de onde ocorreu o erro.

Nem todas as condições de erro serão detectadas e nenhum erro será relatado nas seguintes condições:

- Divisão por zero, o resultado -1 será retornado na pilha.
- Número negativo quando o padrão exige um número positivo. Neste caso, o número será tratado como um número sem sinal.
- O manuseio incorreto da Pilha de Retorno, geralmente faz com que o sistema falhe.
- Endereço inválido fornecido para **EXECUTE**.
- Construção inválida de estruturas **IF-ELSE-THEN**, **BEGIN-UNTIL-WHILE-REPEAT** e **DO-LOOP**.
- Desequilíbrio da pilha ao definir novas palavras.
- Estouro de pilha.
- Execução de palavras de compilação durante o modo interpretativo.

A maioria desses erros tem o potencial de causar o travamento do sistema.



Capítulo 2

2 TELAS E ARQUIVOS

2.1 O DISCO DE RAM

O código fonte em Forth é armazenado em blocos (também chamados de telas) de 1Kb cada. Por conveniência, as telas em Forth são armazenadas em um disco de RAM localizado na memória a partir do endereço armazenado na variável **LO** até o final da RAM (por padrão, de 32768 a 40959). A constante **#SCR** fornece o número total de telas disponíveis no disco de RAM (8 telas por padrão).

Antes de usar o disco de RAM é aconselhável limpá-lo com o comando **FORMAT**.

2.2 CARREGANDO E SALVANDO ARQUIVOS

O comando

n **GET** name

carrega do cartão SD o arquivo com o nome especificado colocando-o no disco de RAM na tela n.

O comando

n1 n2 **PUT** name

salva no cartão SD as telas n1 a n2 com o nome de arquivo escolhido.

O comando

n1 n2 **INDEX**

mostra a primeira linha de cada tela de número n1 a n2. Portanto, é aconselhável armazenar um comentário (descrevendo a tela) na primeira linha de cada tela.

O comando

DELETE name
exclui o arquivo especificado do cartão SD.

O comando

n **LIST**

exibe o conteúdo da tela n. Como na palavra **VLIST**, a listagem é pausada enquanto a tecla BREAK é pressionada.

O comando

addr n **BSAVE** name

grava no arquivo com o nome especificado o bloco de n bytes localizado no endereço addr.

Você pode adicionar os caracteres **+** e **>** no início do nome para forçar a ação caso já exista um arquivo com o mesmo nome:

addr n **BSAVE +**name

renomeará o arquivo de destino para **.BAK**, a menos que já exista um backup, caso em que o erro **ZXP:8** normalmente ocorre.

addr n **BSAVE >**name

substituirá silenciosamente o arquivo de destino se ele já existir.

Obs.: *Se nenhuma extensão de nome de arquivo for fornecida, ZXpand adicionará automaticamente a extensão **.P**, então sempre tente usar uma extensão descritiva do tipo de arquivo a ser criado. Para arquivos de tela o convencional é usar a extensão **.BLK**.*

O comando

addr **BLOAD** name

carrega o arquivo especificado no endereço addr.

Obs.: *Nos comandos acima, quando aplicável, o argumento name pode incluir o caminho no nome do arquivo referenciado.*

O comando

CAT path

Lista o conteúdo do diretório especificado pelo caminho ou do diretório atual se o caminho não for especificado. Exemplos de uso:

CAT	mostra o conteúdo do diretório atual
CAT dir	mostra o conteúdo do diretório dir
CAT /	mostra o conteúdo do diretório raiz
CAT >dir	move para o diretório dir
CAT >..	sobe um nível de diretório
CAT >/	move para o diretório raiz
CAT +dir	cria um novo diretório dir

etc

Quando **CAT** preenche a tela, a listagem é pausada até que uma tecla seja pressionada.

2.3 CARREGANDO TELAS

O comando

n **LOAD**

carregará o código fonte do programa Forth da tela n (carregado anteriormente no disco de RAM pela palavra **GET**). Todo o texto na tela carregada será interpretado como se estivesse sendo digitado na linha de comando.

A palavra '\ ' (uma barra invertida seguida de um espaço) significa um comentário até o final da linha atual.

O comando **--->** pode ocorrer em uma tela e significa que a próxima tela também deve ser carregada.

O comando

RUN name

é equivalente a

```
1 GET name
1 LOAD
```

Ele carregará um arquivo no disco de RAM e imediatamente carregará (compilará ou executará) o código-fonte contido na tela 1 desse arquivo.

As telas podem conter comandos **LOAD**, para que outras telas possam ser carregadas a partir de uma tela.

Quando ocorrer um erro durante o carregamento, **ABORT** deixará o valor de **>IN** e o número da tela onde ocorreu o erro na pilha, então **WHERE** pode ser usado para iniciar o editor na tela dada, com o cursor posicionado à direita da palavra que causou o erro.

2.4 SALVANDO SUAS ALTERAÇÕES NO TF79

Para salvar o sistema Forth estendido você deve usar a palavra **FSAVE**:

FSAVE name

Ex: **FSAVE EDITOR.F79**

Isto irá gerar o arquivo EDITOR.F79 que pode ser carregado da maneira usual a partir do BASIC com o comando `LOAD ''EDITOR.F79''` ou diretamente do Forth com **FLOAD EDITOR.F79**. Caso já exista um arquivo com o mesmo nome, os caracteres **+** ou **>** também podem ser utilizados para direcionar a ação **FSAVE**, conforme visto na seção 2.2.

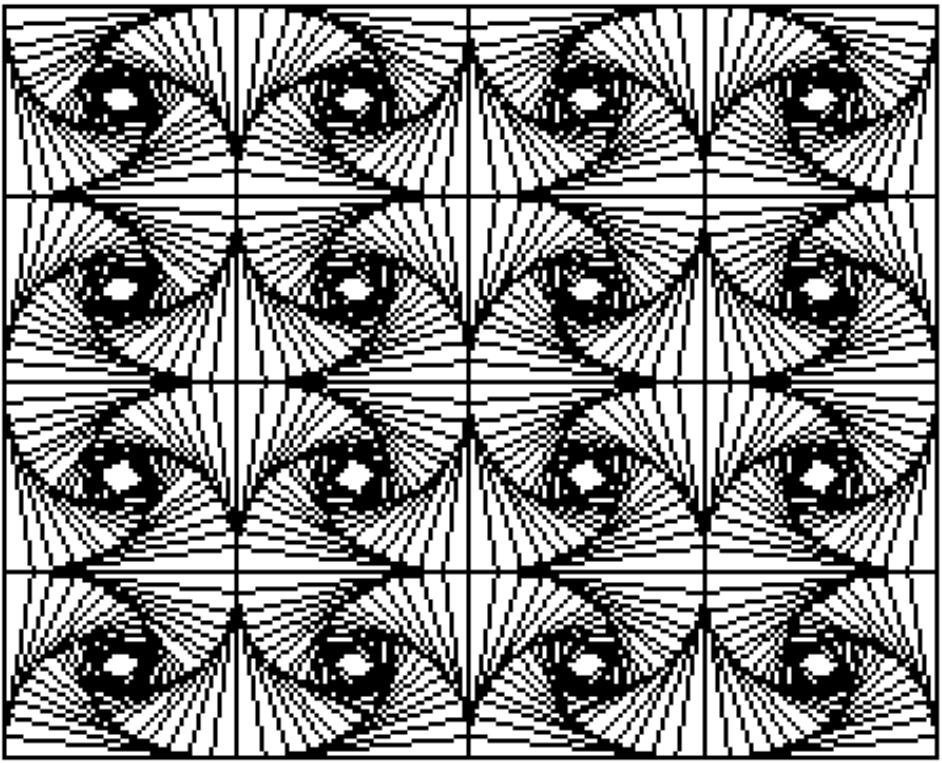
Obs.: *O uso da extensão **.F79** para arquivos Forth compilados é para diferenciá-los dos arquivos **.P** normais. Se nenhuma extensão for fornecida, a ZXpand adicionará automaticamente a extensão **.P**.*

SE você fizer alguma alteração no kernel, como substituir o conjunto de caracteres por outro, poderá salvar permanentemente as alterações com o **BSAVE**:

```
8192 8192 BSAVE >TF79.BIN
```

or

```
8192 8192 BSAVE +TF79.BIN
```



Capítulo 3

3 O EDITOR

3.1 INTRODUÇÃO

O Editor de telas foi desenvolvido por Lennart Benschop para seu ZX Spectrum Forth-83 (https://lennartb.home.xs4all.nl/index.html) e adaptado para uso com o TF79, incluindo as novas funções de copiar linha e colar linha. Lennart gentilmente me concedeu permissão para incorporar seu Editor ao TF79, muito obrigado Lennart!

O código fonte do editor está disponível na pasta SCREENS, para carregá-lo digite a sequência:

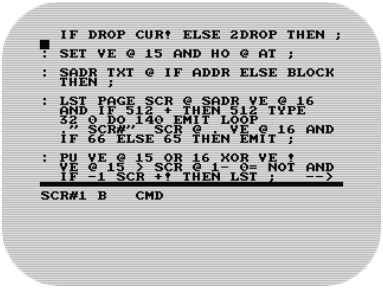
```
RUN SCREENS/EDITOR.BLK
```

Obs.: *O editor já compilado está disponível para uso imediato, você pode carregá-lo no BASIC com*

```
LOAD ''EDITOR.F79''.
```

Tradicionalmente, as telas Forth são formatadas em 16 linhas por 64 colunas, mas no TF79 são definidas telas de 32 linhas por 32 colunas (1024 bytes).

O editor de tela é iniciado com n **EDIT**, onde n é o número da tela a ser editada. Devido às limitações do ZX81, a tela Forth será dividida em duas partes de 16 linhas por 32 colunas, sendo mostrada uma metade de cada vez com o cursor. O número da tela será exibido na parte inferior, seguido pela letra **A** se a metade superior for exibida ou a letra **B** se a metade inferior for exibida. Mais à direita, na mesma linha, é informado o estado de funcionamento do editor (**EDT** ou **CMD**).



3.2 AS TECLAS DE CONTROLE

O editor inicia no modo de edição (EDT) onde as seguintes teclas têm uma função especial:

Cursor keys (SHIFT+5 to SHIFT+8): Move o cursor.
RUBOUT (SHIFT+0): Exclui o caractere no local do cursor. O resto da linha desloca uma posição para a esquerda. Se uma palavra for dividida entre o final da linha atual e o início da próxima linha, as linhas subsequentes rolarão para a esquerda, evitando que essas palavras sejam divididas.
SHIFT+9: chaveia o CAPS LOCK
NEWLINE: Move o cursor para a primeira posição na próxima linha.
SHIFT+NEWLINE: muda para o modo de comando (**CMD**).

O modo de comando fornece as seguintes teclas de função:

6: Insere uma linha em branco no local do cursor. As linhas abaixo movem uma posição para baixo, a última linha é excluída.
7: copie a linha no local do cursor para o **PAD** e exclua-a. As linhas abaixo sobem uma posição, uma linha em branco aparece no final.
8: Insere um espaço no local do cursor. O resto da linha muda uma posição para a direita. Se o caractere mais à direita no final da linha não for um espaço ou se a última palavra na linha atual for rolada junto com a primeira palavra na próxima linha, ele passará para a próxima linha. Uma ou mais linhas subsequentes também podem ser deslocadas para a direita.
C: copia a linha na localização do cursor para o **PAD**.
V: limpa a tela atual.
E: Apaga a linha no local do cursor preenchendo-a com espaços.
H: move o cursor para a primeira posição na primeira linha da tela atual.
K (+): move para a próxima metade da tela. De 1A a 1B, de 1B a 2A.
J (-): move para a metade anterior da tela. De 2A a 1B, de 1B a 1A. Ambas as teclas podem ser usadas para rolar pelo disco de RAM.
R: substitui a linha na localização do cursor pelo conteúdo do **PAD**.
Q: sai do editor. Você pode desfazer as edições na última tela com **EMPTY-BUFFERS**.

Para retornar ao modo de edição, toque na tecla NEWLINE.

3.3 OS MODOS DO EDITOR

O editor pode ser operado em dois modos: modo de arquivo e modo de bloco.

O comando **FILE** coloca o editor no modo de arquivo. Todo o disco de RAM é tratado como um grande arquivo de texto. As inserções e exclusões cobrem todo o disco de RAM. O texto será movido pelas telas quando as linhas forem inseridas ou excluídas. As teclas do cursor também se movem pelas telas.

O comando **BLOCKS** coloca o editor em modo de bloco. Cada tela é tratada individualmente. Se uma linha for inserida, a última linha da tela atual desaparecerá e não será movida para a próxima tela. Esta é a maneira normal de editar código-fonte no Forth.

3.4 O VOCABULÁRIO DO EDITOR

Palavras deste vocabulário são usadas internamente pelo editor e normalmente não são usadas por outros programas, então as descrições são muito curtas. O editor trabalha nos dados do buffer de bloco quando está no modo **BLOCKS** e diretamente no disco de RAM no modo **FILE**.

CT -- addr
temporizador para cursor piscando.

CUR c1 c2 --
alterna entre cursor e caractere de texto.

DEL --
apaga o caractere na posição do cursor.

DN --
move o cursor uma linha para baixo.

DRLIM -- addr
um a mais que o último endereço do disco de RAM (40960, por default). Se o disco de RAM for movido ou tiver sua capacidade alterada, **DRLIM** deve ser ajustado para refletir a nova condição.

EKEY -- c
lê um caractere do teclado e retorna o valor ASCII c.

HO -- addr
variável contendo a posição horizontal do cursor.

HOME --
move o cursor para o início da tela atual.

INS --
insere um caractere de espaço na posição do cursor.

LCLR --
preenche a linha atual com espaços em branco.

LCPY --
copie a linha atual para o PAD.

LDEL --
exclui a linha atual.

LE --
mova o cursor uma posição para a esquerda.

LIM -- addr
Um a mais que o último endereço do texto. É o endereço além do buffer do bloco no modo **BLOCKS**, é 40960 no modo **FILE**.

LINS --
insere uma linha em branco.

LPOS -- addr
endereço inicial da linha atual.

LREPL --
substitua a linha atual pelo conteúdo do PAD.

LREST -- u
número de caracteres a serem exibidos na tela a partir da linha atual.

LST --
imprime a meia tela atual na tela de vídeo com uma linha abaixo dela mostrando o número da tela e **A** ou **B**.

PD --
move meia tela para baixo.

POS -- addr
retorna o endereço no texto.

PU --
move meia tela para cima.

REST -- u
número de caracteres a serem exibidos na tela após a posição do cursor.

RI --
move o cursor uma posição para a direita.

SADR -- addr
retorna o endereço inicial da tela atual ou buffer de bloco.

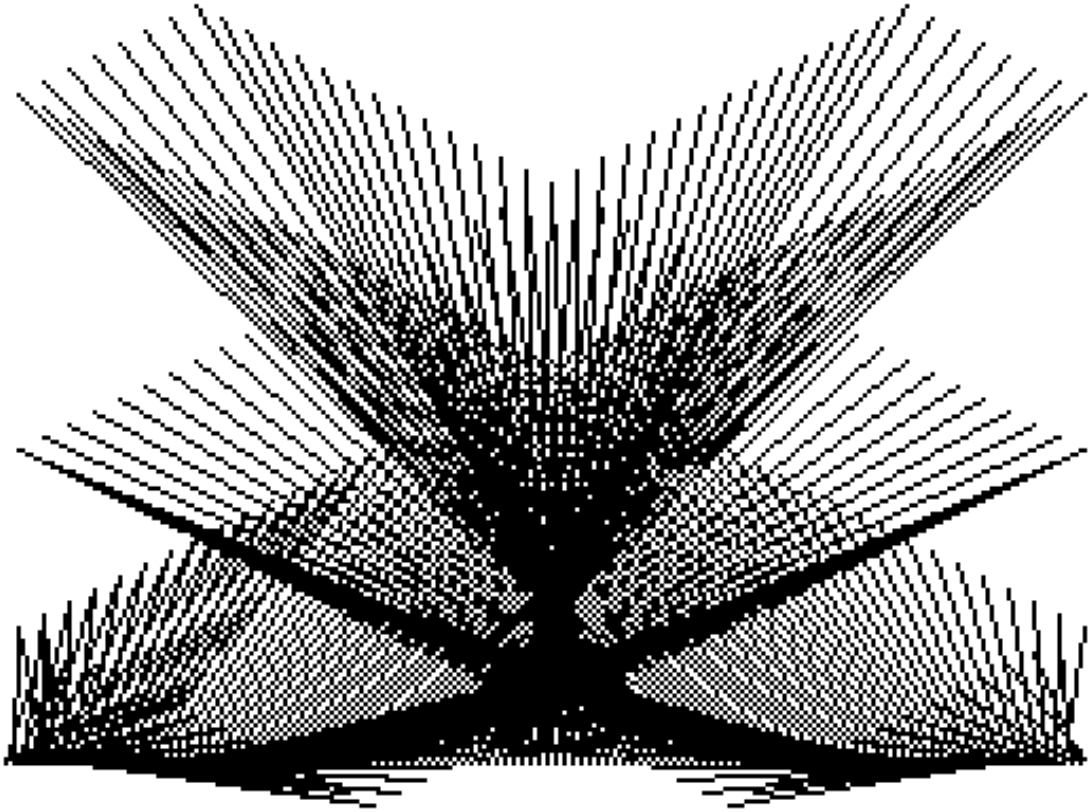
SCLR --
preenche a tela atual com espaços em branco.

SET --
Define a posição de impressão na tela de vídeo para a localização do cursor no texto.

TXT -- addr
variável contendo o modo do editor(**BLOCKS** ou **FILE**).

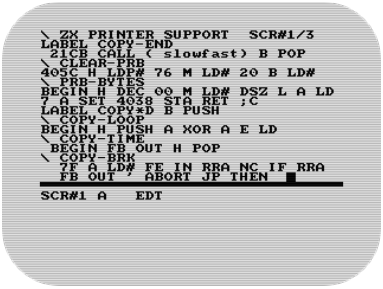
UP --
Move o cursor uma linha para cima.

VE -- addr
variável contendo a posição vertical do cursor desde o início da tela atual (intervalo 0..31).



Capítulo 4

4 O CONJUNTO DE PALAVRAS DA EXTENSÃO ASSEMBLER



4.1 INTRODUÇÃO

Usando o assembler você pode criar palavras em código de máquina. Este manual pressupõe que você esteja familiarizado com o Z80 e seus mnemônicos padrão. Existem duas razões pelas quais este assembler usa mnemônicos diferentes do padrão:

- O assembler padrão usa a mesma palavra como um mnemônico para instruções que devem ser traduzidas de forma diferente em código de máquina.
  Por exemplo, LD é usado para

  ```
  LD A,B
  LD (IX+12),23
  LD HL,(RPTR)
  LD A,I
  LD SP,HL
  ```

  e todos estes usam opcodes diferentes.
- Forth é adequado para expressões em notação pós-fixada. Portanto, é simples escrever um montador que espere mnemônicos de opcode após os operandos e o uso de expressões em operandos também é direto.

Os rótulos raramente são usados em um assembler FORTH. Em vez disso, fazemos uso de estruturas de controle como IF..THEN, como também é feito em FORTH.

Este montador foi projetado por Coos Haak em Utrecht, Holanda e ele o utilizou em seu próprio FORTH. Foi portado para FORTH83 por L.C. Benschop e Toddy Forth-79 por mim.

