SUPER BASIC TK

MALDONADO
& GROSSI

para SINCLAIR ZX-81, CP-200,
TK 82/83/85, Ringo,
TS 1000

A ELABORAR PROGRAMAS EM
RODAM COM A RAPIDEZ
AGEM DE MÁQUINA

SUPER BASIC TK

FERNANDO GROSSI

MILTON MALDONADO JR

SUPER BASIC TK

Editora Aleph

EDITORA MODERNA

Coordenação Editorial:
Pierluigi Piazzi
Avaliação, Editoração e Revisão técnica:
Roberto Bertini Renzetti
Arte:
Ana Lúcia Antico
Arte da Capa:
Pierluigi Piazzi e Roberto Bertini Renzetti
Produção:
Rosana de Angelo

Distribuição exclusiva em livrarias

Editora Aleph

ALEPH PUBLICAÇÕES
ASSESSORIA PEDAGÓGICA LTDA.
Av. Brig. Faria Lima, 1451 cj. 31
01451 — São Paulo - SP
Tel.: (011) 813-4555

EDITORA MODERNA
Rua Afonso Brás, 431
04511 - São Paulo - SP
Tel.: (011) 531-5099

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

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

Maldonado Júnior, Milton, 1967-
M211s Super BASIC TK : para TK-85, Sinclair, Ringo R-470, CP-200 / Milton Maldonado Jr. e Fernando Grossi. — São Paulo : Aleph : Ed. Moderna, 1985.

Bibliografia.

1. BASIC (Linguagem de programação para computadores) 2. Compilação (Computadores eletrônicos) 3. Microcomputadores — Programação 4. TK (Computador) — Programação I. Grossi, Fernando, 1967- II. Título.

17. CDD-651.8
18. -001.642
19. -001.6424
18. -001.6425

85-0776

Índice para catálogo sistemático:

1. BASIC : Linguagem de programação : Computadores : Processamento de dados 651.8 (17.) 001.6424 (18.)
2. Compiladores : Computadores : Processamento de dados 651.8 (17.) 001.6425 (18.)
3. Microcomputadores : Programação : Processamento de dados 651.8 (17.) 001.642 (18.)
4. Programação : Microcomputadores : Processamento de dados 651.8 (17.) 001.642 (18.)
5. TK : Computadores : Programação : Processamento de dados 651.8 (17.) 001.642 (18.)

SUMÁRIO

AGRADECIMENTOS

Ao Perluigi pela chance dada, aos meus pais pelo apoio, à minha tia Ercília pela ajuda, e ao Gil, o polivalente primo programador, radioamador, fotógrafo e campista, pelo meu início nesse mundo maravilhoso da computação.

FERNANDO

Ao Pierluigi, por acreditar em nosso potencial e tornar em realidade nossas aspirações, à minha irmã Viviane, pelo apoio dado e pela valiosa colaboração para o desenvolvimento desta obra.

MILTON

NOTA DO EDITOR

Um dos erros mais trágicos, cometidos pelos 20 anos de ditadura militar pela qual o Brasil passou, foi paradoxalmente cometido em boa fé. Muitos "revolucionários" bem intencionados tentaram fazer do Brasil um grande país: investiu-se muito em grandes projetos e em obras faraônicas (que além de tudo têm a grande vantagem de pagar boas comissões), mas nada se fez de significativo para criar uma grande nação.

Uma nação é feita de gente, não de pontes ou inúteis centrais nucleares. Investir em gente significa investir em educação e, principalmente, investir em inteligência!

Infelizmente nosso sistema educacional não só não se preocupa em desenvolver inteligência, como tenta quase que intencionalmente castrar intelectualmente aquela porcentagem biologicamente pré-determinada de excepcionais positivos. E nas raras oportunidades em que a criança ou o adolescente super-dotado são reconhecidos como tais, são utilizados como argumentos de marketing de colégios inescrupulosos, sendo expostos à admiração pública como fenômenos de parque de diversões.

A explosão da informática, e principalmente, a disseminação de microcomputadores pessoais, colocaram na mão destes jovens um instrumento que evidencia de maneira dramática esta inteligência. Assistimos a uma proliferação de "geninhos" que deixam os adultos abismados com suas

habilidades em programar, alterar e dominar um microcomputador.

O Milton e o Fernando são dois típicos jovens entusiastas, que de maneira totalmente auto-didática, aprenderam a fazer verdadeiros milagres com seus micros. Sofisticação tecnológica? Claro que não! Trata-se simplesmente de entusiasmo e inteligência.

Eles fazem com um micro coisas que muito pomposo executivo de CPD nem sequer sonharia em fazer, utilizando uma fração insignificante das verbas que ele usaria.

Agora vem uma pergunta: antes do advento desses micros, onde estavam estes "geninhos"? Será que eles são fruto da interação do homem com a máquina? Será que o uso do computador torna o jovem mais inteligente?

Vinte e cinco anos de magistério permitem-me responder com segurança e com conhecimento de causa a todas estas perguntas: os "geninhos" sempre existiram. Há um quarto de século encontro-os, ano após ano, massacrados e neurotizados por um sistema educacional projetado por medíocres e para medíocres! O micro não torna ninguém mais ou menos inteligente: ele apenas permite evidenciar a inteligência latente, por representar um campo no qual os imbecis do sistema ainda não conseguiram pôr suas patas, cerceando criatividade e nivelando por baixo.

Não tivesse outro mérito, o microcomputador permitiu, pelo menos, evidenciar o absurdo desperdício de capital humano que um país tão pobre e endividado como o Brasil não podia ter permitido que acontecesse.

O Fernando e o Milton foram meus alunos quando os preparei para o vestibular. Com orgulho vi-os ingressar na Universidade, e com orgulho publico este livro, no qual eles transmitem para outros jovens não só todo o conhecimento adquirido, mas principalmente, o entusiasmo e o prazer de pensar.

Este orgulho, porém, vem acompanhado de uma ponta de preocupação: ao cursar universidades brasileiras estarão se expondo a um sistema estruturado em moldes copiados (e mal copiados!) de modelos estrangeiros, afastados de nossas necessidades e nossas realidades. Espero, de todo coração, que não se deixem envolver pela armadilha e conservem toda sua criatividade, toda sua inteligência e todo seu entusiasmo.

É disso que o Brasil precisa, agora mais do que nunca!

Pierluigi Piazzi

INTRODUÇÃO

Quem não tem um computador, do pequeno ZX81 ao grande Sinclair QL, que não disse alguma vez: "Farei um jogo melhor que um reles cartucho"?

Porém, quantas vezes você já tentou programar em BASIC e sentiu que não ia dar, por problemas de velocidade, falta de memória e outros incômodos que o levam a desistir do projeto?

Aí você pensa em Assembly, sai correndo atrás de literatura especializada, livros de consulta e, depois de rodar a Santa Ifigênia de ponta a ponta, descobre alguns livros nacionais feitos de papel e alguns outros importados feitos de ouro. E, entre uma e outra folheada, você vira para o vendedor e diz:

— Não era bem isso que eu queria.

E vemos um usuário de Sinclair voltar para o seu ninho desolado e continuar rodando programas comprados ou "chupados" de amigos.

De repente, no silêncio do seu pequeno mundo, aparece uma fada madrinha e pergunta:

— Você sabe o que é um compilador BASIC?

Voltando à realidade, você pergunta ao mundo o que é um compilador, ao que ele responde:

— É a sua salvação.

Um compilador BASIC é, essencialmente, um programa-ferramenta que transforma BASIC em Assembly.

O que nós tentaremos passar a você neste livro são dois anos de dedicação integral a um programa, tentando explorar dele todas as suas capacidades, otimizando os ganhos e minimizando as desvantagens.

Não espere deste livro um curso de linguagem BASIC completo, pois esperamos que alguém que deseja fazer um jogo já domine o vocabulário BASIC e sua sintaxe. O que será explanado neste livro é o que chamamos dialeto BASIC.

— Dialeto BASIC?!?!?!...

Duvido que um gaúcho converse com um cearense e entenda 50% do que ele falou. Em uma única linguagem, existem várias maneiras de expressar uma mesma idéia.

Lendo as listagens, você poderá encontrar uma solução técnica que nunca se aplicaria a um programa BASIC normal.

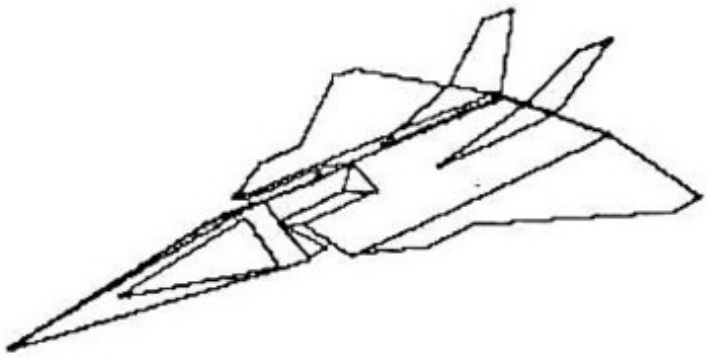
Quando você usaria um loop FOR-NEXT de Ø a 1ØØØØ para criar uma pausa de alguns segundos no modo SLOW? Em Roma, como os romanos. Já que tudo o que você digitar será transformado em Assembly, por que não seguir um BASIC voltado para esta linguagem?

Para quem estiver mesmo com vontade de dominar a técnica do compilador, um conselho: digite todos os programas deste livro, pois atrás dos programas mais simples, existem grandes idéias.

Não se acanhe em dizer: "Não entendi". A melhor maneira de se aprender é errando. E não se esqueça nunca: duas cabeças pensam melhor que uma, portanto, troque idéias com os amigos sempre que puder. Na medida do possível, tente aperfeiçoar todos os jogos compilados. Lembre-se: o limite da perfeição é o infinito, quando a paciência também o é.

Agora pode começar: prepare seus dedos, almofada na cadeira e mãos à obra...

Os Autores



CAPÍTULO 1

O COMPILADOR

Afinal, o que é um compilador? Trata-se de um programa tradutor que converte uma linguagem de alto nível em uma linguagem de baixo nível, voltada para a máquina.

Muitas linguagens de computador só funcionam da maneira compilada, como por exemplo, Assembly , Cobol e Fortran. Outras admitem o modo interpretativo e o compilado, como o Basic e o Forth.

Normalmente, um compilador profissional é bastante complexo e extenso, podendo ocupar 50 Kbytes de memória ou mais. Além disto, ele precisa do auxílio de um outro programa chamado "Link-editor", que se encarrega de distribuir o código-objeto ao longo da memória de acordo com suas divisões: área de programa, variáveis, "stacks", etc.

Calma, não se apavore, pois o nosso compilador não é assim. Ele ocupa somente 2 Kbytes e não requer programas auxiliares. Entretanto, processa as funções mais importantes do BASIC com eficiência, e nos capítulos posteriores nós o ensinaremos a otimizar seu funcionamento.

Sua localização dá-se na primeira linha do programa e o código objeto situa-se na segunda linha. O programa a ser compilado inicia-se na linha 3 e seu término é marcado pelo primeiro STOP da área de programa (após esta linha, nada mais é compilado).

Seu sistema aritmético é restrito aos números inteiros situados entre −32768 e 32767, não apresentando as funções matemáticas que usam ponto flutuante, como seno, cosseno, tangente, logaritmo e suas inversas, PI, tampouco álgebra booleana e tratamento de strings.

Estas limitações deram margem a críticas por parte de publicações especializadas, que afirmavam ser mais sensato juntar rotinas em Assembly ao Basic interpretativo. Deste modo, entretanto, a grande velocidade da linguagem de máquina é desperdiçada pela lentidão do BASIC normal. Com o compilador, você usufrui da velocidade do Assembly sem ter que conhecê-lo a fundo. E mesmo quando se utiliza rotinas em linguagem de máquina com o compilador, estas são bastante simples e servem principalmente para economizar memória.

Portanto, após esta pequena explicação, convidamos todos os céticos a lerem este livro para que conheçam as reais capacidades deste compilador.

Podemos afirmar com relativa convicção que o compilador adotado por nós é o único que compila o Basic do ZX81, o que evidencia o descaso de quase toda a Terra com a linguagem residente do micro.

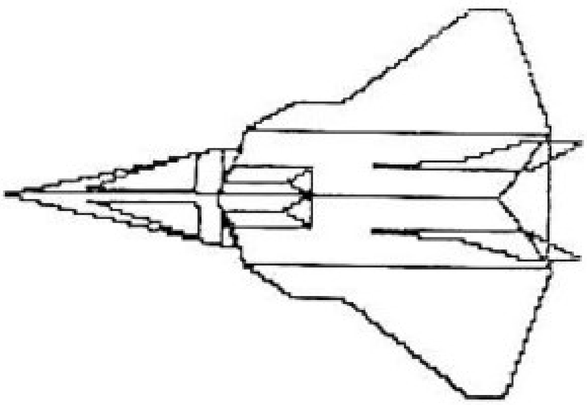
Aqui no Brasil, ele é comercializado por várias empresas de software sob os mais diversos nomes, mas a versão original foi desenvolvida pela:

INTERCOMPUTER INC.
P.O. BOX 90 PRUDENTIAL CENTER
BOSTON, MASSACHUSETTS 02199
USA
TEL (617) 437 1190
TELEX 951 140 COFAR

Estas informações, entretanto, são discutíveis, pois o manual do usuário fornecido pela empresa está incompleto, não mencionando a existência do comando INPUT no vocabulário das palavras compiláveis, tampouco o uso da tecla "D" durante o processo de compilação.

Convém explicar que a versão do compilador apresentada neste livro não é mais a original. Ele passou por modificações que melhoraram seu desempenho permitindo compilações mais rápidas, otimizando o CLS (muito lento) e cancelando instruções deficientes ou inúteis (INPUT, DIM, RAND, FAST e SLOW).

Para ter seu próprio compilador gravado em fita, siga as instruções do apêndice 1 (pág. 103).



CAPÍTULO 2

SAIBA COMO E USE BEM

Bem, chegou a hora de você aprender a operar o compilador. Mesmo que você já o conheça, é aconselhável acompanhar este capítulo, especialmente se as modificações do capítulo anterior foram realizadas.