4.2 CARREGANDO O ASSEMBLER

O assembler consiste em três arquivos: TASM.BLK (1 tela), ASSEMBL1.BLK e ASSEMBL2.BLK (7 e 1 telas, respectivamente). Se você deseja ter o montador como parte permanente do FORTH (por exemplo, se deseja salvar o sistema completo com o montador), digite:

```
RUN ASSEMBL1.BLK
```

e então você pode salvar o sistema conforme descrito na seção 2.4.

Se você usar o assembler temporariamente, por exemplo, apenas para carregar outras extensões como o Double Number Extension Word Set, digite:

```
RUN TASM.BLK
```

Desta forma o montador é carregado em um endereço de memória alta fora do dicionário, deixando aproximadamente 10Kb de memória disponível para uso com o Assembler. Com o comando **DISPOSE** você pode remover o montador do sistema após o uso e quaisquer palavras definidas posteriormente (incluindo definições de código criadas com o assembler) permanecerão no dicionário.

4.3 CRIANDO PALAVRAS EM CÓDIGO

Para a criação de definições em código, as seguintes palavras estão disponíveis. Se a palavra for seguida pela letra A, a palavra fará parte do Conjunto de Palavras de Extensão Assembler. Se a palavra for seguida da letra F, está localizada no vocabulário **FORTH**, caso contrário, está localizada no vocabulário **ASSEMBLER**. Estado de montagem significa: estado de interpretação com **ASSEMBLER** como o vocabulário **CONTEXT** e **BASE** definido como 16.

;C -- F
sinônimo para **END-CODE**

;CODE -- FICA
Versão assembler de **DOES>**. Usado em uma definição de dois pontos contendo **CREATE**. Compila **(;CODE)** e coloca **FORTH** no estado de montagem. A definição de dois pontos em tempo de execução modificará o campo de código da palavra definida, de modo que chama o código de máquina montado após **;CODE**. A palavra criada, por sua vez, chamará o código de máquina montado após **;CODE** com o endereço do campo de parâmetro na pilha.

ASSEMBLER -- FA
Vocabulário contendo todos os mnemônicos do Assembler, definições de registro e afins.

CODE -- FA
lê uma palavra do fluxo de entrada e cria uma definição de código com esse nome. Coloca **FORTH** no estado de montagem e define o bit 'smudge', para que a palavra não seja encontrada.

END-CODE -- FA
termina uma definição de código. Limpa o bit 'smudge', para que a palavra definida como mais recente possa ser encontrada. **CONTEXT** é definido como **CURRENT** e **BASE** é definido como 10. Use esta palavra para encerrar uma definição em assembler que foi iniciada com **CODE**, **;CODE** ou **LABEL**.

ENDM -- IC
ssinônimo parar **;** finaliza uma macro.

LABEL -- F
lê uma palavra do fluxo de entrada e cria uma palavra com esse nome, que retornará o endereço de seu campo de parâmetro em tempo de execução. Em seguida, ele coloca **FORTH** no estado de montagem. Uma palavra criada com **LABEL** pode ser usada como endereço de salto ou chamada na definição de outras palavras de código.

MACRO -- F
Como **:** mas torna **ASSEMBLER** o vocabulário de contexto e define **BASE** como 16. Quando a macro for executada, ela reunirá (adicionará a uma nova definição de código) as instruções contidas nela.

XY -- addr F
variável que indica qual dos dois registros de índice será usado. Os valores são 0DDH para IX, 0FDH para IY, mesmos prefixos de opcode usados pelo Z80.

4.4 OS REGISTRADORES

O assembler usa os nomes A, B, C, D, E, H e L para os registradores Z80 de 8 bits. Além disso, usa o nome M, que pode ser usado na maioria dos lugares onde um registrador é permitido. M é o endereço de memória apontado por HL (é como o assembler 8080 que usa M em vez de (HL)).

Os pares de registradores são denominados B, D e H para indicar os pares de registradores BC, DE e HL, respectivamente. Além disso, SP e AF são nomes de pares de registradores. AF só é permitido com PUSH e POP. B, D, H e SP são permitidos em muitas instruções usando pares de registradores.
A ordem em instruções de dois operandos é origem seguida de destino, que é o inverso da notação padrão Z80 ou 8080. Por isso

```
B C LD
```

é equivalente a

```
LD C,B
```

Uso dos registradores de índice IX e IY:

- Use a palavra **X** na frente de instruções onde o uso do registrador HL está implícito, por exemplo **X LDSP** para indicar LD SP,IY
- Use **XH** ou **XL** em vez de **H** ou **L** para instruções que usam H ou L explicitamente.
- Para instruções que efetivamente fazem indexação (IY+12) em vez de (HL), use mnemônicos especiais começando com **)**

Por padrão, o registrador **X** escolhido é IY. O registrador IX está vinculado às rotinas de vídeo e dificilmente será usado no ZX81.
A palavra **%X** altera o registrador de índice usado para IX, **%Y** o altera de volta para IY.

4.5 O CONJUNTO DE INSTRUÇÕES

Usamos a mesma notação de pilha que na lista do glossário, mas com as seguintes adições para denotar valores (todos os valores são uma única entrada de pilha):

b: bit 0-7
cc: código de condição: z, cs, pe, m, v ou suas negações obtidas com a palavra **NOT**
r: registrador A, B, C, D, E, H, L ou M
rp: pares de registradores B, D ou H
rps: pares de registradores ou SP.
rpa: pares de registradores ou AF.
disp: deslocamento entre -128 e 127 para uso com endereçamento indexado.

LD `r1 r2 --`
copia r1 para r2. Equivalente a LD r2, r1

)LD r1 disp --
carrega r1 com o conteúdo do endereço IY+disp. Equivalente a LD r1,(IY+disp)

)ST r disp --
escreve r no endereço IY+disp. Equivalente a LD (IY+disp),r

LD# 8b r --
carrega r com valor imediato 8b. Equivalente a LD r,8b

)LD# 8b disp --
Carrega o valor imediato 8b no endereço IY+disp. Equivalente a LD (IY+disp),8b

MOV rp1 rp2 --
copia o par de registradores rp1 para rp2. Monta-se em duas instruções Z80. Por exemplo,
D B MOV é equivalente a
```
LD B,D
LD C,E
```

LDP# 16b rps --
carrega o par de registradores rps com valor imediato 16b. Equivalente a LD rps,16b

LDP addr rps --
carrega o par de registradores rps com o conteúdo do endereço addr.
Equivalente a LD rps,(addr)

)LDP rps disp --
carrega o par de registradores rps com o conteúdo do endereço do endereço IY+disp.
B 2)LDP é equivalente a:
```
LD C,(IY+2)
LD B,(IY+3)
```

STP addr rps --
escreve o par de registradores rps no endereço addr. Equivalente a LD (addr),rps

)STP rps disp --
escreve o par de registradores rps no endereço IY+disp. **B 2)STP** é equivalente a:
```
LD (IY+2),C
LD (IY+3),B
```

LDHL addr --
como **LDP**, mas para registrador HL. Usa a forma de opcode mais curta.

STHL addr --
como **STP**, mas para registrador HL. Usa a forma de opcode mais curta.

LDA addr --
carrega A do endereço addr. Equivalente a LD A,(addr)

STA addr --
escreve A no endereço addr. Equivalente a LD (addr),A

LDAP rp --
rp=B ou D apenas! Carrega A do endereço rp. Equivalente a LD A,(BC) ou LD A,(DE).
Mas use **M A LD** para LD A,(HL) !!!

STAP rp --
rp=B ou D apenas! Escreve A no endereço rp. Equivalente a LD (BC),A ou LD (DE),A.
Mas use **A M LD** para LD (HL),A !!!

LDSP --
carrega SP com HL. Equivalente a LD SP,HL

LDAI --
carrega A com I. Equivalente a LD A,I

LDIA --
carrega I com A. Equivalente a LD I,a

EXAF --
troca entre AF e AF'. Equivalente a EX AF,AF'

EXDE --
troca entre DE e HL. Equivalente a EX DE,HL

EXSP --
troca entre HL e o elemento superior na pilha. Equivalente a EX (SP),HL

CLR rps --
Carrega rps com 0. Equivalente a LD rps,0

ADD r --
adiciona r ao acumulador A. Equivalente a ADD A,r
As instruções **ADC**, **SUB**, **SBC**, **AND**, **XOR**, **OR** e **CP** funcionam da mesma maneira.

)ADD disp --
adiciona o conteúdo do endereço IY+disp ao acumulador A. Equivalente a ADD A,(IY+disp)
As instruções **)ADC**, **)SUB**, **)SBC**, **)AND**, **)XOR**, **)OR** e **)CP** funcionam da mesma maneira.

ADD# 8b --
adiciona a constante 8b ao acumulador A. Equivalente a ADD A,8b
As instruções **ADC#**, **SUB#**, **SBC#**, **AND#**, **XOR#**, **OR#** e **CP#** funcionam da mesma maneira.

ADDP rps --
adiciona rps ao registrador HL. Equivalente a ADD HL,rps
As instruções **ADCP**, **SUBP** e **SBCP** funcionam da mesma forma.
Observe que **SUBP** é uma macro composta por **A AND** e rps **SBCP**

INC rps --
incrementa o par de registradores rps em 1. Equivalente a INC rps
A instrução **DEC** funciona da mesma maneira.

INR r --
incrementa o registro r em 1. Equivalente a INC r
A instrução **DER** funciona da mesma forma (equivalente a DEC r).

)INR disp --
incrementa o byte no endereço IY+disp em 1. Equivalente a INC (IY+disp)
A instrução **)DER** funciona da mesma forma (equivalente a DEC (IY+disp)

RL r --
gira o registrador r uma posição para a esquerda. Equivalente a RL r
As instruções **RR**, **RLC**, **RRC**, **SRL**, **SRA** e **SLA** funcionam da mesma forma.

)RL disp --
gira o byte no endereço IY+disp uma posição para a esquerda. Equivalente a RL (IY+disp).
As instruções **)RR**, **)RLC**, **)RRC**, **)SRL**, **)SRA** e **)SLA** funcionam da mesma forma.

BIT b r --
teste do bit b do registrador r. Equivalente a BIT b,r
As instruções **RES** e **SET** funcionam da mesma forma.

)BIT b disp --
teste do bit b no endereço IY+disp. Equivalente a BIT b,(IY+disp).

As instruções **)RES** e **)SET** funcionam da mesma forma.

TST rp --
testa se o par de registradores rp é zero. B TST se expande para as instruções.

```
LD A, B
OR A, C
```

JP addr --
salta para o endereço absoluto addr. Equivalente a JP addr.
As instruções **JPNZ**, **JPZ**, **JPNC**, **JPC**, **JPPO**, **JPPE**, **JPP**, **JPM**, **CALL**, **JR**, **JRNZ**, **JRZ**, **JRNC**, **JRC**, **DJNZ** e **RST** funcionam da mesma maneira. As instruções de ramificação relativas (JR etc.) recebem endereços absolutos como entradas, mas são montadas com deslocamentos de ramificação relativos.
Exemplo: addr **JPNZ** é equivalente a JP NZ,addr

JPHL --
salta para o endereço em HL. Equivalente a JP (HL).
A instrução **JPIY** funciona da mesma maneira.

CALLC addr cc --
chamada condicional para o endereço addr. cc é um dos códigos de condição que podem ser especificados em letras minúsculas conforme descrito em 4.6.
Equivalente a CALL cc,addr

RETC cc --
retorno condicional. cc é especificado da mesma forma que em **CALLC**.
Equivalente a RET cc.

PUSH rpa --
empurra rpa para a pilha. Equivalente a PUSH rpa
A instrução **POP** funciona da mesma maneira.

PRT --
Chama a rotina interna Forth para imprimir um caractere.

IN 8b --
lê o acumulador A da porta 8b. Equivalente a IN A,(8b)
A instrução **OUT** funciona da mesma maneira.

INBC r --
lê o registrador r da porta no registrador C. Equivalente a IN r,(C)

A instrução **OUTBC** funciona da mesma forma (equivalente a OUT (C),r).

As instruções a seguir não têm operandos e usam exatamente os mesmos mnemônicos do assembler Z80 padrão: **NOP**, **RLCA**, **RRCA**, **RLA**, **RRA**, **HALT**, **RET**, **DAA**, **CPL**, **SCF**, **CCF**, **DI**, **HALT**, **EXX**, **LDIR**, **LDDR**, **CPIR** , **IM1**, **EI**, **IM2** e **NEG**.

Observe que nem todas as instruções do Z80 são implementadas por este assembler, mas é improvável que as instruções ausentes sejam usadas. Elas podem ser adicionados ao assembler se necessário ou seus opcodes podem ser montados diretamente com **,** ou **C,**.

RPTR, **UPTR** e **NEXT** são 3 constantes de endereço. Eles denotam o endereço do ponteiro da pilha de retorno, o endereço do ponteiro da área do usuário e o endereço do interpretador interno.

4.6 ESTRUTURAS DE CONTROLE

O montador pode usar as estruturas de controle **IF**..**THEN**, **IF**..**ELSE**..**THEN**, **BEGIN**..**UNTIL** e **BEGIN**..**WHILE**..**REPEAT** do FORTH e elas são implementadas com saltos relativos. **IF**, **WHILE** e **UNTIL** são precedidos por um código de condição, que pode ser **Z**, **NZ**, **CS** ou **NC**.

```
BEGIN
..
NC UNTIL
```

denota um loop que deve se repetir até que o bit de carry seja zero. Portanto, a instrução de salto relativa no final será JR C.

Além disso, temos **BEGIN**..**AGAIN** para um loop infinito e **BEGIN**..**DSZ** para um loop de contagem regressiva com uma instrução DJNZ no final.

As mesmas estruturas de controle, exceto **BEGIN**..**DSZ**, também podem ser usadas com saltos absolutos. Para especificar saltos absolutos, especifique as palavras relevantes em minúsculas. Os códigos de condição permitidos são **z**, **cs**, **pe**, **m** e **v** (onde **v** é o mesmo que **pe**). Cada um desses códigos de condição pode ser seguido por **NOT** para indicar a condição oposta.

A mesma construção **BEGIN**..**UNTIL** especificada com saltos absolutos se tornará:

```
begin
..
cs NOT until
```

Aqui estão alguns exemplos dessas estruturas, Z80 assembler convencional à esquerda e a conversão para Z80 Forth assembler à direita:

```
        code seg. 1
        JR Z,LBL01
          code seg. 2
LBL01:  code seg. 3
```

```
code seg. 1
NZ IF
  code seg. 2
THEN
code seg. 3
```

```
LBL01:  code seg.
        DJNZ LBL01
```

```
BEGIN
  code seg.
DSZ
```

```
        code seg. 1
        JR NC,LBL01
        code seg. 2
        JR LBL02
LBL01:  code seg. 3
LBL02:  code seg. 4
```

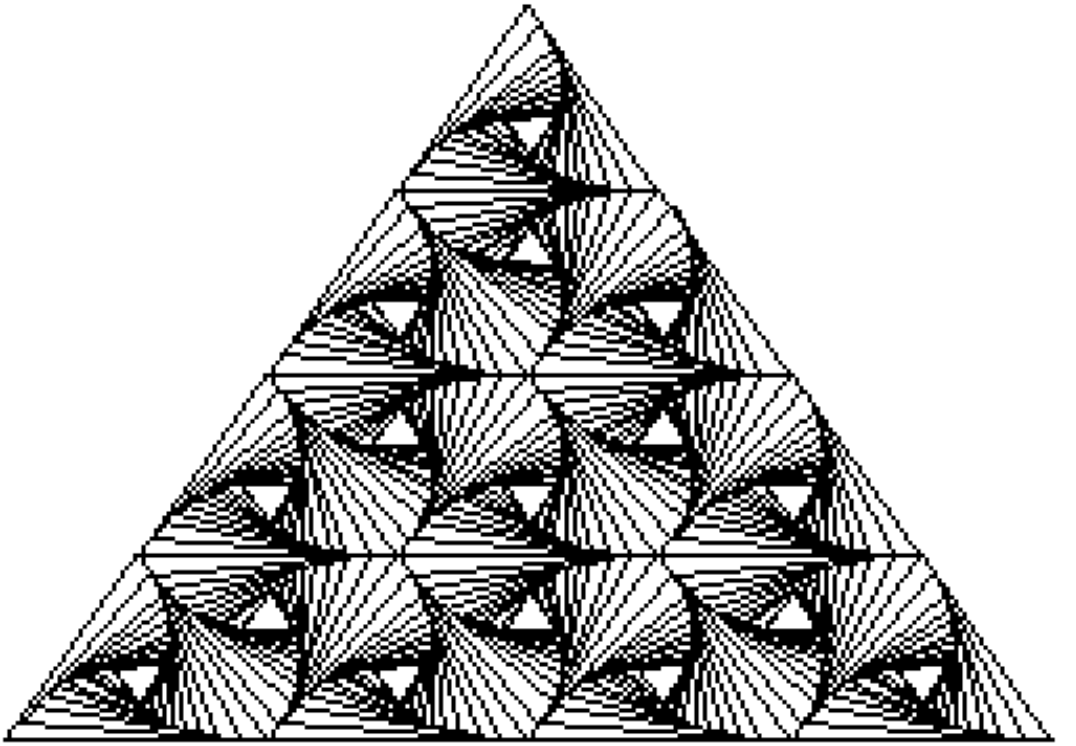
```
code seg.1
CS IF
  code seg. 2
ELSE
  code seg. 3
THEN
code seg. 4
```

```
LBL01:  code seg. 1
        JR Z,LBL02
        code seg. 2
        JR LBL01
LBL02:  code seg. 3
```

```
BEGIN
  code seg. 1
NZ WHILE
  code seg. 2
REPEAT
code seg. 3
```

```
LBL01:  code seg. 1
        JP C,LBL01
```

```
begin
  code seg. 1
cs NOT until
```



Capítulo 5

5 O CONJUNTO DE PALAVRAS DA EXTENSÃO DE NÚMEROS DUPLOS

5.1 INTRODUÇÃO

O arquivo DOUBLE.BLK (3 telas em tamanho) contém todas as palavras necessárias para implementar o Conjunto de Palavras de Extensão de Número Duplo completo e, adicionalmente, todas as palavras necessárias para realizar todas as operações com precisão dupla, incluindo multiplicação, divisão e raiz quadrada.

Antes de carregar **DOUBLE**, você deve primeiro carregar o assembler conforme explicado na seção 4.2. Em seguida, carregue **RUN DOUBLE.BLK** e finalmente (se aplicável) **DISPOSE** para remover o assembler.

5.2 PALAVRAS EXTRAS

2CONSTANT xd -- D
(runtime) -- xd
como **CONSTANT**, mas para um valor de 32 bits.

2UNDER xd1 xd2 -- xd2
remove o segundo valor da pilha.

2VARIABLE -- D
(runtime) -- addr
como **VARIABLE**, mas para um valor de 32 bits.

D* d1 d2 -- d3
multiplica d1 por d2.

D/ d1 d2 -- d3
divide d1 por d2.

D/MOD d1 d2 -- d3 d4
divide d1 por d2. d3 é o resto e d4 o quociente.

D2* d1 -- d2
Multiplica d1 por 2.

D2/ d1 -- d2 D
divide wd1 por 2.

D= xd1 xd2 -- f D
f=true se xd1 e xd2 forem iguais, false caso contrário.

D> d1 d2 -- f D
f=true se d1 for maior que d2, false caso contrário.

DMAX d1 d2 -- d3 D
d3 é o máximo de d1 e d2.

DMIN d1 d2 ---d3 D
d3 é o mínimo de d1 e d2.

DMOD d1 d2 -- d3
d3 é o resto da divisão de d1 por d2.

DU. ud --
como **D.** mas para número sem sinal.