Em primeiro lugar, certifique-se de que o compilador e o monitor do Apêndice I são os únicos dados que estão na memória (a presença do monitor não é obrigatória). A seguir, digite a listagem exatamente como ela está impressa. Não se assuste com a "forma estranha" de alguma linhas, você as entenderá mais adiante.

```
 10 REM ▒▒▒TESTE-COMP▒▒▒
 20 FOR M=0 TO 63
 30 FOR N=0 TO 43
 40 IF M<=63 THEN PLOT M,N
 50 NEXT N
 60 NEXT M
 70 FOR M=1 TO 22
 80 SCROLL
 90 PRINT "▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒";
100 NEXT M
110 GOSUB 520
120 FOR L=0 TO 21
130 PRINT AT L,L;CHR$ (L+28)
```

```
 140 NEXT L
 150 LET A=PEEK 16396+256*PEEK 1
6397
 160 FOR B=A TO (A+792)
 170 IF PEEK B<>118 THEN POKE B
,PEEK B-128)
 180 NEXT B
 190 PAUSE 120
 200 GOSUB 520
 210 FOR L=-10 TO 10
 220 SCROLL
 230 PRINT L;AT 21,5;ABS L;AT 21
,10;(L+L);AT 21,15;(L*2);AT 21,2
0;(L-L);AT 21,25;(L/2)
 240 NEXT L
 250 PAUSE 120
 260 CLS
 270 FOR L=1 TO 1000
 280 IF L<500 THEN PLOT (RND/512
),(RND/745)
 290 IF L>500 THEN UNPLOT (RND/5
12),(RND/745)
 300 NEXT L
 310 PAUSE 240
 320 LET L=USR 2602
 330 FOR L=1 TO 22
 340 LET Z(L)=RND
 350 NEXT L
 360 FOR L=1 TO 22
 370 SCROLL
 380 PRINT "Z(";L;")=";Z(L)
 390 NEXT L
 400 PAUSE 120
 410 CLS
 420 PRINT AT 10,5;"APERTE NEWLI
NE"
 430 IF CODE INKEY$<>118 THEN GO
TO 430
 440 GOSUB 520
 450 PRINT AT 0,0;"FIM DAS OPERA
COES"
 460 LET X=0
 470 FOR L=16514 TO 18814
 480 LET X=X+PEEK L
 490 NEXT L
 500 PRINT "SOMA SINTATICA=";X
 510 GOTO 590
 520 LET A=PEEK 16396+256*PEEK 1
6397
 530 LET B=A
 540 IF PEEK B<>118 THEN POKE B,
0
```

```
550 LET B=B+1
560 IF B<(A+792) THEN GOTO 540
570 PRINT AT 0,0;
580 RETURN
590 STOP
600 LET L=USR 18823
```

Antes de prosseguir, faça uma gravação do programa. Agora você vai testar se o compilador está OK através do TESTE-COMP, um programa que aciona todas as funções a fim de testá-las.

A compilação:

Sempre que for compilar um programa, verifique se não existe nenhuma linha 2 na memória. A inobservância desta regrinha pode causar o "over-flow" da memória e até provocar um "crash" do sistema. A seguir, passe o micro para o modo SLOW e digite:

```
RAND USR 17389
```

Deverá aparecer na tela da TV uma parte do Basic com um sinal de interrogação reverso no canto inferior esquerdo. Não se preocupe com o estranho espacejamento que aparece entre algumas palavras da listagem. Para avançar, aperte a tecla "C". A tela apagará e aparecerá a continuação da listagem com a "?" reversa. Repita o procedimento até que a listagem tenha percorrido a tela por três vezes, quando então surgirá o código 00.

Outros modos de compilar:

Você não é obrigado a pressionar somente a tecla "C". Existem outras teclas de controle. Se você apertar NEW LINE, a listagem avançará "subindo" a tela.

No caso de precisar de uma cópia da tela na impressora, pressione a tecla "Z" e o micro realizará um comando COPY sem interromper a compilação.

Para "xeretar" o trabalho do compilador, pressione a tecla "D". Nesta modalidade, o compilador mostra os bytes que estão sendo criados por ele à razão de 2 por segundo. É uma tremenda perda de tempo, pois ninguém possui paciência bastante para compilar um programa inteirinho nesta velo-

cidade. O número que acompanha cada byte pode ter dois significados:

a) Na primeira "passada" da listagem, indica a posição do byte em relação ao início da área compilada
b) Na segunda e terceira "passadas", indica o endereço do byte.

Interrupções do processamento:

Três podem ser as causas de paradas antecipadas do processo de compilação: o pressionamento da tecla BREAK, erros do BASIC ou "overflow" da memória, sendo mais comuns as duas primeiras.

A tecla BREAK pode ser pressionada a qualquer instante, e neste caso a compilação será abortada. Neste momento, verifique se a linha 2 já foi gerada, e se foi, apague-a.

O "overflow" da memória é muito perigoso, pois é pouco provável que o programa sobreviva após um deles. Pode ter duas causas: programa longo demais ou acúmulo de linhas 2 devido a várias compilações

Os erros do BASIC são muito comuns e ocorrem com grande freqüência com usuários não habituados ao compilador.

Erros do BASIC:

Durante o processo de compilação, o sistema pode, eventualmente, detectar algum erro no BASIC. Se isto acontecer, deverá surgir um "S" inverso no final da linha errada, bem como a mensagem "S"/Ø. Corrija esta linha e compile novamente (neste caso não é necessário apagar a linha 2).

Causas de erros do Basic:

As causas de erros podem ser:

- Palavras estranhas ao compilador (DIM, SLOW, FAST, SIN, COS, TAN, PI, etc.).
- Uso de ponto decimal ou notação científica (são formas inválidas).
- Omissão de parênteses em expressões matemáticas fora do comando LET.
- Ausência do STOP no fim do programa.
- Linhas com estrutura danificada (por um POKE acidental ou por erro de leitura na fita).

No próximo capítulo, as palavras compiláveis serão mostradas com detalhes.

Execução do programa:

Antes de executar o programa, verifique se a linha 2 está presente, pois é ela quem contém o programa compilado. Se ela não existir, compile o programa. **Não** execute um programa após uma compilação abortada, isto é muito perigoso.

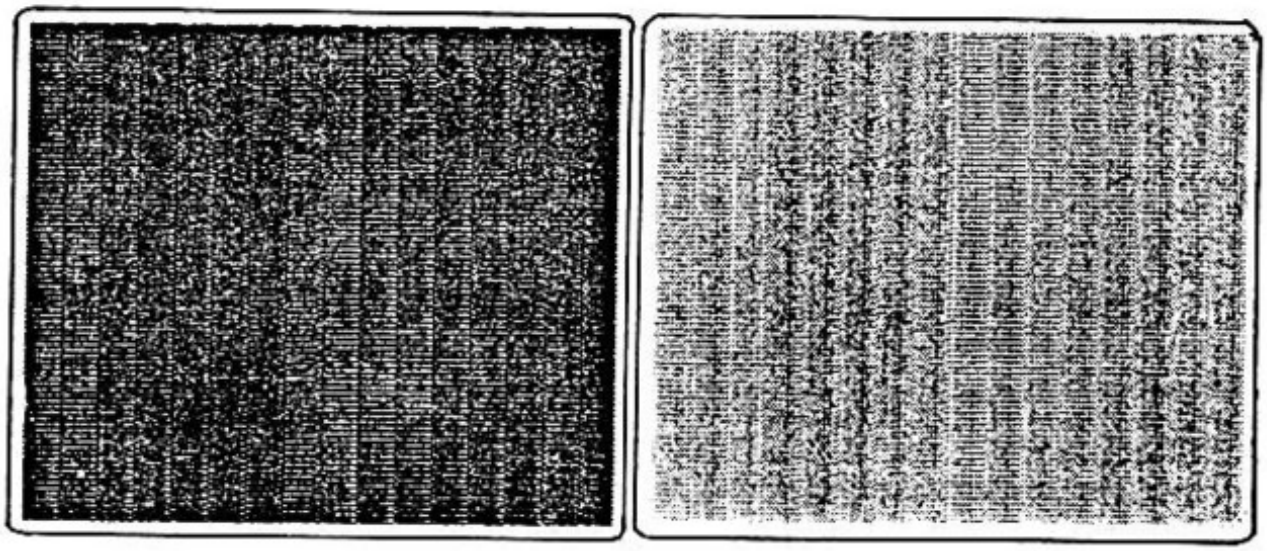
Para chamar o programa compilado, digite:

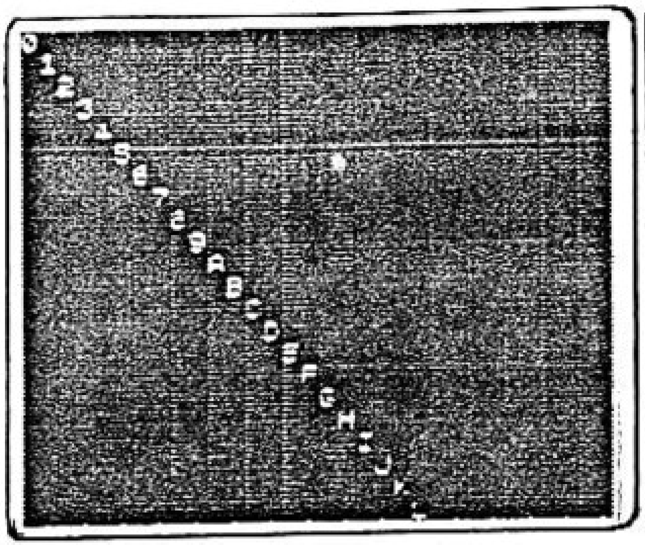
```
LET L=USR 18823
```

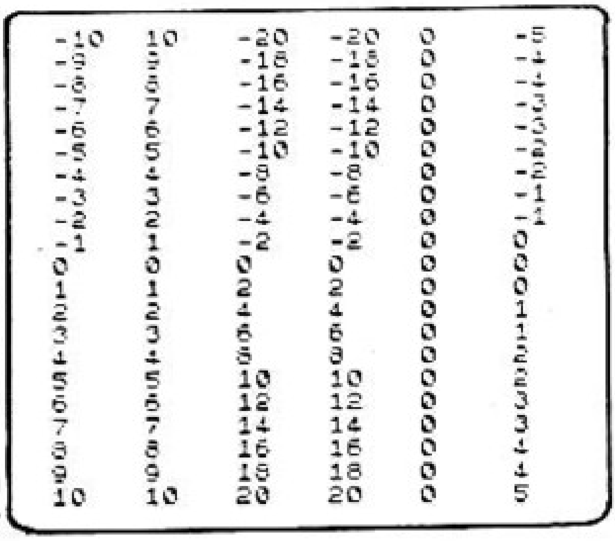
Não use o tradicional RAND USR, porque parece que o PLOT e o UNPLOT do compilador são alérgicos a este modo de chamada, e pode até acontecer do seu programa não achar o rumo de casa na hora de retornar ao BASIC.

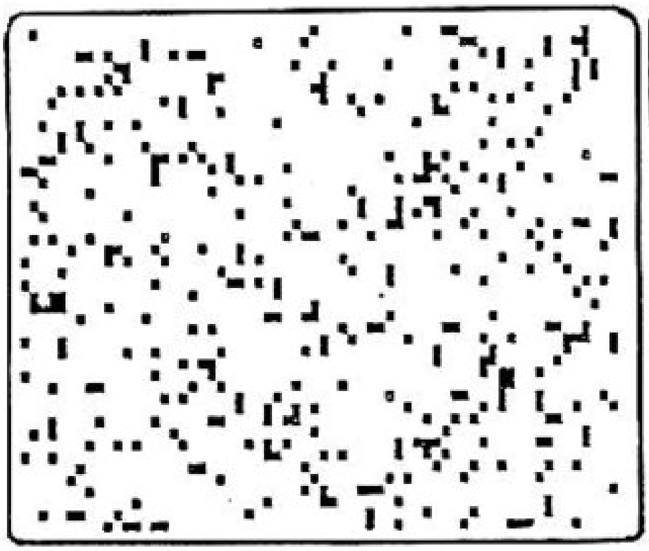
O teste:

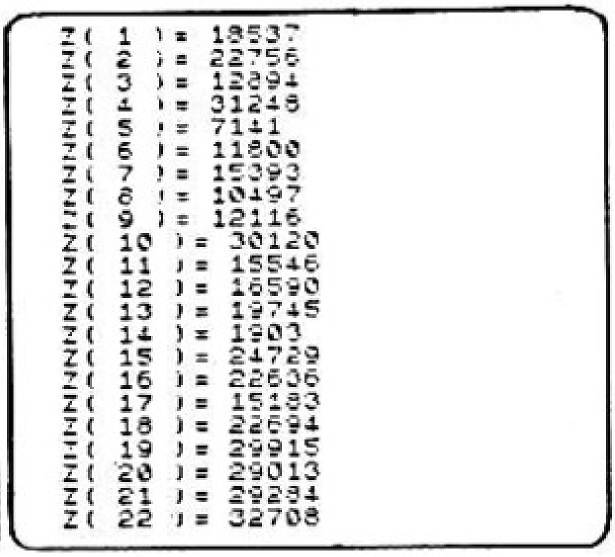
Se tudo foi digitado corretamente, você verá a tela sendo preenchida por uma seqüência de PLOTs muito rápida, e logo depois por "quadradinhos cinzas" (CHR$ 8). Siga as telas listadas a seguir, confira-as e vá pressionando NEW LINE. Se acontecer alguma divergência, ou se ainda o micro "sair do ar", confira primeiro a listagem, e depois, os bytes do compilador.





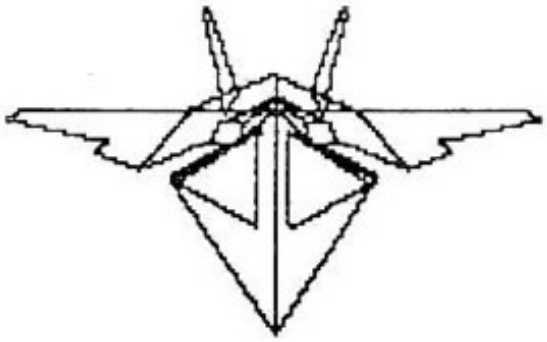

-10 10 -20 -20 0 -5
-9 9 -18 -18 0 -4
-8 8 -16 -16 0 -4
-7 7 -14 -14 0 -3
-6 6 -12 -12 0 -3
-5 5 -10 -10 0 -2
-4 4 -8 -8 0 -2
-3 3 -6 -6 0 -1
-2 2 -4 -4 0 -1
-1 1 -2 -2 0 0
0 0 0 0 0 0
1 1 2 2 0 0
2 2 4 4 0 1
3 3 6 6 0 1
4 4 8 8 0 2
5 5 10 10 0 2
6 6 12 12 0 3
7 7 14 14 0 3
8 8 16 16 0 4
9 9 18 18 0 4
10 10 20 20 0 5




Z(1)= 18537
Z(2)= 22756
Z(3)= 12894
Z(4)= 31248
Z(5)= 7141
Z(6)= 11800
Z(7)= 15393
Z(8)= 10497
Z(9)= 12116
Z(10)= 30120
Z(11)= 15546
Z(12)= 16590
Z(13)= 19745
Z(14)= 1903
Z(15)= 24729
Z(16)= 22636
Z(17)= 15183
Z(18)= 22694
Z(19)= 29915
Z(20)= 29013
Z(21)= 29284
Z(22)= 32708


APERTE NEWLINE


FIM DAS OPERACOES
SOMA SINTATICA= 4410



CAPÍTULO 3

AS FUNÇÕES DO COMPILADOR

Após observar a velocidade de operação do compilador, você deve ter-se perguntado como isso foi conseguido. É uma boa pergunta, bem a propósito, pois chegou a hora de certas explicações.