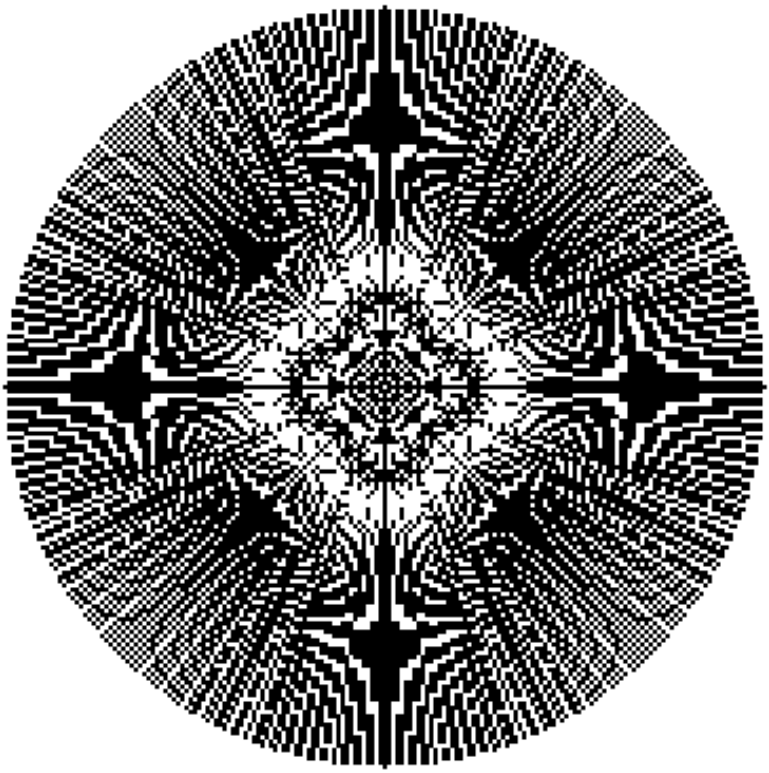
DU.R ud u --
como **D.R** mas para número sem sinal.

DU< ud1 ud2 -- f D
f=true se ud1 for menor que ud2, false caso contrário.

SQRT ud -- u
u é a raiz quadrada de ud, arredondada para baixo.

U.R u1 u2 --
como **.R** mas para número sem sinal.

UD/MOD ud1 ud2 -- ud3 ud4
divide ud1 por ud2. ud3 é o resto e ud4 é o quociente.



Capítulo 6

6 O CONJUNTO DE PALAVRAS DA EXTENSÃO DE PONTO FLUTUANTE

6.1 INTRODUÇÃO

Com esta extensão você pode usar números de ponto flutuante em Forth. Esses números têm 32 bits de tamanho e ocupam duas entradas na pilha de dados cada, portanto, você pode usar **2@** e **2!** para ler e escrever números de ponto flutuante na memória e todas as palavras de pilha de precisão dupla, como **2DUP** e **2SWAP**, para manipulá-los na pilha.

A palavra **NUMBER** é estendida com um meio de processar números de ponto flutuante. Agora você pode inserir números de ponto flutuante do interpretador de texto imediatamente após um número (que pode ou não conter um ponto decimal) com um sinal **&**. O sinal **&** pode opcionalmente ser seguido por um sinal **+** ou **−** e, em seguida, um pequeno inteiro que representa o expoente. O expoente é a potência da **BASE** com a qual o número deve ser multiplicado. Dessa forma, podemos inserir números de ponto flutuante em notação científica em qualquer base numérica.

Examplos

2&	o número 2
-2.718&	o número -2.718
12&+3	o número 12000
-0.041&	o número -0.041
-4.1&-2	o número -0.041

As palavras de ponto flutuante estão contidas em dois arquivos. O primeiro arquivo, FLOATING.BLK (8 telas de tamanho) contém os operadores básicos, palavras para imprimir números de ponto flutuante, a extensão de **NUMBER** para permitir a entrada de ponto flutuante e uma função de raiz quadrada. O assembler deve ser carregado primeiro. Isto pode ser feito com **RUN TASM.BLK** ou **RUN ASSEMBL1.BLK**. Então a extensão de ponto flutuante pode ser carregada com **RUN FLOATING.BLK**. Finalmente, o assembler pode ser removido com **DISPOSE**, se desejado.

O segundo arquivo TRANSCEN.BLK (4 telas de tamanho) contém funções trigonométricas e logarítmicas. Este arquivo não requer o assembler, mas requer que FLOATING já esteja carregado.

Se o sistema for salvo (conforme descrito na seção 2.4) e posteriormente recarregado ou se for reiniciado por meio de **COLD**, a palavra **FLOAT** deve ser executada para reativar a extensão para **NUMBER** que permite a entrada de ponto flutuante.

6.2 FORMATO E PRECISÃO

Um número de ponto flutuante consiste em duas partes. Os três bytes menos significativos formam a parte fracionária (também chamada de mantissa), um número de 24 bits entre 1 e 2. O bit inicial representa 1, o segundo bit representa 1/2, o terceiro bit representa 1/4 e assim por diante. Como o bit inicial é sempre 1, este é omitido do número e esta posição é usada para o sinal do número (0 para positivo, 1 para negativo). O byte mais significativo é o offfset do expoente por 128. Isso representa a potência de dois (no intervalo -128..127) com a qual a parte fracionária é multiplicada.

O maior número negativo (mais próximo de zero) e o menor número positivo representam o número zero. O maior número positivo e o menor número negativo representam + ou - infinito. Esses números são resultado de estouro aritmético ou de operações ilegais.

Os números de ponto flutuante têm uma precisão de cerca de 7 dígitos decimais. Os seguintes intervalos de números podem ser representados:

- -3*10**38 .. -3*10**-39
- 0
- +3*10**-39 e 3*10**38

O formato de ponto flutuante é semelhante, mas não idêntico ao formato de ponto flutuante binário de precisão simples IEEE-754. Não possui números subnormais e nenhum NaN como valor diferente de infinito. Além disso, o offset do expoente é diferente (127 para IEEE-754, 128 para o formato Forth).

6.3 PALAVRAS DE FLOATING

A notação de pilha é a mesma dos capítulos anteriores, mas fp é adicionado para representar um número de ponto flutuante de 32 bits. Palavras usadas apenas internamente (como rótulos do assembler) não são listadas aqui.

1& -- fp
A constant 1.

10& -- fp
A constant 10.

2CONSTANT 32b --
(runtime) -- 32b
Veja o capítulo 5

D->F d -- fp
converte um inteiro de precisão dupla d em um número de ponto flutuante com o mesmo valor (que não pode ser representado exatamente para inteiros grandes).

F* fp1 fp2 -- fp3
multiplicação de ponto flutuante

F+ fp1 fp2 -- fp3
adição de ponto flutuante

F- fp1 fp2 -- fp3
subtrai fp2 de fp1

F->D fp -- d
converte o número de ponto flutuante fp em um inteiro de precisão dupla, arredondando em direção a zero.

F. fp --
imprime fp em notação científica usando o símbolo & como símbolo "dez elevado à potência".

F.R fp n1 n2 --
imprime fp justificado à direita em um campo de pelo menos n1 caracteres de largura, inserindo espaços à esquerda, se necessário. n2 dígitos são impressos após o ponto decimal. Qualquer base numérica pode ser usada.

F/ fp1 fp2 -- fp3
divide fp1 por fp2

F0< fp --- f
f true se fp for menor que zero, caso contrário, false.

F0= fp -- f
f true se fp for igual a zero, caso contrário, false.

F2* fp1 -- fp2
multiplica fp1 por 2.

F2/ fp1 --- fp2
divide fp1 por 2.

F< fp1 fp2 -- f
f é verdadeiro se fp1 for menor que fp2, caso contrário, falso.

F= fp1 fp2 -- f
f é verdadeiro se fp1 for igual a fp2, caso contrário, falso.

F> fp1 fp2 -- f
f é verdadeiro se fp1 for maior que fp2, caso contrário, falso.

FABS fp1 -- fp2
fp2 é o valor absoluto de fp1.

FERRNUM f --
A extensão de **NUMBER** para entrada de ponto flutuante. É executado por **ERRNUM**. Analisa o número como um número de ponto flutuante e converte em um valor de ponto flutuante se f for verdadeiro. Se a conversão falhar, um erro será relatado

FI** fp1 n -- fp2
Eleva fp1 à potência n. n é um número inteiro.

FLOAT --
Ativa a extensão de ponto flutuante para **NUMBER**. Deve ser executado após um **COLD** start.

FNEGATE fp1 -- fp2
subtrai fp1 de 0.

FSQRT fp1 -- fp2
calcula a raiz quadrada de fp1.

NAN -- fp
Constante, valor infinito

X! fp1 8b -- fp2
Substitui o byte expoente de fp1 por 8b.

X@ fp -- fp 8b
8b é o byte expoente de fp.

6.4 PALAVRAS DE TRANSCEN

180/PI -- fp
constante, 180 dividido po pi.

2, 32b --
anexa 32b ao final do dicionário, versão de 32 bits de **,**

2VARIABLE --
(runtime) -- addr
Veja o capítulo 5.

ATN2 -- fp
Calcula o ângulo em radianos em relação ao eixo X, representado pelo vetor contido nas variáveis FX e FY.

ATNTAB -- addr
addr é o endereço inicial de uma tabela contendo os arco-tangentes de potências negativas de 2. Usado internamente por cálculos trigonométricos.

ATNTAB@ u -- fp
fp é o arco-tangente de 2 elevado a -u.

DEG fp1 -- fp2
converte o ângulo fp1 em radianos para o ângulo fp2 em graus.

F** fp1 fp2 -- fp3
eleva fp1 à potência fp2. fp1 deve ser não negativo. Elevar a uma potência inteira com **FI**** pode ter um primeiro argumento negativo.

F*2** fp1 n -- fp2
divide fp1 por 2 elevado a n.

F10** fp1 -- fp2
calcula 10 elevado a fp1.

FARCCOS fp1 -- fp2
calcula o arco-cosseno de fp1 em radianos.

FARCSIN fp1 -- fp2
calcula o arco-seno de fp1 em radianos.

FARCTAN fp1 -- fp2
calcula o arco-tangente de fp1 em radianos.

FCOS fp1 -- fp2
fp1 está em radianos, calcula o cosseno.

FEXP fp1 -- fp2
eleva e à potência de fp1.

FLN fp1 -- fp2
calcula o logaritmo natural de fp1.

FLOG fp1 -- fp2
calcula o logaritmo de base 10 de fp1.

FSIN fp1 -- fp2
fp1 está em radianos, calcula o seno.

FTAN fp1 -- fp2
fp1 está em radianos, calcula a tangente.

FX -- addr
FY -- addr
Duas variáveis de ponto flutuante contendo um vetor usado por cálculos trigonométricos.

LN10 -- fp
constante, o logaritmo natural de 10.

LN2 -- fp
constante, o logaritmo natural de 2.

LNTAB -- addr
addr é o endereço inicial de uma tabela contendo os logaritmos naturais de 1+2**-i. Usado por cálculos logarítmicos

LNTAB@ u -- fp
fp é o logaritmo natural de 1+2**-u

PI -- fp

constante, o número pi.

PI/180 -- fp

constante, pi dividido por 180.

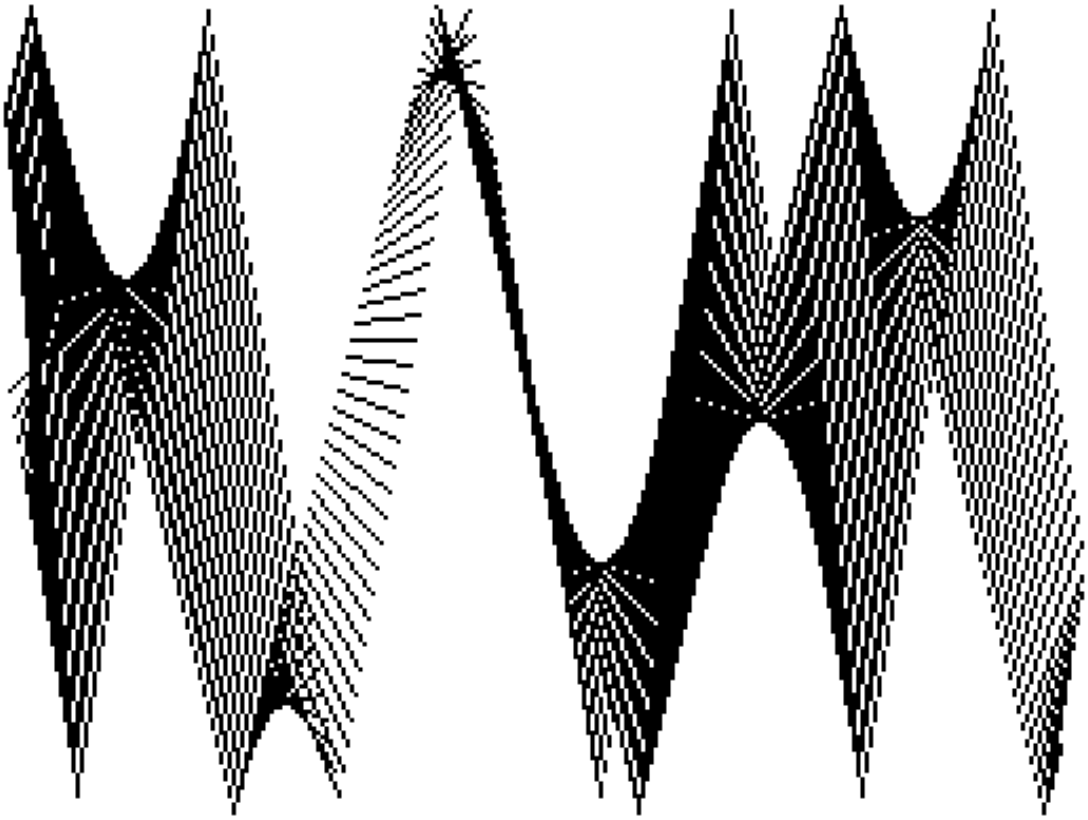
RAD fp1 -- fp2

converte o ângulo fp1 em graus para o ângulo fp2 em radianos. Necessário se você deseja calcular seno, cosseno ou tangente de ângulos expressos em graus. Exemplo: Para calcular o seno de 45 graus, digite:

```
45& RAD FSIN F.
```

TAN2 fp --

fp é positivo. Cria um vetor nas variáveis FX e FY que possui um ângulo de fp radianos em relação ao eixo x.



Capítulo 7

7 O CONJUNTO DE PALAVRAS DA EXTENSÃO DA IMPRESSORA

O arquivo PRINTER.BLK adiciona ao ZX81 Toddy Forth-79 a funcionalidade de redirecionar a saída de impressão para a ZX Printer. Antes de carregá-lo, você deve primeiro carregar o assembler conforme explicado na seção 4.2

No TF79 a palavra **EMIT** é um vetor de execução que por padrão executa a palavra **(EMIT)**. Para redirecionar a saída para a impressora ZX, **>P** redireciona **EMIT** para executar a nova palavra **PRINT**.

As seguintes palavras são definidas pela extensão Printer:

>P --
Redireciona a saída de texto para a impressora.

>S --
Direciona a saída de texto para a tela.

COPYSCR --
Envia uma cópia de toda a tela para a impressora.

LLIST n1 n2 --
Envia o conteúdo das telas n1 a n2 para a impressora.

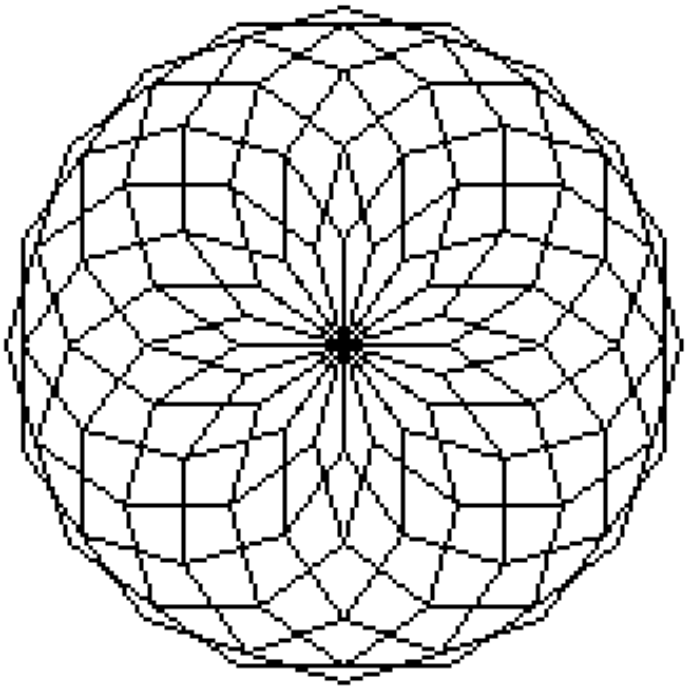
PRINT c --
Envia o caractere c para a impressora.

Digite

>P ." HELLO WORLD!" <enter>

Observe que após pressionar ENTER, todas as saídas de texto serão direcionadas para a impressora, incluindo o "**OK**". O que estiver sendo digitado somente será impresso após pressionar a tecla NEWLINE ou quando o buffer da impressora estiver cheio. Pressionar a tecla BREAK interrompe a impressão e executa uma reinicialização **WARM**.

Outra característica interessante é que a impressão utiliza a fonte dos caracteres armazenados no endereço 15360 (1024 bytes), mesmo levando em consideração o bit 0 do registrador Z80 I. Este bit determina se a geração de caracteres será no modo CHR$64 ou no modo CHR$128 (que é o padrão).



Capítulo 8

8 O CONJUNTO DE PALAVRAS DA EXTENSÃO CHROMA-81

O arquivo CHROMA.BLK implementa todas as palavras necessárias para o suporte total à interface Chroma-81. Antes de carregá-lo, você deve carregar o assembler conforme explicado anteriormente e então executar **RUN CHROMA.BLK**.

8.1 A INTERFACE CHROMA-81

A Chroma-81 foi desenvolvido por Paul Farrow para ser conectado ao barramento de expansão do ZX81, permitindo a conexão a uma TV através de um conector SCART para fornecer uma imagem RGB nítida e clara. Também permite adicionar cores à imagem gerada pelo ZX81, suportando dois modos de operação: o Modo de Cor de Caracteres (Modo 0) e o Modo de Cor de Atributo (Modo 1). Para usá-lo com o TF79, coloque as chaves 3 e 6 na posição ON.

O caractere Color Mode permite definir duas cores (uma cor de primeiro plano e uma cor de fundo) para cada scanline de cada um dos 128 caracteres (caracteres com códigos ZX 0-63 e 128-191). A tabela de cores de caracteres está posicionada do endereço 49152 ($C000) ao endereço 50175 ($3CFF), 8 bytes para cada caractere.

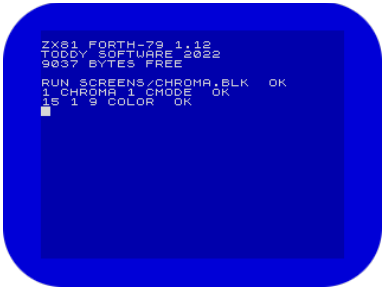
O Modo de Cor por Atributo usa uma área de atributo para mapear as cores de primeiro e segundo plano para cada posição de caracter na tela, semelhante à área de atributos do ZX Spectrum. A área de atributos está localizada do endereço 49323 ($C0AB) ao endereço 50115 ($C3C3).

É recomendo ler a documentação do Chroma-81 para um entendimento completo de seus recursos.