A alta velocidade do compilador foi conseguida principalmente pelo abandono da técnica decimal, isto é, o compilador somente opera com números inteiros entre -32768 e 32767. Isto pode parecer uma grande limitação, porém para a maioria dos jogos normais é ideal. A técnica de simular casas decimais será mostrada num capítulo em especial, juntamente com as funções matemáticas. Outra parte complexa do BASIC Sinclair que foi abandonada é o tratamento de strings, porém sua simulação é simples, através das funções PEEK e POKE.

Realmente, a maior arma deste compilador são as funções PEEK e POKE, que mostram todo o seu poder quando usadas para controle de imagens na tela, (se você não domina bem esta aplicação, dê uma olhada no apêndice 2) e também simulando strings.

A seguir mostraremos todas as palavras do BASIC MCoder, com uma explicação de sua nova sintaxe.

STATEMENTS – MCoder:

RND – como estamos trabalhando agora sem casas decimais, a função RND fornece um número entre Ø e 32767, variando de 1 em 1.

INKEY$ – como não trabalhamos mais com strings, é necessário que peçamos como resposta o código da tecla, portanto, a função CODE, deve preceder a palavra INKEY$, (esta condição é necessária e suficiente) e ser igualada ou comparada a um valor numérico ou variável.

AT – posiciona o PRINT na posição desejada (linha de Ø a 21 e coluna de Ø a 31). Note que o compilador não permite a impressão nas 2 linhas inferiores. Qualquer tentativa força o sistema a dar um SCROLL automático para cima.

INT – palavra inútil no compilador, pois ele fornece todos os resultados em números inteiros.

ABS – reduz um número ao seu módulo, isto é, transforma o número em positivo.

PEEK – uma das poderosas funções deste compilador, fornece o conteúdo da memória do endereço dado.

USR – como o BASIC, chama uma rotina em Assembly que inicia no endereço dado. Tem como uso principal chamar rotinas feitas para rodar em conjunto com o programa compilado.

CHR$ – usado com PRINT, imprime o caractere do código dado. Note que você nunca conseguirá um erro B/ por tentar imprimir caracteres de código acima de 255, pois o compilador os imprime em módulo de 256, isto é, a cada 256 caracteres impressos eles começam a se repetir (há casos em que o módulo é 64).

IF...THEN – condicional tradicional, um pouco prejudicada por não poder operar com álgebra booleana (AND, OR, NOT). Porém, essa deficiência será contornada mais à frente.

FOR...TO – controlador de loop. Seu STEP é sempre igual a 1. Porém, mais à frente, iremos ver como simular outros "steps". Esta é uma função muito rápida no compilador.

STOP – esta palavra nada tem a ver com o STOP do BASIC Sinclair. No compilador ela tem como única função indicar aonde acaba o programa a ser compilado. Isto serve para você poder deixar partes em BASIC para fora do que será compilado.

SCROLL – como o computador é forçado a ter no mínimo 16K de memória para operar com o compilador, este scroll é feito com a tela aberta, e portanto, mais rápido que o normal.

REM – o mesmo que no BASIC Sinclair, absolutamente nada. O compilador pula todas que encontra.

GOTO – salto incondicional para a linha dada. É aconselhável que a linha exista, pois ao contrário do BASIC Sinclair, o compilador não procura a próxima existente e fica completamente confuso.

GOSUB – salto incondicional para uma sub-rotina que se encontra na linha dada. A exemplo do GOTO, é recomendável que a linha exista.

LET – usado normalmente para atribuição de variáveis. O compilador somente aceita variáveis de uma letra de A a Y, ou a variável indexada Z (...).

PAUSE – parada temporizada em quadros: de Ø a 32767 quadros (de Ø a 9 min 6 s). Digitar qualquer tecla durante a pausa, a desativa. Ela difere do BASIC Sinclair porque a tela trabalha em slow (ela não "pula"), e não há "debounce", (se houver uma tecla apertada antes de começar a pausa, ela é ignorada).

NEXT – fecha o loop FOR...TO até o valor limite, provoca o incremento da variável de loop e o retorno da execução para a 1ª instrução após o FOR...TO.

POKE – grava na memória de um determinado endereço um valor dado.

PRINT – imprime uma seqüência de caracteres entre aspas, um número, uma variável, um caractere, etc.

PLOT – acende o pixel gráfico correspondente à coordenada dada. Esta função é aproximadamente 25 vezes mais rápida que o "plotador" do BASIC Sinclair. Porém, devido à falta de alguns testes, quando se "plota" ou "unplota" em cima de algum caractere não gráfico (blocos cinzas), caracteres estranhos poderão aparecer na tela.

UNPLOT – apaga o pixel gráfico correspondente à coordenada dada. No mais, é idêntico ao PLOT.

CLS – limpa a tela, porém muito mais rapidamente que o CLS normal usado no BASIC Sinclair.

RETURN – marca aonde o retorno da sub-rotina deve ser feito. Se estiver fora de qualquer sub-rotina, provoca um retorno ao BASIC.

CONDICIONAIS

Em instruções IF...THEN, além de todas as instruções já apresentadas, ainda podemos usar todos os testes de condição:

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

ÁLGEBRA

Por enquanto, a matemática no compilador fica restrita às 4 operações básicas com números inteiros:

+	soma
–	subtração
*	multiplicação
/	divisão

O compilador segue à risca todas as prioridades matemáticas (* e / antes de + e –).

Todos os cálculos realizados fora de uma instrução LET devem estar entre parênteses. Eles também servem caso se queira mudar alguma prioridade matemática.

SIMULANDO FUNÇÕES COMPLEXAS

As funções mais importantes e que não constam no vocabulário do compilador são:

SGN	STEP	SLOW
OR	COPY	FAST
AND	STOP	**

Mostraremos agora, como simulá-las facilmente.

SGN — a função SGN deve fornecer o sinal do operando, ou seja, —1,0 ou 1.

```
10 LET Y=SGN X
20 ...
```

```
10 IF X=0 THEN LET Y=0
20 IF X<>0 THEN LET Y=(X/ABS X
)
30 ...
```

OBS.: a divisão de X por seu módulo dará 1 com o sinal de X (X≠ 0).

OR — a função OR deve associar 2 (duas) condições e tornar a sentença verdadeira, se pelo menos uma delas for verdadeira.

```
10 IF X=A OR Y=B THEN GOTO 100
20 ...
```

```
10 IF X=A THEN GOTO 100
20 IF Y=B THEN GOTO 100
```

OBS.: a maneira mais rápida e mais fácil é separar as condições em outras IF...THEN.

AND — a função AND deve associar 2 (duas) condições e tornar a sentença verdadeira apenas se as duas condições forem verdadeiras.

```
10 IF X=A AND Y=B THEN GOTO 10
0
20 ...
```

```
10 IF X=A THEN IF Y=B THEN GOT
O 100
20 ...
```

OBS.: foi usada uma técnica de aninhamento de condicionais. Note que a instrução GOTO 100 somente será executada se os dois IFs forem verdadeiros.

** — esta função eleva o 1º operando ao 2º operando.

```
10 LET X=X**N
20 ...
```

```
10 LET Y=X
20 FOR A=2 TO N
30 LET X=X*Y
40 NEXT A
50 ...
```

OBS.: o programa anterior somente se mostra necessário se a potência é variável, pois se não, na faixa de trabalho do compilador, fica mais fácil fazer várias multiplicações seguidas (X*X*X*X*X p.ex.).

STEP — o comando STEP faz com que a variável de loop seja incrementada de quanto em quanto se deseja.

```
  10 FOR X=0 TO 10 STEP 2
  20 ...
1000 NEXT X
```

```
  10 FOR X=0 TO 10
  20 ...
 999 LET X=X+1
1000 NEXT X
```

OBS.: na linha imediatamente anterior ao NEXT, deve ser introduzida uma linha que soma à variável de loop o valor do step desejado menos um.

COPY - FAST - STOP - SLOW — Como todas estas funções são simples e não possuem operando, elas são chamadas diretamente por USR. Damos a seguir uma tabela de endereços.

```
10 LET L=USR 2153   ( COPY )
20 LET L=USR 3292   ( STOP )
30 LET L=USR 3875   ( FAST )
40 LET L=USR 3883   ( SLOW )
```

Como o leitor deve ter notado, o vocabulário do compilador é bem completo, provendo o programador de um BASIC relativamente extenso e poderoso.

As principais mudanças foram em torno de números e strings, porém, de fácil aprendizado.

Com todas as informações fornecidas até agora, já é possível escrever jogos usando o compilador.

Divirta-se!

CAPÍTULO 4

PRIMEIROS PROGRAMAS

Agora que você tem em mãos um compilador BASIC funcionando, está na hora de começar a fazer jogos. O primeiro programa deste capítulo é um tradicional jogo de carrinho, escrito da maneira convencional, e o segundo, é o mesmo jogo, porém escrito especialmente para rodar com o compilador. Digite os dois jogos e veja as diferenças nas *soluções técnicas.*

Tanto no BR-116 quanto no Indianápolis, o seu objetivo é conduzir o carro (V) que se dirige para baixo, por entre os tortuosos caminhos das estradas. A única diferença entre ambos é a velocidade.

Seus comandos são: . tecla 5 — esquerda
tecla 8 — direita
tecla Ø — reinício

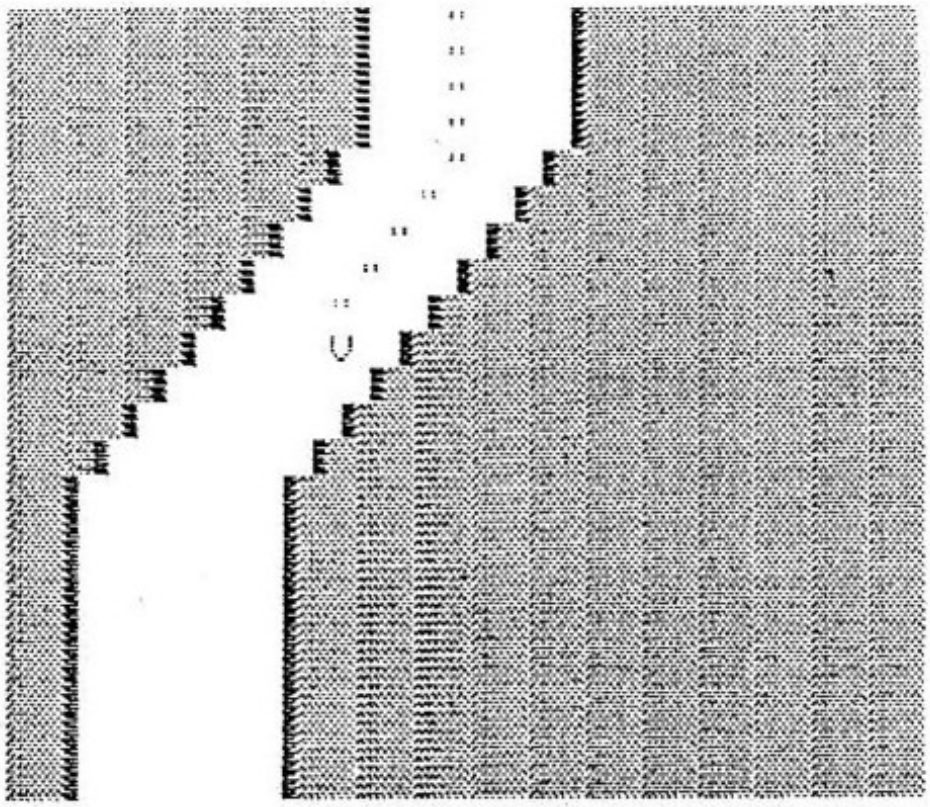
O rastro (") é um ótimo "dedo duro" de suas barbeiragens.

```
MAPA DE VARIAVEIS BR-116

A-COLUNA DO CARRO
B-COLUNA DA ESTRADA
```

```
C-FATOR RANDOMICO
D-PLACAR
E-
F-
G-
H-
I-
J-
K-
L-TECLAS
M-
N-
O-
P-
Q-
R-
S-
T-
U-
V-
W-
X-1■ LOOP
Y-
Z(...)-
```

```
 10 REM ▒▒▒INICIALIZACAO▒▒▒
 20 LET A=15
 30 LET B=12
 40 LET D=0
 50 FOR X=0 TO 21
 60 PRINT AT X,0;"▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒
    ▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒";
 70 NEXT X
 80 REM ▒▒▒ACAO CENTRAL▒▒▒
 90 LET L=CODE INKEY$
100 PRINT AT 9,A;CHR$ 11
110 IF L=33 THEN LET A=A-1
120 IF L=36 THEN LET A=A+1
130 SCROLL
140 PRINT AT 9,A;
150 IF PEEK (PEEK 16398+256*PEE
K 16399)<>0 THEN GOTO 250
160 PRINT AT 9,A;"V";
170 LET D=D+1
180 REM ▒▒▒IMPRESSAO PISTA▒▒▒
190 LET C=RND/3276
200 IF C<4 THEN IF B>2 THEN LET
B=B-1
210 IF C>5 THEN IF B<22 THEN LE
T B=B+1
```

```
 220 PRINT AT 21,0;"
                    ";AT 21,B;"
       ";
 230 GOTO 90
 240 REM
 250 PRINT AT 9,A;"";AT 1,1;D;"
KMS PERCORRIDOS";
 260 STOP
 270 LET L=USR 18823
 280 IF INKEY$<>"0" THEN GOTO 28
0
 290 GOTO 270
```



MAPA DE VARIAVEIS

A-POSICAO CARRO
B-POSICAO ESTRADA
C-DESVIO ESTRADA
D-PLACAR
E-FATOR RANDOMICO
F-
G-
H-
I-
J-
K-
L-TECLAS

```
M-
N-
O-
P-
Q-
R-
S-
T-
U-
V-
W-
X-1■ LOOP
Z(...)-
```

```
  10 REM ▒▒▒INICIALIZACAO▒▒▒
  20 LET G=PEEK 16396+256*PEEK 1
6397
  30 LET A=G+315
  40 LET B=G+694
  50 LET C=12
  60 LET D=0
  70 FOR X=0 TO 21
  80 PRINT AT X,0;"▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒
     ▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒";
  90 NEXT X
 100 REM ▒▒▒ACAO CENTRAL▒▒▒
 110 LET L=CODE INKEY$
 120 POKE A,11
 130 IF L=33 THEN LET A=A-1
 140 IF L=36 THEN LET A=A+1
 150 SCROLL
 160 IF PEEK A<>0 THEN GOTO 330
 170 POKE A,59
 180 LET D=D+1
 190 REM ▒▒▒MONTAGEM PISTA▒▒▒
 200 LET E=RND/3276
 210 IF E>5 THEN IF C<23 THEN LE
T C=C+1
 220 IF E<4 THEN IF C>1 THEN LET
 C=C-1
 230 FOR X=0 TO 31
 240 POKE (B+X),136
 250 NEXT X
 260 POKE (B+C),5
 270 POKE (B+C+7),133
 280 FOR X=1 TO 6
 290 POKE (B+C+X),0
 300 NEXT X
 310 GOTO 110
 320 REM ▒▒▒SAIDA▒▒▒
 330 POKE A,151
```

```
 340 PRINT AT 1,1;D;"KMS PERCORR
IDOS"
 350 STOP
 360 LET L=USR 18823
 370 IF INKEY$<>"0" THEN GOTO 37
0
 380 GOTO 360
```

SOLUÇÕES TÉCNICAS

Como você deve ter percebido com os programas BR-116 e Indianápolis, a função PRINT, apesar de muito mais rápida que no BASIC Sinclair, é lenta para jogos de ação. Isso pode ser facilmente contornado com o uso da função POKE, muito rápida, e com sua reversa, o PEEK. Para poder explorar perfeitamente as funções PEEK e POKE, dê uma olhada no apêndice 2 — tabelas.