8.2 OPERAÇÃO DO CHROMA-81 COM O TF79

Após carregar a extensão Chroma-81, a primeira coisa a fazer é vetorizar as palavras **EMIT** e **PAGE** para trabalhar com as novas funcionalidades, o que é feito com a palavra **CHROMA**:

```
1 CHROMA
```



Em seguida, com a palavra **CMODE** defina o modo em que deseja trabalhar:

```
1 CMODE
```
(seleciona o modo 1)

Agora defina as cores da tela:

```
15 1 9 COLOR
```

(você pode experimentar outros valores da lista abaixo)

```
0 Preto
1 azul
2 Vermelho
3 Magenta
4 Verde
5 Ciano
6 Amarelo
7 Branco
8 Preto brilhante
9 Azul brilhante
10 Vermelho brilhante
11 Magenta brilhante
12 Verde brilhante
13 Ciano brilhante
14 Amarelo brilhante
15 Branco brilhante
```

BORDER define a cor da borda da tela, **INK** e **PAPER** definem as cores de primeiro plano e de fundo para as próximas impressões. Experimente:

```
: HELLO
  16 0 DO
   CR I INK 15 I - PAPER
   I BORDER
   ."Hello World!" 10 WAIT
  LOOP ;
HELLO
```

Obs.: **INK** *e* **PAPER** *só funcionam no Modo 1 (Modo de Cor por Atributo).*

Para o modo 0, os atributos de ink e paper da palavra **COLOR** são aplicados a todo o conjunto de caracteres:

```
0 CMODE
```
(seleciona o modo 0)
```
12 0 4 COLOR
```

DEFCOL é usado para definir as cores de um determinado caractere e **DEFCHR** define sua forma. Ambos esperam 9 valores na pilha, o TOS é o código ASCII do caractere e os demais representam cada linha do caractere com a primeira linha correspondente ao número mais à esquerda. Cada número de cor no **DEFCOL** representa as cores de fundo e primeiro plano codificadas como paper*16+ink. Assim, para paper=amarelo e ink=vermelho brilhante, temos 6*16+10=106. O exemplo abaixo define as cores e uma nova forma para o caractere '**>**':

```
7 7 6 10 6 13 13 15 62 DEFCOL
24 24 18 126 88 24 100 70 62 DEFCHR
EMIT 62
```

E, por último, para desativar os recursos de cores da Chroma-81, use

```
0 CHROMA
```

8.3 PALAVRAS DA CHROMA-81

BORDER (n --)
Define a cor da borda da tela.

CHROMA (f –)
Ativa/desativa o suporte a cores na Chroma-81. Quando f é verdadeiro, as palavras **EMIT** e **PAGE** são vetorizadas para **cemit** e **cpage**, respectivamente. Com f false, **EMIT** e **PAGE** são vetorizados novamente para suas ações padrão **(EMIT)** e **(PAGE)**, e o recurso de cor do Chroma-81 é desabilitado.

CMODE (f --)
Define o modo de operação da Chroma-81 de acordo com o valor de f:
0 - define o modo de cor de caractere (Modo 0)
1 - define o Modo de Cor por Atributo (Modo 1)

COLOR (ink paper border --)
No modo 0, define a cor da borda e aplica as cores ink e paper a todo o conjunto de caracteres. No modo 1, define as cores de toda a tela.

DEFCHR (n0 n1 n2 n3 n4 n5 n6 n7 c --)
Define a forma de um determinado caractere c.

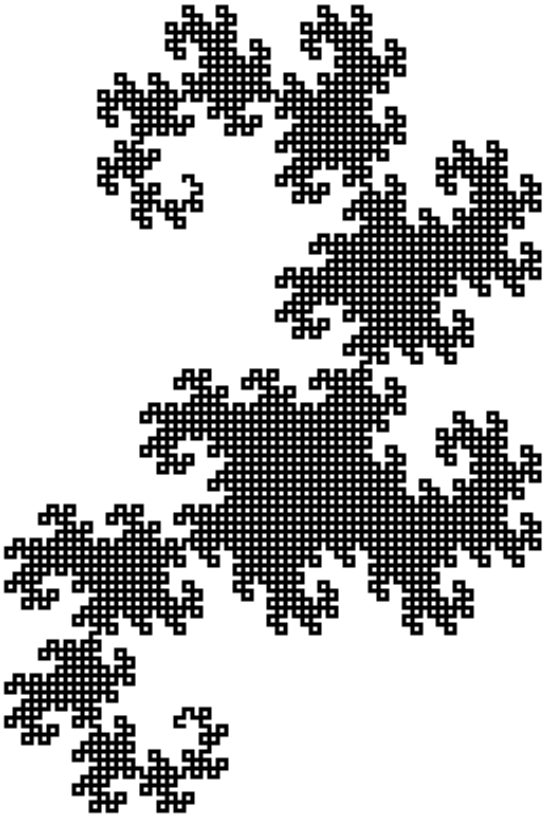
DEFCOL (n0 n1 n2 n3 n4 n5 n6 n7 c --)
Define as cores ink e paper para cada linha de varredura (n0 a n7) de um determinado caractere c. As cores são codificadas como paper*16+ink.

INK (n --)
Define a cor de primeiro plano para os próximos caracteres a serem enviados para a tela.

PAPER (n --)
Define a cor de fundo para os próximos caracteres a serem enviados para a tela.

cemit (c --)
Envia o caractere c para a tela atualizando os atributos de cores. É atribuído a **EMIT** por **1 CHROMA**.

cpage (--)
Limpa a tela atualizando os atributos de cores. É atribuído a **PAGE** por **1 CHROMA**.



Capítulo 9

9 O CONJUNTO DE PALAVRAS DA EXTENSÃO GRÁFICA HIRES

O arquivo HGR.BLK (8 telas) contém todas as palavras necessárias para implementar um display gráfico de alta resolução total com 256x192 pixels. Ele também oferece um terminal gráfico totalmente funcional que pode substituir completamente o terminal de texto padrão.

Antes de carregar a extensão, você deve primeiro carregar o assembler conforme explicado na seção 4.2. Em seguida, execute **RUN HGR.BLK** e finalmente (se aplicável) **DISPOSE** para remover o montador.

9.1 A TELA GRÁFICA

A tela gráfica ocupa 6144 bytes e está localizada no endereço 32768 (8000h), portanto sobrescrevendo a área do disco DE RAM. Se você tiver mais RAM disponível (com Chroma-81 por exemplo) você pode mover a tela de alta resolução para um endereço com alinhamento de 8192 bytes (A000h, C000h ou E000h são as opções disponíveis), basta editar o valor da constante hfile na tela 1 em HGR.BLK.

A tela gráfica tem resolução de 256x192 pixels, com a coordenada (0,0) localizada na parte inferior esquerda da tela e a coordenada (255.191) na parte superior direita. As palavras gráficas aceitam coordenadas além dos limites da tela, pixels fora da tela são simplesmente ignorados. Para impressão de texto, a tela segue o mesmo formato do terminal padrão, 24 linhas por 32 colunas com a linha 0 na parte superior da tela.

A tela gráfica pode ser manipulada a partir do terminal de texto ou ativando o terminal gráfico com a palavra **HTERM**. A vantagem do primeiro modo é que a tela gráfica não é poluída com os comandos fornecidos. Nas próximas seções veremos como usar os dois modos.

9.2 O TERMINAL DE TEXTO

É o terminal padrão que usamos até agora, não há muito o que adicionar, então vamos direto aos exemplos práticos:

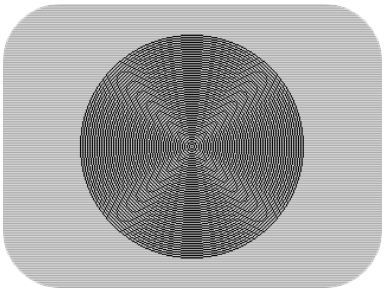
```
GPAGE HGR 1 GPEN 0 0 GSET 255 191 GLINE
```

GPAGE limpa a tela HR, **HGR** a ativa, **GPEN** define a ação dos comandos gráficos (acender pixel neste caso), **GSET** define o pixel na coordenada (0,0) e **GLINE** desenha uma linha da última coordenada até o ponto localizado 255 pixels à direita e 191 pixels acima.

Digite TEXT para retornar ao terminal padrão (você não verá o que está digitando, pois a saída de texto padrão ainda é a tela de texto).

Agora, para um exemplo mais elaborado, insira a seguinte definição (ou carregue GCIRCLE.BLK do diretório SCREENS):

```
\ MIDPOINT CIRCLE ALGORITHM
VALUE X   VALUE Y
VALUE X1  VALUE Y1
VALUE P   VALUE PY  VALUE PXY
: GCIRCLE  ( x y r -- )
     TO X1    TO Y    TO X
  0  TO P    0 TO Y1
  BEGIN X1 Y1 < NOT WHILE
     P  Y1 2* + 1+   TO PY
     PY X1 2* - 1+   TO PXY
     X  X1 +   Y Y1 +   GSET
     X  X1 -   Y Y1 +   GSET
     X  X1 +   Y Y1 -   GSET
     X  X1 -   Y Y1 -   GSET
     X  Y1 +   Y X1 +   GSET
     X  Y1 -   Y X1 +   GSET
     X  Y1 +   Y X1 -   GSET
     X  Y1 -   Y X1 -   GSET
     PY TO P   1 +TO Y1
     PXY ABS PY ABS < IF
        PXY TO P  -1 +TO X1 THEN
  REPEAT ;
```



Em seguida, digite no modo direto:

```
GPAGE 0 0 GSET
255 0 GLINE 0 191 GLINE
-255 0 GLINE 0 -191 GLINE
20 95 GSET
107 -75 GLINE 107 75 GLINE
-107 75 GLINE -107 -75 GLINE
127 95 45 GCIRCLE HGR
```

Legal, não é? ;-)

Por fim, um exemplo com **GLINETO**:

```
: GODSEYE  ( -- )
  76 0 DO 75 I -
     128 OVER 2* - 96 GSET
     128 OVER 24 +    GLINETO
     128 OVER 2* + 96 GLINETO
     128 I 96 +       GLINETO
     128 OVER 2* - 96 GLINETO
     DROP 3 +LOOP ;

HGR GPAGE GODSEYE
```

9.3 O TERMINAL GRÁFICO

O terminal gráfico é um dos recursos mais interessantes implementados pela extensão HGR e, uma vez ativado, as ações de **EMIT**, **AT**, **PAGE** e **SLOW** ocorrerão na tela gráfica,

utilizando a fonte de caracteres armazenada em 15360. Portanto, é uma excelente opção para quem não tem hardware disponível para redefinição de caracteres.

Para ativar o terminal gráfico, basta digitar **HTERM** e, se necessário, **PAGE** para limpar a tela. A tela HGR com o tradicional cursor piscante será apresentada e você poderá agir exatamente da mesma forma que no terminal de texto. Experimente digitar **VLIST** e verá que a resposta será a mesma obtida no terminal padrão, embora um pouco mais lenta (lembre-se, para cada caractere são 8 os bytes transferidos para a tela e o scroll move 6144 bytes contra apenas 793 da tela de texto).

Tudo o que pode ser feito no terminal padrão também pode ser feito no terminal gráfico, mas existem alguns cuidados a serem tomados:

- Palavras que mudam para o modo FAST (**BLOAD**, **BSAVE**, **FLOAD**, **FSAVE**, **GET**, **PUT**, etc) não retornam automaticamente à tela gráfica, exigindo o uso de **SLOW** ou **HGR** após o uso.
- A palavra KEY também perde o acesso à tela se usada no modo FAST mas não se desespere, basta digitar **SLOW** ou **HGR** para retornar.
- É possível usar o Editor, mas apenas as telas 7 e 9 estarão disponíveis para edição. Ainda é possível usar todo o disco de RAM para carregar telas do cartão SD com as palavras **GET** ou **RUN**, o efeito colateral é que a tela gráfica ficará poluída (nada que um **PAGE** posterior não resolva).
- O Extensão da impressora também pode ser usado com o terminal gráfico, mas você precisará executar **HTERM** após o uso para retornar à tela HR. Se preferir, redefina a palavra **>S** no arquivo **PRINTER.BLK**:

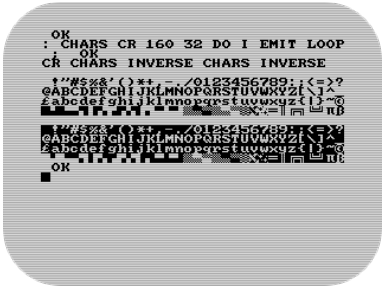
```
: >S  HTERM ;
```

- **COLD** retorna à tela de texto, mas **EMIT**, **PAGE**, **AT** e **SLOW** continuam atuando no display gráfico. Novamente, use **HGR** ou **SLOW** para retornar à tela HR.

Outra característica interessante é que a palavra **INVERSE** apresentada na seção 1.4.2 também funciona aqui e sem as restrições apresentadas anteriormente. Tente:

```
CHARS INVERSE CHARS INVERSE
```

(A definição do **CHARS** foi apresentada na seção 1.4.1)



9.4 ADICIONANDO ALGUMAS CORES

Para quem possui um Chroma-81, as palavras definidas abaixo permitem adicionar até três cores diferentes ao terminal gráfico: cor da borda, cor de fundo e cor de primeiro plano.

```
: BORDER ( n -- )
  48 + 32751 P! ;

: COLOR  ( n1 n2 -- )
  16 * + 32751 P@ 32 AND NOT IF
  [ coords 32774 + DUP 32 + ]
  LITERAL LITERAL DO DUP I C!
  LOOP THEN DROP ;
```

BORDER espera um número n na pilha que corresponde ao número da cor. Portanto, para definir a cor verde para a borda da tela, devemos inserir **4 BORDER**. Isso define a borda da cor e também ajusta a Chroma-81 para o modo de atributo de cor. Para desativar o recurso de cor, use **0 32751 P!**.

COLOR espera dois números na pilha, n1 e n2, que correspondem à cor de primeiro plano (ink) e cor de fundo (paper), respectivamente. Assim, **7 1 COLOR** atribuirá as cores branca à ink e azul a paper.

Consulte a lista de códigos de cores no capítulo 8.

Obs.: *Veja um exemplo de utilização de* **BORDER** *e* **COLOR** *no arquivo COMPART.BLK disponível na pasta SCREENS.*

9.5 PALAVRAS DE HGR

(SLOW) (--)
Rotina slow padrão.

GINVERT (--)
Inverte a tela gráfica.

GLINE (dx dy --)
Desenhe uma linha reta a partir do último ponto, passando por dx pixels para a direita e dy pixels para cima. Tanto dx quanto dy podem ser negativos, produzindo linhas indo para a esquerda e para baixo.

GLINETO (x y --)
Uma alternativa para **GLINE**, com esta palavra você especifica as coordenadas de destino da linha, que é desenhada para lá a partir do último ponto.

GMOVE (x y --)
Altere as coordenadas do "último ponto" para (x,y), sem afetar o pixel.

GPAGE (--)
Limpa a tela gráfica.

GPEN (n --)
Defina o comportamento de **GSET**, **GLINE** e **GLINETO**:
n = 0 --> apaga o pixel
n = 1 --> acende o pixel
n = 2 --> inverte o pixel

GPOS? (-- x y)
Obtenha as coordenadas do último ponto referido por qualquer outra palavra gráfica.

GSET (x y --)
Altera o ponto com as coordenadas (x,y) de acordo com a condição definida pelo **GPEN**.

GSET? (x y -- flag)
Deixa um flag verdadeiro se o pixel nas coordenadas (x,y) estiver aceso.

HGR (--)
Muda para a tela gráfica de alta-resolução.

HTERM (--)
Ativa o terminal gráfico e redireciona as ações de **SLOW**, **AT**, **EMIT** e **PAGE** para a tela de alta resolução.

SLOW (--)
Um vetor de execução, executa **(SLOW)** quando no terminal padrão ou **hslow** quando no terminal gráfico.

TERM (--)
Retorna ao terminal padrão restaurando as ações default de **SLOW**, **AT**, **EMIT** e **PAGE**.

TEXT (--)
Muda para a tela normal de texto.

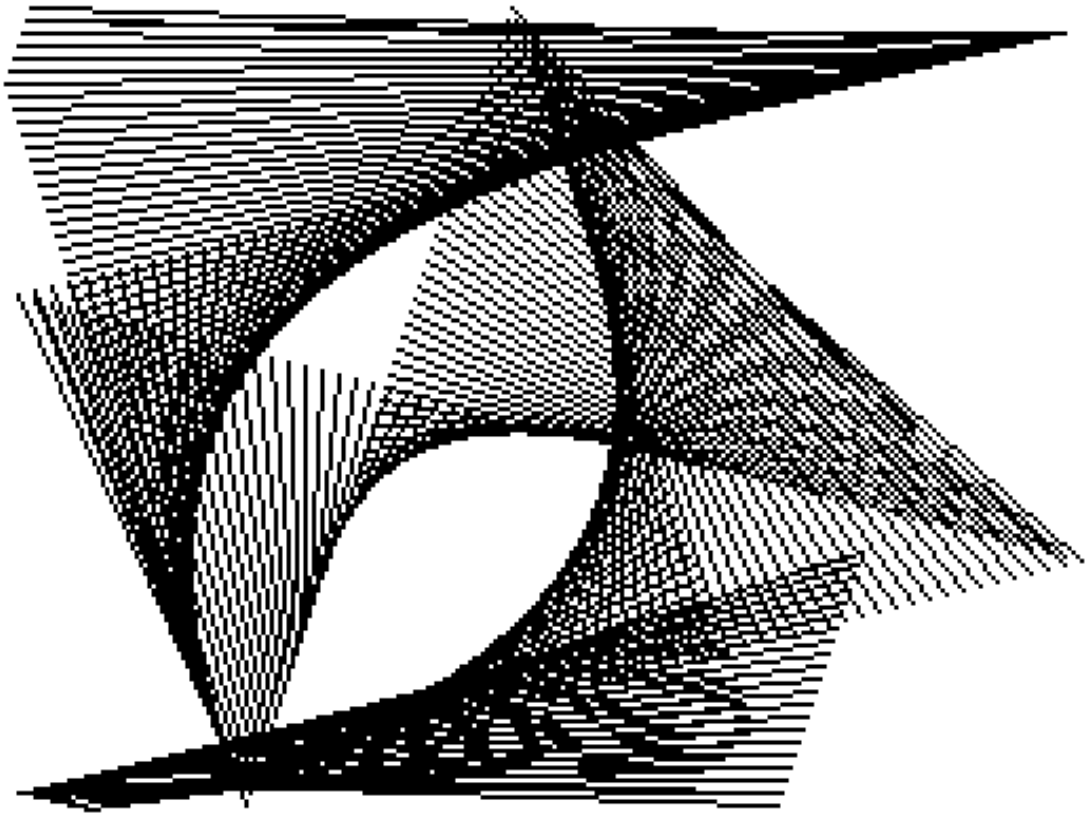
coords (-- addr)
Deixa o endereço do local de memória onde estão armazenadas as coordenadas do último pixel manipulado. Coordenada Y em addr, coordenada X em addr+2, 2 bytes para cada um.

hat (n1 n2 --)
Define a posição de impressão na tela gráfica para a linha n1 e a coluna n2.

hemit (c --)
Transmite o caractere c para o terminal gráfico.

hfile (-- addr)
Deixa o endereço da tela HGR

hslow (--)
Rotina do slow para a alta-resolução



Capítulo 10

10 MULTI-TASKING SUPPORT

Um dos novos recursos mais interessantes do TF79 1.13 é o suporte para programação multitarefa, embora isso não seja novidade no mundo Forth. Aqui foi adotado um modelo muito próximo ao usado por Pygmy Forth/Z88 Camel Forth.