Note também o problema para a impressão das aspas (se você tentar imprimir as aspas duplas, ele literalmente imprimirá as 2 aspas na tela (" ")) no BR-116. Porém, no Indianápolis, é colocado no D-FILE diretamente o código das aspas normal (11).

No BR-116 é necessário achar a posição do carro na tela de vídeo, para posteriormente testá-la e verificar a colisão. Porém, no Indianápolis, a posição do carro no D-FILE já é dada pela variável A, e o seu conteúdo por apenas um PEEK.

Achamos que nesta altura, o leitor já está mais do que convencido da superioridade do POKE sobre o PRINT.

COMO ORGANIZAR SEUS JOGOS

Damos aqui, como exemplo, como foi elaborado o Indianápolis.

O primeiro passo para a elaboração de um jogo é a sua proposição. o que ele deve fazer. Após isso, você deve criar um visual compatível com a ação. Preferencialmente, delimite num papel quadriculado uma área de (24 x 33 + 1) quadrados (inclusive os CHR$ 118, os newlines para ter um total controle do vídeo do equipamento.

Com tudo pronto para começar, monte o mapa das variáveis do jogo, de A até Z, pois o compilador não aceita variáveis com mais de uma letra

Tente daqui para frente, manter um padrão das variáveis que você usa. Quando seus jogos começarem a complicar, você vai ver o quanto é útil este procedimento.

Após isso você deve elaborar o jogo, porém, não há necessidade de fazer um fluxograma muito sofisticado, afinal estamos trabalhando em BASIC, apenas um simples fluxograma de blocos é mais do que suficiente. Você deve optar na hora de começar o programa, se vai fazer um programa linear ou um programa estruturado.

Entre estes dois métodos, há diferenças fundamentais quanto à ORGANIZAÇÃO e à VELOCIDADE. Programas que necessitem de altíssima velocidade devem ser feitos linearmente, e programas complexos devem ser feitos estruturadamente.

Programas Lineares – A ação do jogo é "linear", isto é, segue um roteiro linear no programa: executa uma série de rotinas ordenadamente, e em seguida volta ao início para executar tudo novamente.

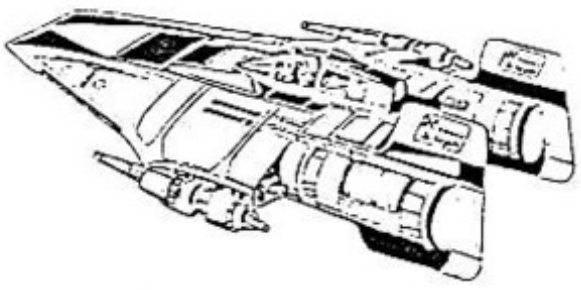
Programas Estruturados – Quando os programas devem controlar várias ações ao mesmo tempo na tela, é preferível que se faça da forma estruturada, isto é, um núcleo central que acessa a várias sub-rotinas específicas, que juntas, formarão a ação do jogo. Este sistema gasta menos memória, e também é mais fácil de ser expandido e complementado.

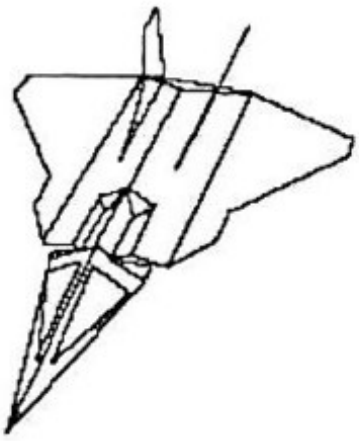
A partir da escolha, elabore as rotinas. É bom separá-las com REMs para melhor identificá-las.

Lembre-se: nunca deixe de gravar o programa em fita após alguma modificação, pois ele pode ficar grande demais para o compilador, e você pode perder tudo por algum erro de lógica. *Qualquer descuido pode ser fatal*!

Após o programa definitivamente pronto (never say "never" again), guarde na sua fita "arquivo de programas compilados" uma versão BASIC do programa. Após compilar, apague o BASIC e faça sua versão "apresentável" do jogo.

Pronto, você acaba de iniciar sua carreira de programador de SOFTWARE.





CAPÍTULO 5

APRIMORANDO SEUS JOGOS

No capítulo anterior, com o Indianápolis e o BR-116, você viu os primeiros programas feitos no compilador. Eram programas simples, porém esse fato não retira seu valor.

Neste capítulo, iremos estudar novas técnicas de controle, que serão mostradas nos programas Space-I e Space-II.

A descrição é a mesma, pois os dois são o mesmo jogo, escritos de maneira diferente.

Seu objetivo é destruir o ladrão espacial antes que ele chegue ao tesouro, que está localizado nas bordas do vale. No fundo desse vale está uma base lançadora de mísseis, que você deve disparar e dirigir de encontro à nave inimiga. Para isso use as teclas:

5 — esquerda
8 — direita
Ø — dispara míssil

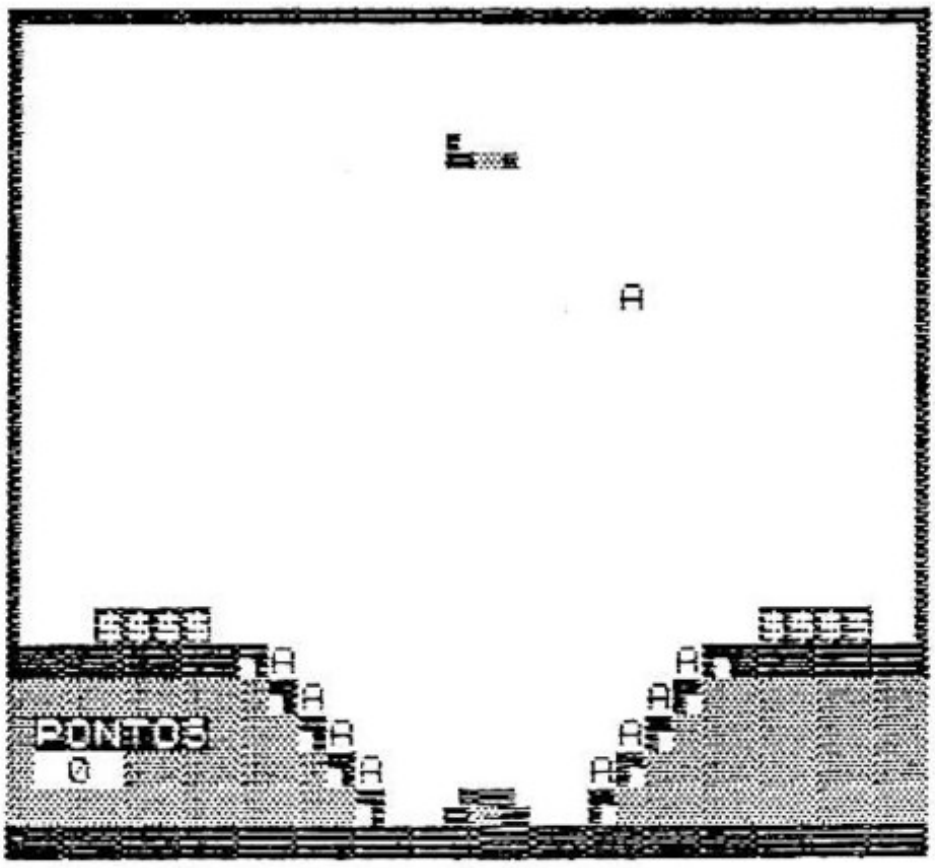
Lembre-se que a cada invasor destruído, outro invade seu espaço aéreo. Se ele conseguir capturar os dois tesouros você perde o jogo.