O arquivo MULTI.BLK contém todas as palavras necessárias para implementar o ambiente Muti-tasking. Antes de carregá-lo, você deve primeiro carregar o assembler conforme explicado na seção 4.2. Em seguida, execute **RUN MULTI.BLK** e finalmente (se aplicável) **DISPOSE** para remover o assembler.

As instruções abaixo são em grande parte retiradas da documentação do Z88 Camel Forth v3.00 e adaptadas adequadamente para o TF79.

10.1 O AMBIENTE DE MULTI-TASKING

O TF79 inicia no modo de tarefa única; neste modo, uma única tarefa (a tarefa do terminal) está sendo executada, e a palavra de comutação de tarefa **PAUSE** é vetorizada para **NOOP** para que nenhum tempo seja desperdiçado "comutando" para a próxima tarefa.

No modo multitarefa, **PAUSE** faz com que a tarefa atualmente ativa mude o controle para a próxima tarefa "acordada" na lista circular. Você pode usar **PAUSE** especificamente ou confiar em uma das seguintes palavras, que o invocam:

- **KEY** - a cada loop de varredura do teclado
- **WAIT** - a cada 20ms

Apenas a tarefa do terminal deve aceitar entrada do teclado ou carregar outras fontes. Todas as áreas de dados são compartilhadas entre tarefas (como PAD, buffers de bloco, etc.), portanto, se você precisar usar essas áreas em mais de uma tarefa, certifique-se de salvá-las/restaurá-las conforme necessário antes e depois de cada possível alternância de tarefas. Cada tarefa tem suas próprias pilhas e configurações de **BASE**; todo o resto é compartilhado.

10.2 DEFINIÇÃO E EXECUÇÃO DE TAREFAS

A primeira coisa que você precisa fazer é definir suas tarefas com **TASK:**. Cada tarefa requer 266 bytes de RAM para armazenar suas pilhas e variáveis de usuário. Como as tarefas podem ser configuradas para executar qualquer palavra a qualquer momento, você

só precisa definir quantas forem executadas ao mesmo tempo (excluindo a tarefa do terminal, é claro). Por exemplo, defina uma tarefa chamada **A**:

```
TASK: A
```

A execução de **A** retorna um ID de tarefa exclusivo, que é um argumento ou resultado para a maioria das palavras de multiprogramação restantes.

Em seguida, defina a tarefa para executar uma palavra específica com **TASK!**. Normalmente, isso conterá um loop infinito com **PAUSE** em alguma posição estratégica. Como um exemplo:

```
: TEST  BEGIN  BELL PAUSE  AGAIN ;
' TEST A TASK!
```

Por fim, você pode habilitar a multitarefa com **MULTI** e definir a tarefa em execução com **WAKE**:

```
MULTI  A WAKE
```

10.3 CONTROLE DE TAREFAS

Você pode desabilitar a multitarefa a qualquer momento com **SINGLE**, que tem o efeito de bloquear o sistema na tarefa em execução no momento. Todas as tarefas anteriormente "acordadas" ainda estão ativadas e começarão a ser executadas novamente assim que a multitarefa for reativada com o **MULTI**.

Qualquer tarefa (exceto a tarefa do terminal) pode ser interrompida com **SLEEP**. Por exemplo, para interromper a tarefa **A**, faça:

```
A SLEEP
```

É possível re-**WAKE** a tarefa, ou usar **TASK!** primeiro para configurá-la para executar uma nova palavra. É perfeitamente seguro usar **WAKE** em uma tarefa já ativa ou **SLEEP** em uma tarefa inativa; no entanto, se você tentar usar o **TASK!** em uma tarefa ativa, pode ocorrer uma falha do sistema.

Se você deseja que uma tarefa seja executada apenas por um tempo especificado, inclua a palavra **STOP** em algum lugar na palavra em que ela é executada; isso colocará a tarefa atual em repouso e alternará para a próxima.

Finalmente, você pode usar **PURGE** para desativar todas as tarefas em execução.

10.4 MULTI-TASKING DEMO

Como teste, configure duas tarefas, cada uma das quais incrementa um contador associado. Um incrementa seu contador a cada 40ms e o outro uma vez a cada 400ms.

```
VARIABLE C1  VARIABLE C2
TASK: T1     TASK: T2
: TST1 BEGIN 2 WAIT ( 40 ms ) 1 C1 +! AGAIN ;
: TST2 BEGIN 20 WAIT ( 400 ms ) 1 C2 +! AGAIN ;
' TST1 T1 TASK!
' TST2 T2 TASK!
T1 WAKE  T2 WAKE  MULTI
```

Ocasionalmente verifique os valores do contador, por ex.

```
C1 ?
C2 ?
C1 ?
C2 ?
```

etc.

Em seguida, como teste adicional, configure mais duas tarefas que apenas exibem os valores dos contadores que as tarefas do bloco anterior estão incrementando:

```
TASK: T3     TASK: T4
: CUR@ 16398 @ ;
: CUR! 16398 ! ;
: TST3 BEGIN 1 WAIT ( 20 ms ) CUR@ 0 26 AT
  C1 @ 6 .R  CUR! AGAIN ;
: TST4 BEGIN 1 WAIT ( 20 ms ) CUR@ 2 26 AT
  C2 @ 6 .R  CUR! AGAIN ;
' TST3 T3 TASK!
' TST4 T4 TASK!
T3 WAKE   T4 WAKE
```

Observe como o endereço do cursor na tarefa do terminal é preservado em **TST3** e em **TST4**.

10.5 PALAVRAS DE MULTI

#TASKS (-- addr)
Uma variável que contém o número de tarefas atualmente ativas.

(PAUSE) (--)
Comuta para a próxima tarefa.

AWAKE? (task -- task'|0)
Retorna 0 se a tarefa especificada estiver inativa; se estiver ativo, o ID da tarefa anterior na fila é retornado.

LINK (-- task)
Retorna o ID da tarefa em execução no momento. Esse ID é, na verdade, o endereço de uma variável que contém o ID da próxima tarefa na fila.

MULTI (--)
Ativa a multitarefa.

MULTI-ABORT (--)
Versão especial do **ABORT** para que uma tarefa não-terminal entre em suspensão (com **STOP**) em vez de executar **QUIT**.

PURGE (--)
Mata todos os processos e retorna para **TERM**.

SINGLE (--)
Desativa a multitarefa.

SLEEP (task --)
Remova a tarefa da fila, a menos que seja **TERM**.

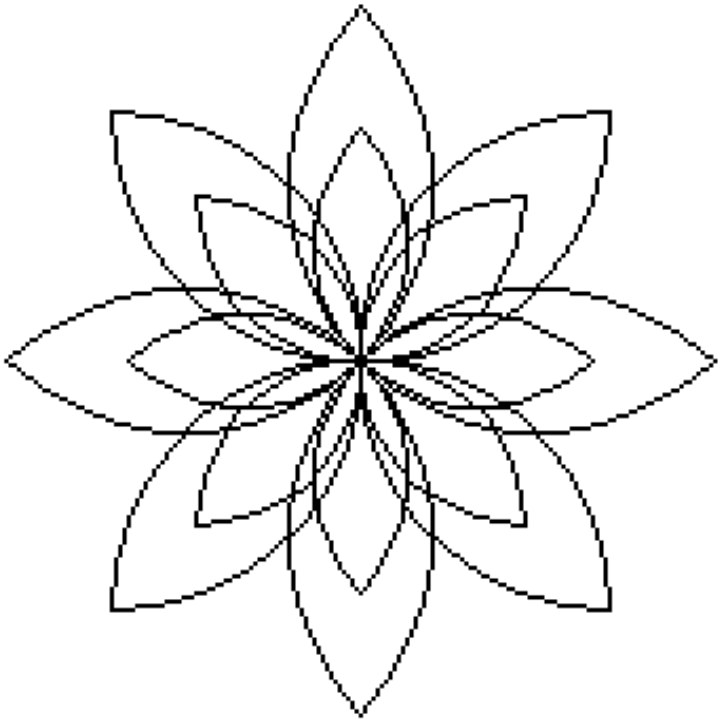
STOP (--)
Coloca a tarefa atual em repouso.

TASK! (addr task --)
Configure uma tarefa pronta para **WAKE**.

TASK: ("name" --)
Cria uma nova tarefa.

TERM (-- task)
Retorna o ID da tarefa do terminal.

WAKE (task --)
Insere a tarefa na fila.



Capítulo 11

11 DENTRO DO COMPILADOR

11.1 O INTÉRPRETE INTERNO

O interpretador interno adota o Direct Threaded Code (DTC), o que significa que ele salta diretamente para os endereços contidos em uma definição de dois pontos. Portanto, o campo de código de cada palavra forth contém instruções de máquina. Para definições em código, estas são as instruções de máquina para a definição. Para constantes, variáveis, definições de dois pontos e variáveis de usuário, o Code Field contém uma chamada de subrotina para a rotina de tempo de execução apropriada (DOCON para constantes, DOCREATE para variáveis, DOCOLON para definições de dois pontos, DODOES para a cláusula **DOES>** e DOUSER para variáveis de usuário). Para palavras criadas por **CREATE DOES>** o Campo de Código contém uma chamada de subrotina para o endereço imediatamente após **(;CODE)** na palavra definida. Uma chamada de sub-rotina para DODOES é compilada neste endereço. A primeira chamada coloca o Parameter Field na pilha, a segunda chamada inicia a execução da parte **DOES>** na palavra definida como uma definição de dois pontos.

Em todos os casos, a instrução CALL faz com que o Campo de Parâmetro seja empurrado para a pilha, tornando-o prontamente disponível para a rotina do manipulador (DOCON, DOCOLON, DODOES, DOUSER), ou apenas deixando-o lá no caso de variáveis.

Cada definição de código termina com um salto para a rotina NEXT, feito com a instrução JP (IY) (para as rotinas Forth mais importantes, NEXT é construído inline ao invés de ter um jumper para ela), onde o endereço da rotina NEXT é sempre armazenado no registrador IY. A rotina NEXT é o próprio interpretador interno. Ele lê o endereço da próxima palavra a ser executada (apontada pelo ponteiro de instrução, que é incrementado) e salta para ela.

A rotina NEXT consiste em 7 instruções, como segue:

```
NEXT:   EX DE,HL      ; Armazena o ponteiro de instrução em HL
        LD E,(HL)     ; Leia o próximo endereço, byte menos significativo
        INC HL
        LD D,(HL)     ; Leia o próximo endereço, byte mais significativo
        INC HL
        EX DE,HL      ; Ponteiro de instrução em DE, endereço de salto em HL
        JP (HL)       ; Salta para o endereço de execução
```

- O item no topo da pilha de dados (TOS) é armazenado em BC
- O registro de trabalho (W) é HL
- O ponteiro de instrução (IP) é DE
- O ponteiro da pilha de dados do Forth (PSP) é SP

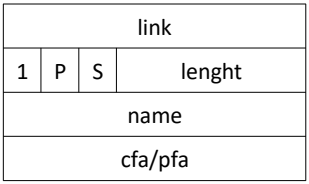
- O endereço NEXT é armazenado em IY
- O ponteiro da pilha de retorno (RSP) é armazenado no endereço 4007H
- O ponteiro da área do usuário (UP) é armazenado no endereço 400AH
- O registrador X não é usado nesta implementação. Normalmente é usado para acessar o campo de parâmetro da palavra que está sendo executada, mas este é empurrado na pilha pela instrução CALL no campo de código.

Como na maioria dos sistemas, tanto a pilha de dados quanto a pilha de retorno crescem para baixo. Sempre que nos referimos ao topo da pilha, significa que é o endereço de memória mais baixo e o fundo da pilha é o endereço de memória mais alto.

11.2 ESTRUTURA DO CABEÇALHO

As palavras no TF79 são compostas por quatro campos, são eles:

- **Link Field:** aponta para o Name Field da palavra anterior no dicionário.
  Na primeira palavra, este campo contém 0. O tamanho é sempre 2 bytes.
- **Name Field:**
  Primeiro byte:
  bit 7: sempre é 1
  bit 6: Precedence bit, 1 se é uma palavra imediata
  bit 5: Smudge bit, 1 se a palavra não for encontrável
  bit 4..0: comprimento do nome
  Os bytes seguintes contêm os caracteres do nome.
- **Code Field:** 3-byte instrução CALL
- **Parameter Field:** contém todas as informações adicionais usadas pela palavra, por exemplo, o valor armazenado em uma variável, a lista de endereços de execução em uma definição de dois pontos.



Obs.: *As definições em código não têm um campo de código e um campo de parâmetro separados. As instruções de máquina começam no endereço do campo de código e a definição de código contém tantos bytes de código de máquina conforme necessário. Palavras de vetores de execução são definições em código.*

FIND e **'** sempre retornam o endereço do Campo de Código, que é o mesmo que o endereço de compilação.

>NFA converte o endereço do Code Field para o endereço do Name Filed.
>PFA converte o endereço do Code Field para o endereço do Parameter Field.
>CFA converte o endereço do Name Field para o endereço do Code Field.

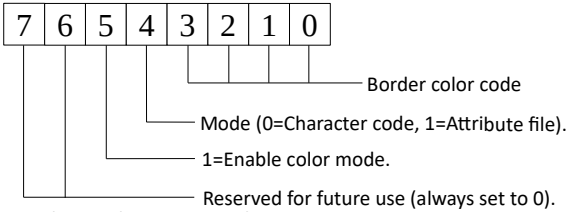
11.3 MAPA DE MEMÓRIA

O TF79 requer o uso da interface ZXpand com a memória RAM mapeada na faixa de 8K-40K. Para esta configuração, o uso de memória é o seguinte (endereços em decimal):

Obs.: *O TF79 é completamente independente do interpretador BASIC, então algumas variáveis do sistema BASIC foram substituídas por variáveis Forth.*

8192: Forth kernel
15360: Tabela de definição de caracteres
16384: Variáveis de Sistema BASIC e Forth:

- 16391: RSP - Ponteiro de pilha de retorno
- 16394: UP - ponteiro de área do usuário
- 16396: DFILE - endereço da tela
- 16398: CURADD - endereço do cursor da tela
- 16400: LKEY - última tecla lida
- 16401: REP - Contador para repetição de teclas
- 16402: HCURADD - Endereço do cursor na tela de HR
- 16417: FTFLAGS - Flags do Forth: CFxxSKxA
  - C = cursor on/off, 1=on
  - F = Fast mode flag
  - S = Stop on/off
  - K = caps lock flag
  - A = attribute color mode, 1=ativo
  - x = flag não usada
- 16418: CMBDR - Chroma color mode and border color



16419: ATTR - atributos de cor para ink e paper

16430: INVCHR - flag de inversão de caracter
16431: CTIMER - temporizador para cursor piscante
16444: PRTBUF

ÁREA DO USUÁRIO:
Variáveis por tarefa:
- 16477: LINK
- 16479: ‘SP
- 16481: S0
- 16483: R0
- 16485: BASE

Variáveis globais:
- 16487: DP
- 16489: FENCE
- 16491: STATE
- 16493: >IN
- 16495: HLD
- 16497: VOC-LINK
- 16499: CONTEXT
- 16501: CURRENT
- 16503: DPL
- 16505: BLK
- 16507: SCR

16509: Linhas BASIC para carregar o kernel do Forth

```
1 CONFIG "M=L"
2 LOAD "TF79.BIN;8192"
3 RAND USR 0
```

16555: Arquivo de tela (D-File)
17350: A palavra **FORTH**
17371: O dicionário do usuário começa aqui
31217: Fim do dicionário
31485: Início do buffer do terminal de entrada (TIB, 128 bytes) e base da pilha de dados (S0), que cresce para baixo e tem 256 bytes de tamanho. Por segurança, há um intervalo de 12 bytes entre a pilha de dados e o final do dicionário.
31741: Base da pilha de retorno (R0), cresce em direção ao TIB e tem 128 bytes.
31743: Buffer de bloco e sua marcação (qual tela está contida nele).
32768: Início do disco de RAM, 8192 bytes
40959: Última posição da RAM.

11.4 MUDANDO O MAPA DE MEMÓRIA

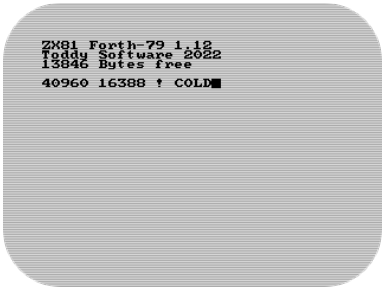
O mapa de memória padrão pode ser modificado conforme necessário alterando o valor da RAMTOP e **#SCR** e reiniciando Forth. Por exemplo, se você precisar de mais 2Kb de RAM para o dicionário, você pode ajustar o valor de RAMTOP para o endereço 34816, modificar o conteúdo da constante **#SCR** para refletir a nova capacidade do disco de RAM e finalmente executar **COLD** para reiniciar o Forth:

```
34816 16388 !  6 TO #SCR  COLD
```

Conseqüentemente, as localizações do TIB, da pilha de dados, da pilhas de retorno e do buffer de bloco também serão alteradas. **COLD** sempre define a variável **LO** (o endereço inicial do disco RAM) para o endereço fornecido pela RAMTOP.

Ao conectar uma interface Chroma 81 ao sistema, você ganha mais flexibilidade no uso da memória. Neste caso, você pode transferir o disco de RAM para a memória no endereço 49152 deixando os 8 Kb a partir de 32768 para o dicionário ou outra finalidade:

```
40960 16388 !  COLD  49152 LO !
```

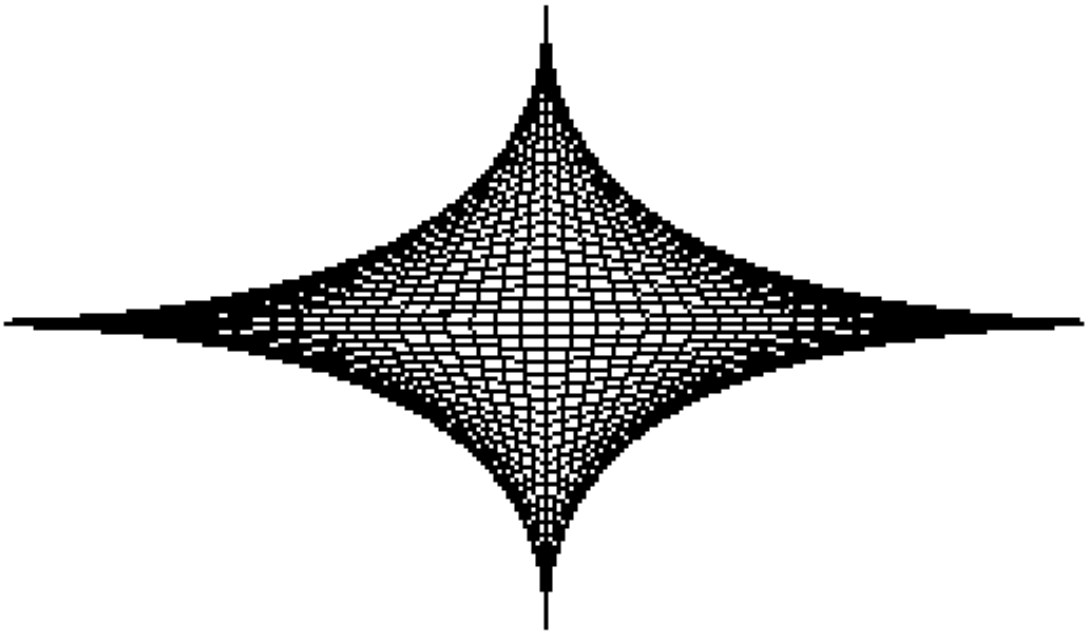


Neste caso não seria necessário alterar o conteúdo do **#SCR** pois o disco de RAM não altera sua capacidade original de 8 telas. Mas se for utilizar o Editor em modo **FILE**, então faz-se necessário atualizar a variável **DRLIM** para apontar para a primeira posição após o disco de RAM:

```
57344 DRLIM !
```

Não há restrições nas definições de dois pontos criadas acima do endereço 32767, mas as definições de código nesta área da memória devem ser evitadas, pois apenas um número limitado de instruções de máquina é permitido alí. O uso de uma instrução de máquina proibida fatalmente causará uma falha no sistema.

Um último lembrete: se você salvar um programa cujo dicionário exceda o endereço 32767, você deve alterar a RAMTOP para o valor apropriado antes de carregá-lo do BASIC.



Apêndice A

APPENDIX A - O CONJUNTOS DE PALAVRAS IMPLEMENTADAS

A.1 A NOTAÇÃO DE PILHA

As palavras na lista estão em ordem ASCII e o nome da palavra é seguido por qualquer valor que a palavra espera na pilha, seguido por dois hifens, seguido por qualquer valor retornado na pilha.

Depois disso, um ou mais símbolos em maiúsculas que indicam o atributo da palavra podem ser seguidos:

F79 A palavra é parte do Conjunto de Palavras Obrigatórias do Forth-79
R A palavra faz parte do Conjunto de Palavras de Referência do Forth-79
D A palavra faz parte do Conjunto de Palavras de Número Duplo do Forth-79
(não totalmente implementado)
C A palavra só pode ser usada durante a compilação de uma definição de dois pontos.
I Indica que a palavra é IMEDIATA e será executada durante a compilação,
a menos que uma ação especial seja tomada.
U Uma variável do usuário.

Os valores na pilha (esperados e retornados) são representados da seguinte forma:

addr um endereço de memória
b um valor que representa um byte de 8 bits
c um valor que representa um código de caractere ASCII de 8 bits
d número 'duplo' de 32 bits com sinal {-2.147.483.648..2.147.483.647}
f um valor numérico com dois estados lógicos; 0 = falso,
diferente de zero = verdadeiro.
n número inteiro de 16 bits com sinal {-32.768..32.767}
u número inteiro de 16 bits sem sinal {0..65535}
ud número inteiro de 32 bits sem sinal {0..4.294.967.295}
x qualquer valor de célula

A.2 DEFINIÇÃO DE TERMOS:

buffer: buffer de 1024 bytes que contém a tela atualmente acessada. Este sistema Forth contém um buffer de bloco único. As telas são movidas entre o buffer de bloco e o disco de RAM.

definição de dois pontos: palavra Forth cuja definição foi iniciada por '**:**'. Quando a definição de dois pontos é executada, o interpretador interno executa sucessivamente as palavras contidas na definição de dois pontos. O endereço do qual a definição de dois pontos foi invocada é armazenado na pilha de retorno.

compilação: construindo uma definição de dois pontos adicionando endereços de compilação e literais ao dicionário e executando palavras imediatas.

Endereço de compilação: o endereço de uma palavra Forth que será adicionada ao dicionário durante a compilação.

string contada: string de caracteres ASCII armazenados na memória, precedidos por um único byte contendo o comprimento da string.

dicionário: coleção de vocabulários, que contém todas as palavras Forth. O dicionário ocupa uma área de memória contígua, que é estendida ou reduzida em tamanho a partir da extremidade superior.

Palavra imediata: palavra Forth que sempre é executada, mesmo que o interpretador de texto esteja em estado de compilação.

intérpretador interno: pedaço de código de máquina que executa as palavras contidas em uma definição de dois pontos.

Buffer de entrada: buffer contendo a linha que é digitada no teclado.

fluxo de entrada: texto lido pelo interpretador de texto, seja do buffer de entrada ou de um buffer de bloco.

interpretação: procurando palavras do fluxo de entrada no dicionário e executando-as imediatamente. Quando uma palavra não é encontrada no dicionário, é feita uma tentativa de convertê-la em um número e o valor resultante é colocado na pilha se isso for bem-sucedido.

literal: uma palavra especial dentro de uma definição de dois pontos que, quando executada, empurra o valor imediatamente a seguir na pilha.

loop: estrutura de repetição usada dentro de uma definição de dois pontos, usando um valor terminal (limite) e um contador (índice) que são armazenados na pilha de retorno.

conversão numérica: conversão de um número de representação binária interna para uma cadeia de caracteres ASCII representando o número em forma legível por humanos.

pilha de retorno: pilha LIFO (last in first out) contendo endereços de retorno de definições de dois pontos, bem como outros valores, como contadores de loop e limites.

parte de tempo de execução: palavra que é adicionada à definição de dois pontos por uma palavra imediata durante a compilação. Quando a definição de dois pontos for executada, a parte do tempo de execução será executada.

screen: bloco de 1024 bytes geralmente contendo o texto fonte do programa Forth. As telas são armazenadas em um disco de RAM

pilha: pilha de dados LIFO (last in first out) central para a operação do Forth, na qual os valores são passados entre as palavras.

intérprete de texto: palavra Forth que lê palavras do fluxo de entrada e as interpreta ou compila, dependendo da variável **STATE**.

variável do usuário: Variável armazenada em uma área de memória cujo endereço inicial pode ser alterado. Com a multitarefa, cada tarefa tem sua própria versão das variáveis do usuário.

vocabulário: lista de palavras Forth.

A.3 PALAVRAS NO VOCABULÁRIO FORTH

! n addr -- F79,112 "store"
Armazena n em addr.

ud1 -- ud2 F79,158 "sharp"
Gere a partir de um número duplo sem sinal d1, o próximo caractere ASCII que é colocado em uma string de saída. O resultado d2 que é o quociente após a divisão por BASE, é mantido para processamento adicional. Usado entre <# e #>.

#> d -- addr n F79,190 "sharp-greater"
Fim da conversão de saída numérica ilustrada. Descarta d, deixando o endereço do texto e a contagem de caracteres, adequados para **TYPE**.

#S ud -- 0 0 F79,209 "sharp-s"

Converte todos os dígitos de um número ud de 32 bits sem sinal, adicionando cada um para o texto representativo da saída numérica até que o restante seja zero. Um único zero é adicionado à string de saída se o número era inicialmente zero. Use apenas entre **<#** e **#>**.

#SCR -- n

Retorna o número de telas no disco de RAM.

' -- addr F79,I,171 "tick"

Usado na forma:

' <name>

Se estiver executando, deixe o endereço do campo de código da próxima palavra aceita do canal de entrada. Se estiver compilando, compile este endereço como um literal; a execução posterior colocará este valor na pilha. Existe uma condição de erro se não for encontrada após uma pesquisa dos vocabulários **CONTEXT** e **FORTH**. Dentro de uma definição de dois pontos **'** <nome> é idêntico a **['** <name> **] LITERAL**.

(-- F79,I,122 "paren"

Usado na forma:

(ccc**)**

Aceita e ignora caracteres de comentário do canal de entrada, até o próximo parêntese direito. Como uma palavra, o parêntese à esquerda deve ser seguido de um espaço em branco. Pode ser usado livremente durante a execução ou compilação. Existe uma condição de erro se o canal de entrada se esgotar antes do parêntese direito.

('') -- addr

Lê a palavra do fluxo de entrada, procura-a no dicionário e retorna o endereço de compilação ou a mensagem de erro se a palavra não for encontrada. Usado na definição de **'**.

(;CODE) --

Rotina de execução parte de **DOES>** Só pode ocorrer em uma definição de dois pontos. Quando executado, a definição de dois pontos é encerrada e o endereço seguinte a **(;CODE)** é colocado no campo de código da última definição criada.

(EMIT) c --

Código padrão executado por **EMIT**.

(ERRNUM) f --
Código padrão executado por **ERRNUM**.

***** n1 n2 -- n3 F79,138 "times"
Deixa o produto aritmético de n1 vezes n2.

***/** n1 n2 n3 -- n4 F79,220 "times-divide"
Multiplique n1 por n2, divida o resultado por n3 e deixe o quociente n4. n4 é arredondado em direção a zero. O produto de n1 por n2 é mantido como um valor intermediário de 32 bits para precisão maior do que a sequência equivalente: n1 n2 ***** n3 **/**

***/MOD** n1 n2 n3 -- n4 n5 F79,192 "times-divide-mod"
Multiplique n1 por n2, divida o resultado por n3 e deixe o resto n4 e quociente n5. Um produto intermediário de 32 bits é usado como em ***/** . O resto tem o mesmo sinal de n1.

+ n1 n2 -- n3 F79,121 "plus"
Deixa a soma aritmética de n1 mais n2.

+! n addr -- F79,157 "plus-store"
Adicione n ao valor de 16 bits no endereço, pela convenção dado para **+**.

+- n1 n2 -- n3
Negue n1 se n2 for negativo, deixando o resultado como n3.

+LOOP n -- F79,I,C,141 "plus-loop"
Adicione o incremento com sinal n ao índice de loop usando a convenção para **+** e compare o total com limite. Retorne a execução para o **DO** correspondente até que o novo índice seja igual a ou maior que o limite (n>0), ou até que o novo índice seja menor que o limite (n<0). Ao sair do loop, descarte os parâmetros de controle do loop, continuando a execução à frente. Índice e limite são inteiros com sinal no intervalo {-32768..32767}.

+TO n --
Adicione n ao valor no campo de parâmetro de <nome>. Executado na forma:
n **+TO** <name>

, n -- F79,143 "comma"
Aloque dois bytes no dicionário, armazenando n lá.

- n1 n2 -- n3 F79,134 "minus"
Subtraia n2 de n1 e deixe a diferença n3.

--\> F79,R,I,131 "next-block"
Continue a interpretação no próximo bloco sequencial. Pode ser usado dentro de uma definição de dois pontos que cruza o limite de um bloco.

-ROT n1 n2 n3 -- n3 n1 n2
Move o topo da pilha para a terceira posição.

-TRAILING addr n1 -- addr n2 F79,148 "dash-trailing"
Ajuste a contagem de caracteres n1 de uma string de texto começando em addr para excluir espaços em branco à direita, ou seja, os caracteres em addr+n2 a addr+n1-1 são espaços em branco. Existe uma condição de erro se n1 é negativo.

. n -- F79,193 "dot"
Exibe n convertido de acordo com **BASE** em um formato de campo livre com um espaço em branco à direita. Exibe apenas um sinal negativo.

." F79,I,133 "dot-quote"
Interpretado ou usado na forma:
." ccc**"**
Aceita o texto seguinte no canal de entrada, terminado por " (aspas). Se estiver executando, transmite este texto para o dispositivo de saída selecionado. Se estiver compilando, compila para que mais tarde a execução transmita o texto para a saída do dispositivo selecionado. São permitidos pelo menos 127 caracteres no texto. Se o canal de entrada se esgota antes das aspas, existe uma condição de erro.

.R n1 n2 -- F79,R "dot-r"
Imprime n1 alinhado à direita em um campo de n2 caracteres, de acordo com BASE. Se n2 for menor que 1, nenhum espaço em branco à esquerda será fornecido.

/ n1 n2 -- n3 F79,178 "divide"
Divide n1 por n2 e deixa o quociente n3. n3 é arredondado em direção a zero.

/MOD n1 n2 -- n3 n4 F79,198 "divide-mod"
Divide n1 por n2 e deixa o resto n3 e o quociente n4. n3 tem o mesmo sinal de n1.

0 -- 0
Constante 0

0< n -- f F79,144 "zero-less"
Verdadeiro se n for menor que zero (negativo)

0= n -- f F79,180 "zero-equals"
Verdadeiro se n for zero.

0> n -- f F79,118 "zero-greater"
Verdadeiro se n for maior que zero.

1 -- 1
Constante 1

1+ n -- n+1 F79,107 "one-plus"
Incrementa n em um, de acordo com a operação de **+** .

1- n -- n-1 F79,105 "one-minus"
Decrementa n em um, de acordo com a operação de **-** .

2 -- 2
Constante 2

2! d addr – F79,D "two-store"
Armazena d em 4 bytes consecutivos começando em addr, como para um número duplo.

2* n1 -- n2 F79,R "two-times"
Multiplica n1 por 2 .

2+ n -- n+2 F79,135 "two-plus"
Incrementa n em dois, de acordo com a operação de **+** .

2- n -- n-1 F79,129 "two-minus"
Decrementa n em dois, de acordo com a operação de **-** .

2/ n1 -- n2 F79,R "two-divide"
Divide n1 por 2.

2@ addr -- d F79,D "two-fetch"
Deixa na pilha o conteúdo dos quatro bytes consecutivos começando em addr, como para um número duplo.

2DROP d -- F79,D "two-drop"
Descarte o número duplo do topo da pilha.

2DUP d -- d d F79,D "two-dupe"
Duplique o número duplo no topo da pilha.

2OVER d1 d2 -- d1 d2 d1 F79,D "two-over"
Deixe uma cópia do segundo número duplo na pilha.

2SWAP d1 d2 -- d2 d1 F79,D "two-swap"
Permute os dois primeiros números duplos da pilha.

3 -- 3
Constante 3

79-STANDARD F79,119
Execute garantindo que um sistema FORTH-79 Standard esteja disponível, caso contrário existe uma condição de erro.

: F79,116 "colon"
Uma palavra definidora executada na forma:
: <name> ... **;**
Selecione o vocabulário **CONTEXT** para ser idêntico ao **CURRENT**. Crie uma entrada de dicionário para <name> em **CURRENT** e defina o modo de compilação. Palavras assim definidas são chamadas de 'definições de dois pontos'. Os endereços de compilação das palavras subsequentes do fluxo de entrada que não são palavras imediatas são armazenados no dicionário para serem executados quando <nome> for executado posteriormente. Palavras **IMMEDIATE** são executadas conforme encontradas.
Se uma palavra não for encontrada após uma busca nos vocabulários **CONTEXT** e **FORTH**, tenta-se a conversão e compilação de um número literal, em relação à **BASE** atual; isso falhando, existe uma condição de erro.

; F79,I,C,196 "semi-colon"
Termina uma definição de dois pontos e para a compilação. Se compilando a partir do armazenamento em massa e o fluxo de entrada se esgota antes de encontrarr **;** xiste uma condição de erro.

< n1 n2 -- f F79,139 "less-than"
Verdadeiro se n1 for menor que n2.

<# F79,169 "less-sharp"

Inicializa a saída numérica ilustrada. As palavras:

```
# #> #S <# HOLD SIGN
```

podem ser usadas para especificar a conversão de um número de precisão dupla em uma string de caracteres ASCII armazenada na ordem da direita para a esquerda.

<CMOVE addr1 addr2 n -- F79,R "reverse-c-move"

Copie n bytes começando em addr1 para addr2. A movimentação procede dos bytes da memória alta para a memória baixa.

= n1 n2 -- f F79,173 "equals"

Verdadeiro se n1 for igual a n2.

> n1 n2 -- f F79,102 "greater-than"

Verdadeiro se n1 for maior que n2.

>CFA nfa -- cfa

Converte o 'name field address' de uma definição em seu 'code field address'.

>IN -- addr F79,U,201 "to-in"

Deixa o endereço da variável que contém o presente deslocamento de caracteres dentro do canal de entrada {{0..1023}}

Veja: **WORD (." FIND**

>NFA cfa -- nfa

Converte o 'code field address' de uma definição em seu 'name field address'.

>PFA cfa -- pfa

Converte o 'code field address' de uma definição em seu 'parameter field address'.

>R n -- F79,C,200 "to-r"

Transfere n para a pilha de retorno. Cada **>R** deve ser equilibrado por um **R>** no mesmo nível de aninhamento da estrutura de controle da definição de dois pontos.

? addr -- F79,194 "question-mark"

Exibe o número no endereço, usando o formato de "**.**" .

?DUP n -- n (n) F79,184 "query-dupe"

Duplique n se ele for diferente de zero.

?TERMINAL -- f

Executa um teste do teclado para acionamento da tecla BREAK. Retorna verdadeiro em caso de atuação.

@ addr -- n F79,199 "fetch"

Deixe na pilha o número contido em addr.

ABORT F79,101

Limpe as pilhas de dados e de retorno, configurando o modo de execução. Retorne o controle ao terminal.

ABORT" f -- F79,R,I,C "abort-quote"

Usado em uma definição de dois pontos na forma:

```
ABORT" stack empty"
```

Se flag for verdadeiro, imprima o texto que se segue, até **"**. Então execute **ABORT**.

ABS n1 -- n1 F79,108 "absolute"

Deixa o valor absoluto de um número.

ADDR n1 -- addr

Retorna o endereço da tela n no disco de RAM.

AGAIN F79,R,I,C,114

Efetua um salto incondicional de volta ao início de um loop **BEGIN-AGAIN** .

ALLOT n -- F79,154

Adiciona n bytes ao campo de parâmetro da mais recente palavra definida.

AND n1 n2 – n3 F79,183

Deixe o 'and' lógico bit a bit de n1 e n2.

ASCII -- c (executando) F79,R,I,C
-- (compilando)

Deixa o código ASCII do próximo caractere no canal de entrada diferente de espaço. Se estiver compilando, compile-o como um literal, que será deixado posteriormente quando executado.

AT u1 u2 --

Posiciona o cursor na linha u1 e na coluna u2. **0 0 AT** é a linha mais ao alto e a coluna mais à esquerda.

B/BUF -- 1024 F79,R "bytes-per-buffer"

Uma constante que deixa 1024, o número de bytes por buffer de bloco.

B/SCR -- 1

Constante, o número de buffers de bloco por tela.

BASE -- addr F79,U,115

Deixa o endereço da variável contendo a base de conversão numérica atual. {{2..70}}

BEGIN F79,I,C,147

Usado em uma definição de dois pontos na forma:

BEGIN ... flag **UNTIL** ou

BEGIN ... flag **WHILE** ... **REPEAT**

BEGIN marca o início de uma sequência de palavras para execução repetitiva. Um laço **BEGIN**-**UNTIL** será repetido até que o flag seja verdadeiro. Um loop **BEGIN**-**WHILE**-**REPEAT** será repetido até que o flag seja falso. As palavras após **UNTIL** ou **REPEAT** serão executadas quando qualquer loop for concluído. flag é sempre descartado após ser testado.

BELL F79,R

Emite um som de 1316Hz com duração de 40ms.

BL -- n F79,R,176 "b-l"

Deixa o valor ASCII para o caractere espaço (decimal 32).

BLK -- addr F79,U,132 "b-l-k"

Deixa o endereço da variável contendo o número do bloco de armazenamento em massa sendo interpretado como o canal de entrada. Se o conteúdo é zero, o canal de entrada é tomado do terminal.

BLANKS addr n -- F79,R,152

Preenche uma área de memória de n bytes com o valor ASCII para espaço, começando em addr. Se n for menor ou igual a zero, nenhuma ação é tomada.

BLOCK n -- addr F79,191

Faz com que a tela n seja carregada no buffer de bloco e retorna o endereço desse buffer de bloco. Qualquer tela contida no buffer de bloco que foi alterada, será primeiro copiada de volta para o disco de RAM.

BLOAD addr --

Lê uma palavra do canal de entrada e carrega do armazenamento em massa para a RAM no endereço addr, o arquivo com esse nome.

BSAVE addr u --

Lê uma palavra do canal de entrada e salva u bytes a partir do endereço addr para o armazenamento em massa.

BUFFER n -- addr F79,130

Cria um buffer de bloco vazio para a tela n (sem lê-lo do disco de RAM) e retorna seu endereço.

C! n addr -- F79,219 "c-store"

Armazena os 8 bits menos significativos de n em addr.

C, n -- F79,R "c-comma"

Armazena os 8 bits menos significativos de n no próximo byte no dicionário, avançando o ponteiro do dicionário.

C/L -- u

Retorna o número de caracteres por linha.

C@ addr -- b F79,156 "c-fetch"

Deixe na pilha o conteúdo do byte em addr (com bits mais significaticos zero, em um campo de 16 bits).

CAT --

Mostra a lista de diretórios na ZXpand(+).

CHR$128 n --

Se n = 1, será definido o modo de caracteres CHR$128. Se n=0, será definido o modo de caracteres CHR$64. Se n<>0 e par, existe uma condição de erro.

CMOVE addr1 addr2 n -- F79,153 "c-move"

Move n bytes começando no endereço addr1 para addr2. O conteúdo de addr1 é movido primeiro em direção ao alto da memória. Se n for zero, nada é movido.

COLD --

Inicialização a frio de FORTH. Ele executa as seguintes ações, em adição a **WARM**:

- Configura a RAM a partir do valor na variável de sistema RAMTOP.

- Remove todas as palavras do dicionário em um endereço superior ao dado por **FENCE**.
- inicializa todas as variáveis de usuário relevantes.

COMPILE F79,C,146

Quando uma palavra contendo **COMPILE** é executada, o valor de 16 bits seguindo o endereço de compilação de **COMPILE** é copiado (compilado) no dicionário. Ou seja, **COMPILE DUP** copiará o endereço de compilação de **DUP**.

CONSTANT n -- F79,185

Uma palavra de definição usada na forma:

n **CONSTANT** <name>

para criar uma entrada de dicionário para <nome>, deixando n em seu campo de parâmetro. Quando <nome> for executado posteriormente n será deixado na pilha.

CONTEXT -- addr F79,U,151

Deixa o endereço da variável especificando o vocabulário no qual as pesquisas de dicionário devem ser feitas, durante a interpretação do canal de entrada.

CONVERT d1 addr1 -- d2 addr2 F79,195

Converte para o número equivalente na pilha o texto começando em addr1+1, em relação a **BASE**. O novo valor é acumulado no número duplo d1, sendo deixado como d2. addr2 é o endereço do primeiro caracter não conversível.

COUNT addr -- addr+1 n F79,159

Deixa o endereço addr+1 e a contagem de caracteres do texto começando em addr. O primeiro byte em addr deve conter a contagem de caracteres n. Intervalo de n é {0..255}.

CR F79,160 "c-r"

Faz com que ocorra um retorno de carro e alimentação de linha no dispositivo de saída corrente.

CREATE F79,239

Uma palavra de definição usada na forma:

CREATE <name>

para criar uma entrada de dicionário para <nome>, sem alocar qualquer campo de parâmetro na memória. Quando <name> é subsequentemente executado, o endereço do primeiro byte do campo de parâmetro de <nome> é deixado na pilha.

CURRENT -- addr F79,U,137

Deixa o endereço da variável especificando o vocabulário no qual novas definições de palavras devem ser inseridas.

D+ d1 d2 -- d3 F79,241 "d-plus"

Deixe a soma aritmética de d1 mais d2.

D+- d1 n -- d2

Negue d1 se n for negativo, deixando o resultado como d2.

D- d1 d2 -- d3 F79,D,129 "d-minus"

Subtrai d2 de d1 e deixa a diferença d3.

D. d -- F79,D,129 "d-dot"

Exibe d convertido de acordo com **BASE** em um formato de campo livre, com um espaço em branco à direita. Mostra o sinal apenas se for negativo.

D.R d n -- F79,D "d-dot-r"

Exibe d convertido de acordo com **BASE**, alinhado à direita em um campo de caracteres. Mostra o sinal apenas se for negativo.

D< d1 d2 -- f F79,244 "d-less-than"

Verdadeiro se d1 for menor que d2.

DABS d1 -- d2 F79,D "d-absolute"

Deixa como um número duplo positivo d2, o valor absoluto de um número duplo, d1. {0..2,147,483,647}

DECIMAL F79,197

Defina a base de conversão numérica de entrada-saída para dez.

DEFINITIONS F79,155

Define **CURRENT** para o vocabulário **CONTEXT** tal que as definições serão criadas no vocabulário previamente selecionado como **CONTEXT**.

DELETE

Lê uma palavra do fluxo de entrada e exclui o arquivo com aquele nome.

DEPTH -- n F79,238

Deixa o número da quantidade de valores de 16 bits contidos na pilha de dados, antes de n ser adicionado.

DLITERAL d -- d (executando) FIG
d -- (compilando)

Se estiver compilando, compila um número duplo da pilha em um literal. A execução posterior da definição contendo o literal o colocará na pilha. Se estiver executando, o número permanecerá na pilha.

DNEGATE d -- -d F79,245 "d-negate"

Deixa o complemento de dois de um número duplo.

DO n1 n2 – F79,I,C,142

Usado em uma definição de dois pontos:

DO ... **LOOP** or

DO ... **+LOOP**

Comece um loop que terminará com base nos parâmetros de controle. O índice do loop começa em n2 e termina com base no limite n1. Em **LOOP** ou **+LOOP**, o índice é modificado por um valor positivo ou negativo. O alcance de um **DO-LOOP** é determinado pela palavra terminante. **DO-LOOP** pode ser aninhado. Capacidade para três níveis de aninhamento são especificados como um mínimo para sistemas padrão.

DOES> F79,I,C,168 "does"

Define a ação em tempo de execução de uma palavra criada por uma palavra definidora de alto nível. Usado na forma:

: <name> ... **CREATE** ... **DOES>** ... **;**

e então <namex> <name>

Marca o término da parte definidora da palavra definidora <nome> e começa a definição da ação em tempo de execução para palavras que posteriormente serão definidas por <nome>. Na execução de <namex> a sequência de palavras entre **DOES>** e **;** serão executadas, com o endereço do campo de parâmetro de <namex> na pilha.

DP -- addr FIG,U

Uma variável de usuário, o ponteiro do dicionário, que contém o endereço da próxima posição de memória livre acima do dicionário. O valor pode ser lido por **HERE** e alterado por **ALLOT**.

DPL -- addr F79,R "d-p-l"

Uma variável de usuário contendo o número de dígitos à direita do ponto decimal na entrada de número inteiro duplo. Um valor de -1 para **DPL** indica que não há ponto decimal e o número é reconhecido como um inteiro de 16 bits.

DROP n -- F79,233

Descarta o número do topo da pilha.

DUP n -- n n F79,205 "dupe"

Deixa uma cópia do número no topo da pilha.

ELSE -- F79,I,C,167

Usado em uma definição de dois pontos na forma:

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

ELSE é executado após a parte verdadeira seguinte a **IF**. **ELSE** força a execução a saltar até logo após **THEN**. Não tem efeito sobre a pilha. (Veja **IF**)

EMIT c -- F79,207

Transmite o caractere para o dispositivo de saída atual.

EMPTY-BUFFERS F79,145

Marca o buffer do bloco como vazio. **FLUSH** não o copiará para o disco de RAM.

ERASE addr n -- F79,R,182

Preenche uma área de memória de n bytes com zeros, começando em addr. Se n for zero ou menos, não executa nenhuma ação.

ERRNUM -- addr

Executado se **NUMBER** encontrar um erro. É um vetor de execução, portanto, a palavra **NUMBER** pode ser estendida para analisar outros tipos de dados, por exemplo, números de ponto flutuante.

EXEC:

Uma palavra definidora executada na forma:

EXEC: <name>

para criar uma palavra de vetor de execução, inicialmente designada para executar **NOOP**. A ação de <name> pode então ser atribuída à palavra <action> com o uso da palavra **IS**, como segue:

' <action> **IS** <name>

EXECUTE addr -- F79,163

Executa a entrada do dicionário cujo endereço de compilação está na pilha.

EXIT F79,C,117

Quando compilado dentro de uma definição de dois pontos, termina a execução dessa definição, nesse ponto. Não pode ser usado dentro de um **DO**...**LOOP**.

EXPECT addr n -- F79,189

Transfere caracteres do terminal começando em addr, para cima, até que um "retorno" ou a contagem de n caracteres seja recebida. Não executa nenhuma ação para n menor ou igual a zero. Um ou dois nulos são adicionados no final do texto.

FAST --

Define o modo **FAST**. O processamento ocorre sem gerar vídeo, mas o **KEY** passa automaticamente para o modo **SLOW** para permitir a visualização do que é digitado, retornando ao modo **FAST** ao finalizar.

FENCE -- addr FIG,U

Uma variável de usuário contendo um endereço abaixo do qual **FORGET** é travado. Para esquecer abaixo deste ponto, o usuário deve alterar o conteúdo de **FENCE**.

FILL addr n b -- F79,234

Preenche a memória começando no endereço com uma sequência de n cópias de byte. Se a quantidade n for menor ou igual a zero, não faz nada.

FIND -- addr F79,203

Deixa o endereço de compilação da próxima palavra aceita do fluxo de entrada. Deixa zero se essa palavra não puder ser encontrada no dicionário após uma pesquisa em **CONTEXT** e **FORTH**.

FIRST -- n FIG

Uma constante que deixa o primeiro endereço do buffer de bloco.

FLOAD --

Lê uma palavra do fluxo de entrada e carrega do cartão SD o arquivo com esse nome. Executado na forma:

FLOAD <name.F79>

FLUSH

Salva o conteúdo do buffer de bloco no disco de RAM se ele não está vazio. Em seguida, ele marca o buffer do bloco como vazio.

FORGET F79,186

Execute na forma:

FORGET <name>

Exclue do dicionário <nome> (que está no vocabulário **CURRENT**) e todas as palavras adicionadas ao dicionário após <nome>, independentemente do seu vocabulário. Falha em encontrar <nome> em **CURRENT** ou **FORTH** é uma condição de erro.

FORMAT --

Preenche todo o disco de RAM com espaços em branco.

FORTH -- F79,I,187

O nome do vocabulário principal. Sua execução torna **FORTH** o Vocabulário **CONTEXT**. Novas definições tornam-se parte do **FORTH** até que um vocabulário **CURRENT** diferente seja estabelecido.

Os vocabulários do usuário concluem pelo 'encadeamento' ao **FORTH**, então deve ser considerado que **FORTH** está 'contido' dentro de cada vocabulário do usuário.

FSAVE --

Lê uma palavra do fluxo de entrada e salva no cartão SD com esse nome uma cópia do vocabulário do usuário. Executado na forma:

FSAVE <name.F79>

GET n --

Lê uma palavra do fluxo de entrada e carrega o arquivo com esse nome do armazenamento em massa para o disco de RAM na tela n e quaisquer telas a seguir.

H. n --

Emite n como um inteiro hexadecimal com um espaço em branco à direita. A base atual não é alterada.

HERE -- addr F79,188

Retorna o endereço do próximo local de dicionário disponível.

HEX -- F79,R,162

Define a base de conversão de entrada-saída numérica para dezesseis.

HLD -- addr U

Uma variável de usuário que contém o endereço do último caractere de texto durante a conversão de saída numérica.

HOLD c -- F79,175

Insere c em uma string de saída numérica ilustrada. Só deve ser usado entre **<#** e **#>** .

I -- n F79,C,136

Copia o índice do loop para a pilha de dados. Só pode ser usado na forma:

DO ... **I** ... **LOOP** ou

DO ... **I** ... **+LOOP**

IF f -- F79,I,C,210

Usado em uma definição de dois pontos na forma:

flag **IF** ... **ELSE** ... **THEN** ou

flag **IF** ... **THEN**

Se o flag for verdadeiro, as palavras após **IF** são executadas e as palavras após **ELSE** são ignoradas. A parte **ELSE** é opcional.

Se o flag for falso, as palavras entre **IF** e **ELSE**, ou entre **IF** e **THEN** (quando **ELSE** não é usado), são ignorados. **IF**-**ELSE**-**THEN** podem ser aninhados.

IMMEDIATE F79,103

Marca a entrada de dicionário mais recente como uma palavra que será executada quando encontrada durante a compilação, em vez de ser compilada.

INDEX n1 n2 -- F79,R

Imprime a primeira linha de cada tela no intervalo {n1..n2}. Isso exibe a primeira linha de cada tela do texto fonte, que convencionalmente contém um título.

INKEY -- c

c=0 se nenhuma tecla for pressionada, senão c é o código ASCII da tecla que está sendo pressionada.

INTERPRET F79,R

Começa a interpretação no caractere indexado pelo conteúdo de **>IN** em relação ao número do bloco contido em **BLK**, continuando até que o fluxo de entrada se esgote. Se **BLK** contém zero, interpreta os caracteres da entrada do buffer do terminal.

INVERSE --

Alterna o bit 7 dos próximos caracteres a serem enviados ao terminal.
(Ver palavra CHR$128).

IS --

Usado para atribuir uma ação a uma palavra de vetor de execução. Ver **EXEC:**.

J -- n F79,C,225

Retorna o índice do próximo loop externo. Só pode ser usado dentro de um **DO**-**LOOP** aninhado na forma:

DO ... **DO** ... **J** ... **LOOP** ... **LOOP**

KEY -- c F79,100

Deixa o valor ASCII do próximo caractere disponível no dispositivo de entrada atual.

LATEST -- addr FIG

Deixa o endereço do name field da palavra mais recente no vocabulário **CURRENT**.

LEAVE F79,C,213

Força o encerramento de um **DO-LOOP** no próximo **LOOP** ou **+LOOP** forçando o limite do loop igual ao valor atual do índice. O próprio índice permanece inalterado e a execução prossegue normalmente até que a palavra de terminação do loop seja encontrada.

LIMIT -- addr

O endereço logo acima do buffer de bloco.

LIST n -- F79,109

Lista o conteúdo simbólico ASCII da tela n no atual dispositivo de saída, definindo **SCR** para conter n. n não tem sinal.

LITERAL n -- F79,I,215

Se estiver compilando, compila o valor n da pilha como um literal de 16 bits, que quando executado posteriormente, deixará n na pilha.

LO -- addr

Variável contendo o endereço de início do disco de RAM.

LOAD n -- F79,202

Começa a interpretação da tela n tornando-a a entrada de fluxo; preserva os localizadores da atual entrada de fluxo (de **>IN** e **BLK**). Se a interpretação não for encerrada explicitamente será encerrada quando o fluxo de entrada for exaurido. O controle então retornará ao fluxo de entrada contendo **LOAD**, determinado pelos localizadores de fluxo de entrada **>IN** e **BLK**. **>IN** e **BLK**.

LOOP F79,I,C,124

Incrementa o índice **DO-LOOP** em um, encerrando o loop se o novo índice for igual ou maior que o limite. O limite e o índice são números com sinal no intervalo {-32768 ..32767}.

M/MOD ud1 u2 -- u3 ud4 FIG "m-divide-mod"

Uma operação matemática de magnitude mista sem sinal que deixa um quociente duplo ud4 e resto u3, de um dividendo duplo ud1 e divisor único u2.

MAX n1 n2 -- n3 F79,218 "max"

Deixa o maior de dois números.

MEM --

Informa a quantidade de memória disponível entre o topo da pilha de dados e o ponteiro do dicionário.

MIN n1 n2 -- n3 F79,127 "min"

Deixa o menor de dois números.

MOD n1 n2 -- n3 F79,104

Divide n1 por n2, deixando o resto n3, com o mesmo sinal de n1.

MOVE addr1 addr2 n -- F79,113

Move a quantidade especificada n de células de memória de 16 bits começando em addr1 para a memória em addr2. O conteúdo de addr1 é movido primeiro. Se n for negativo ou zero, nada é movido.

NEGATE n -- -n F79,177

Deixa o complemento de dois de um número, ou seja, a diferença de zero menos n.

NOOP --

Nenhuma operação (não faz nada).

NOT f1 -- f2 F79,165

Reverta o valor booleano de flag1. Isso é idêntico a **0=** .

NUMBER addr -- n F79,R

Converte a contagem e a string de caracteres em addr em um Inteiro de 32 bits com sinal, usando a base atual. Se conversão numérica não é possível, existe uma condição de erro. A string pode conter um sinal negativo anterior.

OR n1 n2 -- n3 F79,223

Deixa o or inclusivo bit a bit de dois números.

OVER n1 n2 -- n1 n2 n1 F79,170

Deixa uma cópia do segundo número na pilha.

P! b addr --

Envia b para a porta de saída no endereço addr.

P@ addr -- b

Lê b da porta de entrada no endereço addr.

PAD -- addr F79,226

O endereço de uma área de rascunho usada para conter strings de caracteres para processamento intermediário. A capacidade mínima do PAD é 64 caracteres (addr a addr+63).

PAGE F79,R

Limpa a tela do terminal ou executa uma ação adequada para o dispositivo de saída atualmente ativo.

PAUSE --

Um vetor de execução que alterna para a próxima tarefa no modo multitarefa. No modo de tarefa única, execute o **NOOP**.

PICK n1 -- n2 F79,240

Retorna o conteúdo do n1-ésimo valor na pilha, sem contar n1 em si. Uma condição de erro resulta para n menor que um.

2 PICK é equivalente a **OVER**. {1..n}

PLOT n1 n2 n3 --

Plota um único ponto com coordenada x n1 e coordenada y n2 com modo n3 (0 = apagar, 1 = plotar, 2 = mover, 3 = inverter). As coordenadas vão de (0,0) a (63,47).

PUT n1 n2 --

Lê uma palavra do fluxo de entrada e grava as telas de n1 a n2 para a mídia de armazenamento em massa em um arquivo com aquele nome.

QUERY F79,235

Aceita a entrada de até 80 caracteres (ou até um 'retorno') do terminal do operador, no buffer de entrada do terminal. **WORD** pode ser usado para aceitar texto deste buffer como entrada stream, definindo **>IN** e **BLK** para zero.

QUIT F79,211

Limpa a pilha de retorno, definindo o modo de execução e retorne o controle para o terminal. Nenhuma mensagem é dada.

R> -- n F79,C,110 "r-from"

Transfira n da pilha de retorno para a pilha de dados.

R0 -- addr U "r=zero"

Uma variável de usuário contendo a localização inicial da pilha de retorno.

R@ -- n F79,C,228 "r-fetch"
Copia o número no topo da pilha de retorno para a pilha de dados.

RAND u --
Defina a semente de números aleatórios. Se u for zero, use o contador FRAMES.

REPEAT -- F79,I,C,120
Usado em uma definição de dois pontos na forma:
BEGIN ... **WHILE** ... **REPEAT**
Em tempo de execução, **REPEAT** retorna para logo após o **BEGIN** correspondente.

RND u1 -- u2
Gera um número pseudo-aleatório entre 0 e u1-1, usando como raiz o valor armazenado no endereço 16434. Veja **RAND** .

ROLL n -- F79,236
Extraia o enésimo valor da pilha para o topo da pilha, não contando n em si, movendo os valores restantes para a posição desocupada. Uma condição de erro resulta se n menor que 1. {1..n}
3 ROLL = **ROT**
1 ROLL = operação nula

ROT n1 n2 n3 -- n2 n3 n1 F79,212 "rote"
Rotaciona os três valores principais, trazendo o mais profundo para o topo.

RP! -- "r-p-zero"
Inicializa o ponteiro da pilha de retorno com o valor de **R0**.

RP@ -- addr "r-p-fetch"
Retorna o valor do ponteiro da pilha de retorno.

RUN --
Lê uma palavra do fluxo de entrada e carrega um arquivo com esse nome no disco de RAM a partir da tela 1. Em seguida, interpreta a primeira tela.

S->D n -- d FIG
Estende um número simples para formar um número duplo.

S0 -- addr F79,R "s-zero"
Retorna o endereço da base da pilha, quando vazia.

SAVE-BUFFERS F79,221

Copia o conteúdo do buffer de bloco para o disco de RAM se ele contiver uma tela válida.

SCR -- addr F79,U,217

Deixa o endereço de uma variável contendo o número da tela mais recentemente listada.

SIGN n -- F79,C,140

Insere o ASCII "-" (sinal de menos) na string da saída numérica ilustrada, se n for negativo.

SLOW

Define o modo SLOW.

SMUDGE -- FIG

Usado durante a definição de palavras para alternar o "smudge bit" do campo de nome da definição. Isso evita uma definição incompleta de ser encontrada durante pesquisas de dicionário, até que a compilação seja concluída sem erros.

SOUND n1 n2 --

Escreve b no registro n do chip de som (compatível com ZONX-81).

SP! --

Inicializa o ponteiro da pilha com o valor de **S0**.

SP@ -- addr F79,R,214 "s-p-fetch"

Retorna o endereço do topo da pilha, antes de **SP@** ser executado.

SPACE -- F79,232

Transmite um espaço em branco ASCII para o dispositivo de saída atual.

SPACES n -- F79,231

Transmite n espaços para o dispositivo de saída atual. Não faça nada se n igual a zero ou menos.

STATE -- addr F79,U,164

Deixa o endereço da variável que contém o estado da compilação. Um conteúdo diferente de zero indica que a compilação está ocorrendo, mas o valor em si pode depender da instalação.

SWAP n1 n2 -- n2 n1 F79,230

Troque os dois primeiros valores da pilha.

THEN F79,I,C,161

Usado em uma definição de dois pontos na forma:

IF ... **ELSE** ... **THEN** ou

IF ... **THEN**

THEN é o ponto onde a execução continua após **ELSE** ou **IF** (quando nenhum **ELSE** está presente).

TIB -- addr

O endereço do buffer de entrada de texto. Este buffer é usado para reter caracteres quando o fluxo de entrada está vindo do dispositivo de entrada atual. Neste sistema, a capacidade do **TIB** é de 128 caracteres.

TO n --

Armazene n no campo de parâmetro de <nome>. Executado na forma:

n **TO** <name>

TOGGLE addr b -- FIG

Complementa o conteúdo de addr pelo padrão de bits b.

TYPE addr n -- F79,222

Transmite n caracteres começando no endereço addr, para o atual dispositivo de saída. Nenhuma ação ocorre para n menor ou igual a zero.

U* un1 un2 -- ud3 F79,242 "u-times"

Realiza a multiplicação sem sinal de un1 por un2, deixando o produto de número duplo ud3. Todos os valores não têm sinal.

U. un -- F79,106 "u-dot"

Exibe un convertido de acordo com BASE como um número sem sinal, em um formato de campo livre, com um espaço em branco à direita.

U/MOD ud1 un2 -- un3 un4 F79,243 "u-divide-mod"

Realiza a divisão sem sinal do número duplo ud1 por un2, deixando o resto un3 e o quociente un4. Todos os valores são sem sinal.

U< un1 un2 -- f F79,150 "u-less-than"

Deixa o flag que representa a comparação de magnitude de un1 < un2 onde un1 e un2 são tratados como inteiros sem sinal de 16 bits.

UNDER x1 x2 -- x2

Remove o segundo valor da pilha.

UNTIL f -- F79,I,C,237

Dentro de uma definição de dois pontos, marca o final de um loop **BEGIN-UNTIL**, que terminará com base no flag. Se o flag for verdadeiro, o loop está terminado. Se o flag for falso, a execução retorna à primeira palavra após **BEGIN**. As estruturas **BEGIN-UNTIL** podem ser aninhadas.

UPDATE F79,229

Marca o bloco referenciado mais recentemente como modificado. O bloco será posteriormente transferido automaticamente para o armazenamento de massa caso seu buffer de memória seja necessário para o armazenamento de um bloco diferente, ou na execução de **SAVE-BUFFERS**. Como este sistema tem um disco de RAM rápido, os buffers sempre serão escritos de volta.

VALUE F79,227

Uma palavra definidora executada na forma:

VALUE <name>

para criar uma entrada de dicionário para <nome> e alocar dois bytes para armazenamento no campo de parâmetro, inicializando-o com zero. Quando <name> for executado posteriormente, o valor armazenado em seu campo de parâmetro será deixado na pilha. Este valor também pode ser alterado por meio das palavras **TO** e **TO+**.

VARIABLE F79,227

Uma palavra definidora executada na forma:

VARIABLE <name>

para criar uma entrada de dicionário para <nome> e atribuir dois bytes para armazenamento no campo de parâmetro, inicializando-o com zero. Quando <nome> é executado posteriormente, ele colocará o endereço de armazenamento na pilha.

VLIST F79,R

Lista os nomes das palavras do vocabulário **CONTEXT** começando com a definição mais recente.

VOC-LINK -- addr U

Uma variável de usuário contendo o endereço de um campo na definição do vocabulário criado mais recentemente. Todos os nomes de vocabulário são ligados por esse campo para permitir o controle de **FORGET** através de vários vocabulários.

VOCABULARY F79,208

Uma palavra definidora executada na forma:

VOCABULARY <name>

para criar (no vocabulário **CURRENT**) uma entrada de dicionário para <nome>, que especifica uma nova lista ordenada de definições de palavras. A execução subsequente de <nome> o tornará o vocabulário **CONTEXT**. Quando <nome> se torna o vocabulário **CURRENT** (ver **DEFINITIONS**), as novas definições serão criadas nessa lista.

Em vez de qualquer outra especificação, novos vocabulários 'encadeiam-se' para **FORTH**. Ou seja, quando uma pesquisa de dicionário através de um vocabulário se esgota, **FORTH** será pesquisado.

WAIT u --

Aguarda até que u interrupções de temporização tenham ocorrido. Enquanto espera, execute a palavra **PAUSE**.

WARM

Inicializção quente do Forth. Ele executa as seguintes ações:

- Redefine **CONTEXT** e **CURRENT** para o vocabulário **FORTH**.
- Reinicializa algumas variáveis do usuário.
- Executa o loop de comando via **QUIT**.

WHILE f -- F79,I,C,149

Usado na forma:

BEGIN ... flag **WHILE** ... **REPEAT**

Seleciona a execução condicional com base no flag. Para uma flag verdadeira, continua a execução até **REPEAT**, que então retorna para logo depois de **BEGIN**. Para uma flag falsa, pula a execução para depois do **REPEAT**, saindo da estrutura.

WORD c -- addr F79,181

Recebe caracteres do fluxo de entrada até que o caractere delimitador diferente de zero seja encontrado ou o fluxo de entrada seja exaurido, ignorando delimitadores iniciais. Os caracteres são armazenados como uma string empacotada com a contagem de caracteres na sua primeira posição. O delimitador real encontrado (char ou nulo) é armazenado no final do texto, mas não incluído na contagem. Se o fluxo de entrada foi esgotado assim que **WORD** foi chamado, então resulta um comprimento zero. O endereço do início desta string empacotada é deixado na pilha.

XOR n1 n2 -- n3 F79,174 "x-or"

Deixa o OR exclusivo bit a bit de dois números.

[I,125 "left-bracket"

Encerra o modo de compilação. O texto do fluxo de entrada é subseqüentemente executado. Ver **]**.

[COMPILE] F79,I,C,179 "bracket-compile"

Usado em uma definição de dois pontos na forma:

[COMPILE] <name>

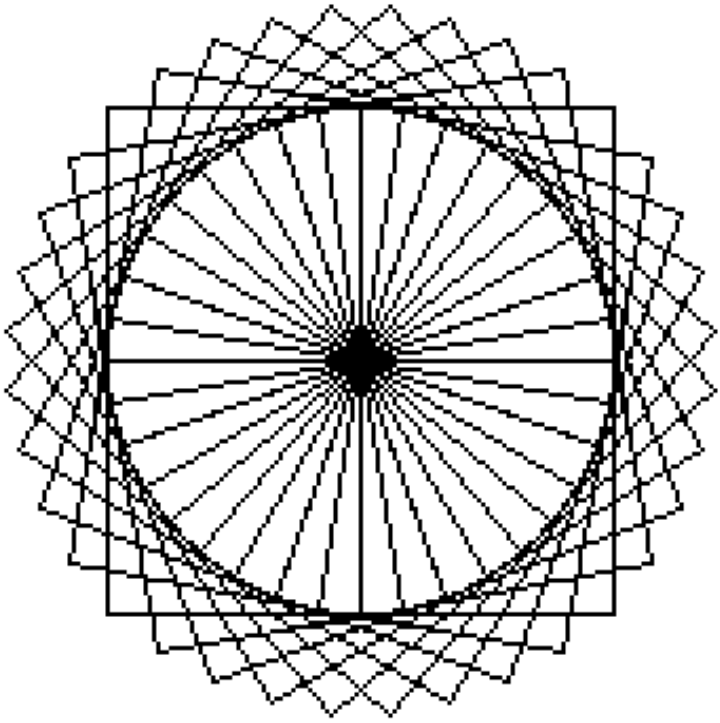
Força a compilação da palavra seguinte. Isso permite a compilação de uma palavra **IMMEDIATE** quando, de outra forma, seria executada.

] F79,126 "right-bracket"

Define o modo de compilação. O texto do fluxo de entrada é subseqüentemente compilado. Ver **[**.

**** "backslash"

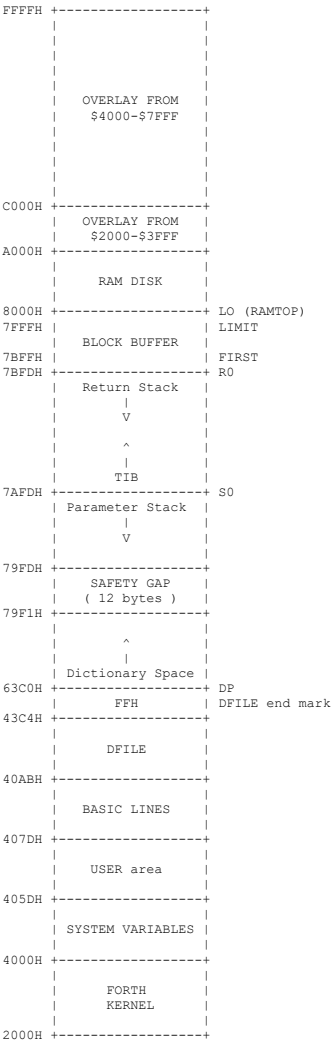
Comentário até o final da linha. Só deve ser usado em uma tela.



Apêndice B

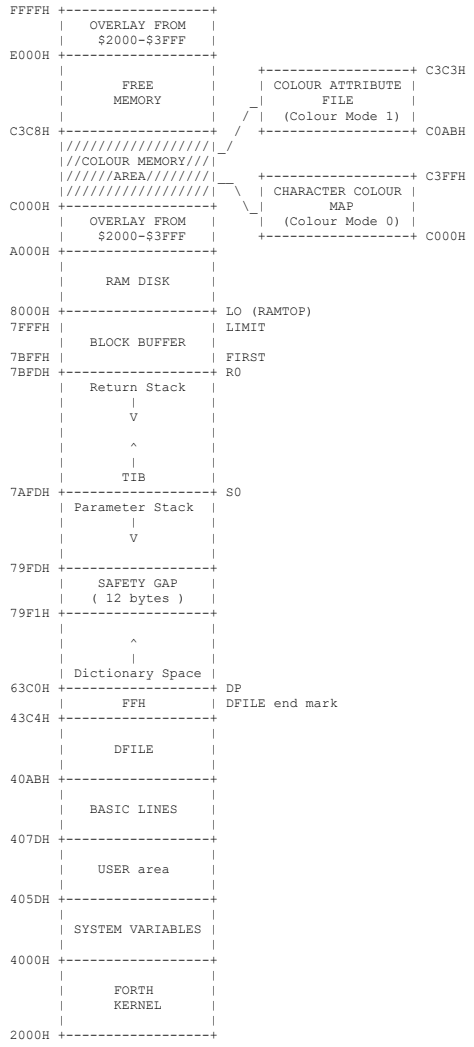
APPENDIX B - OS DIAGRAMAS DE MEMÓRIA

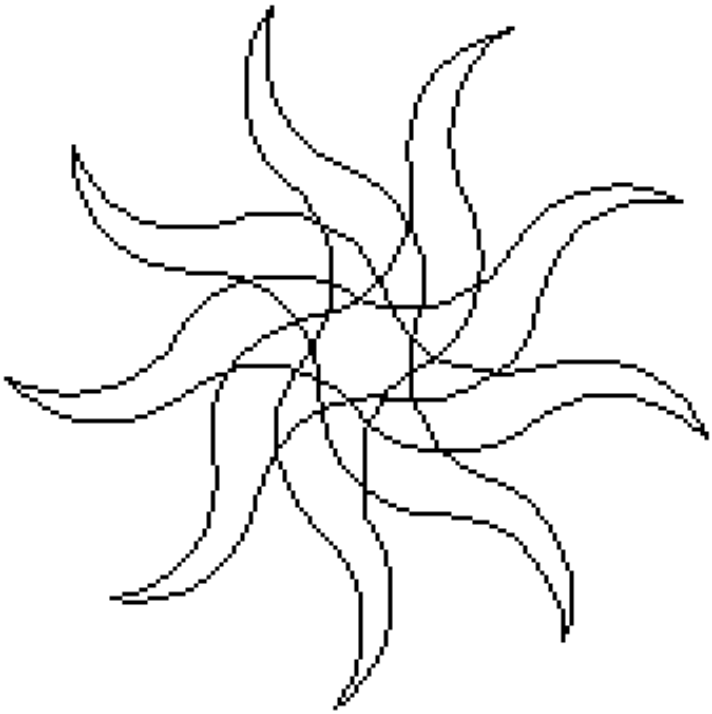
ZXPand



ZXPand + Chroma 81

(Switch 3 + Switch 6 ON)

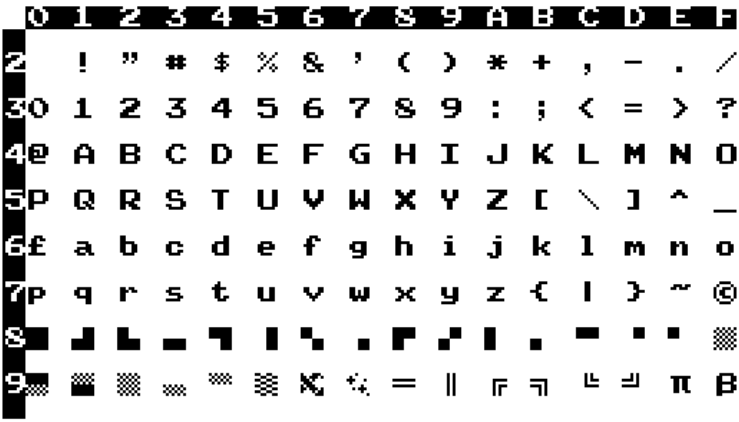




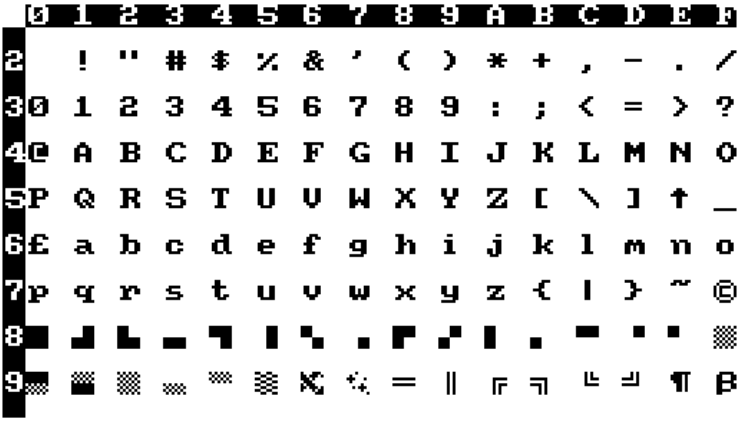
Apêndice C

APPENDIX C – OS CONJUNTOS DE CARACTERES

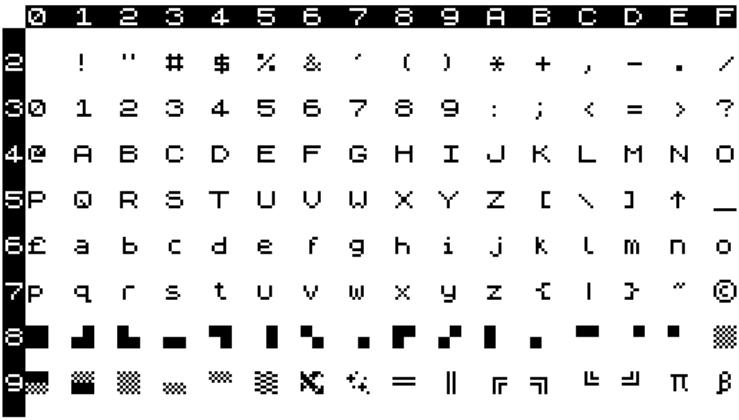
ARCADE.FN



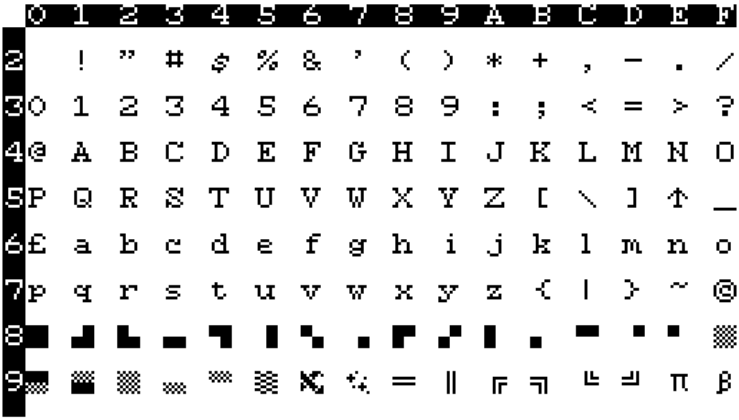
CPC.FN



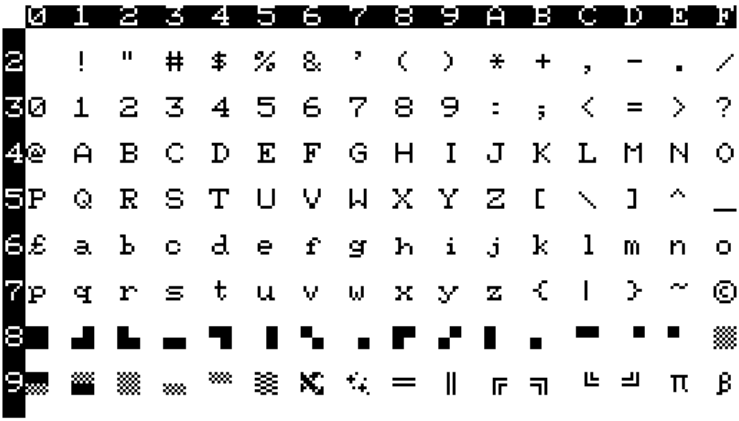
SINCLAIR.FN



SINSERF.FN



TF79.FN



TF79B.FN