A seguir será dada apenas a ficha técnica do Space-II, pois a do Space-I é extremamente parecida.

```
 10 REM ▒▒▒ INICIALIZACAO ▒▒▒
 20 LET A=1
 30 LET B=1
 40 LET C=21
 50 LET D=16
 60 LET E=2
 70 LET F=0
 80 REM ▒▒▒ MONTAGEM TELA ▒▒▒
 90 PRINT AT 0,0;"▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀";
100 FOR X=8 TO 42
110 PLOT 0,X
120 PLOT 63,X
130 NEXT X
140 REM ▒▒▒ ACAO CENTRAL ▒▒▒
150 LET B=B+1
160 IF B<>28 THEN GOTO 200
170 PRINT AT A,28;"   ";
180 LET A=A+1
190 LET B=1
200 PRINT AT A,B;" ▙▖";
210 IF A=17 THEN IF B=27 THEN G
OTO 490
220 REM ▒▒▒ LEITURA TECLAS ▒▒▒
230 LET L=CODE INKEY$
240 IF L=33 THEN IF D>2 THEN LE
T D=D-1
250 IF L=36 THEN IF D<29 THEN L
ET D=D+1
260 IF L=28 THEN IF E=2 THEN LE
T E=0
270 REM ▒▒▒ TIRO ▒▒▒
280 IF E=2 THEN GOTO 150
290 IF E=0 THEN LET C=21
300 IF E=0 THEN LET D=16
310 LET E=1
320 LET C=C-1
330 PRINT AT C,D;"A";AT (C+1),(
D-1);"   ";
340 IF A=C THEN IF B=(D-2) THEN
 GOTO 400
350 IF C<>1 THEN GOTO 150
360 PRINT AT C,D;" ";AT 21,16;"
A";
370 LET E=2
380 GOTO 150
390 REM ▒▒▒ EXPLOSAO ▒▒▒
400 PRINT AT A,B;" ▒▒▒";
410 PRINT AT A,B;" XXX";
420 PRINT AT A,B;"    ";
430 LET A=1
440 LET B=1
450 LET E=2
```

```
 460 LET F=F+10
 470 PRINT AT 21,1;F;AT 21,16;"A
";
 480 GOTO 150
 490 STOP
 500 POKE 16418,0
 510 PRINT AT 17,0;"
```

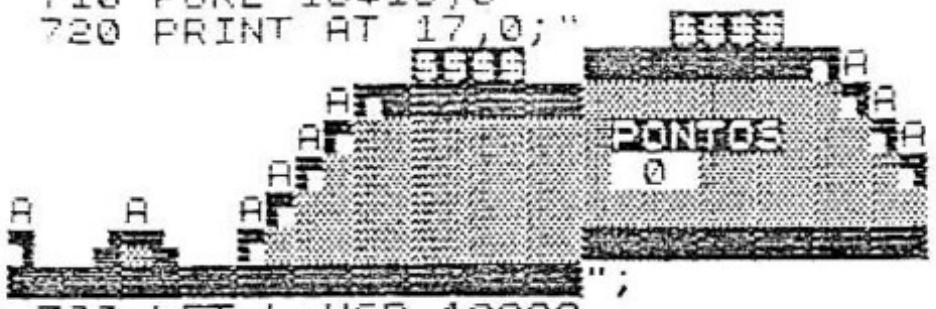

```
";
 520 LET L=USR 18823
 530 PRINT AT 10,10;"FIM DO JOGO
","▌APERTE NEWLINE P/JOGAR DE NO
VO"
 540 IF INKEY$<>CHR$ 118 THEN GO
TO 540
 550 CLS
 560 GOTO 500
```



```
MAPA DE VARIAVEIS SPACE-IT

A-POSICAO ATUAL DA NAVE
B-POSICAO ANTERIOR DA NAVE
C-POSICAO ATUAL DO MISSIL
D-POSICAO ANTERIOR DO MISSIL
E-STATUS DO TIRO
F-PLACAR
G-ENDERECO DO D-FILE
H-CARACTERE DA EXPLOSAO
I-
J-
K-
L-TECLAS
M-
N-
O-
P-
Q-
R-
S-
T-
U-
V-
W-
X-LOOP
Y-
Z(...)-


  10 REM ▒▒▒INICIALIZACAO▒▒▒
  20 LET G=PEEK 16396+256*PEEK 1
6397
  30 LET A=G+34
  40 LET B=A+1
  50 LET C=G+710
  60 LET D=C
  70 LET E=2
  80 LET F=0
  90 REM ▒▒▒MONTAGEM TELA▒▒▒
 100 PRINT AT 0,0;"
";
 110 FOR X=8 TO 42
 120 PLOT 0,X
 130 PLOT 63,X
 140 NEXT X
 150 REM ▒▒▒ACAO CENTRAL▒▒▒
 160 LET A=A+1
 170 IF PEEK (A+2)=133 THEN LET
A=A+5
 180 IF PEEK (A+2)=128 THEN GOTO
 700
```

```
 190 POKE B,0
 200 POKE (B+1),0
 210 POKE (B+2),0
 220 POKE A,130
 230 POKE (A+1),9
 240 POKE (A+2),4
 250 LET B=A
 260 FOR X=1 TO 100
 270 NEXT X
 280 REM ▒▒▒LEITURA TECLAS▒▒▒
 290 LET L=CODE INKEY$
 300 IF L=33 THEN LET C=C-1
 310 IF L=36 THEN LET C=C+1
 320 IF L=28 THEN IF E=2 THEN LE
T E=0
 330 IF E=2 THEN GOTO 160
 340 IF E=0 THEN LET C=G+710
 350 IF E=0 THEN LET D=C
 360 LET E=1
 370 LET C=C-33
 380 IF PEEK C=5 THEN LET C=C+1
 390 IF PEEK C=133 THEN LET C=C-
1
 400 IF PEEK C=131 THEN GOTO 460
 410 IF PEEK C=9 THEN GOTO 510
 420 POKE D,0
 430 POKE C,38
 440 LET D=C
 450 GOTO 160
 460 POKE D,0
 470 LET E=2
 480 POKE (G+710),38
 490 GOTO 160
 500 REM ▒▒▒EXPLOSAO▒▒▒
 510 POKE D,0
 520 LET E=2
 530 LET H=136
 540 GOSUB 640
 550 LET H=189
 560 GOSUB 640
 570 LET H=0
 580 GOSUB 640
 590 LET F=F+1
 600 PRINT AT 21,1;F
 610 LET A=G+34
 620 LET B=A+1
 630 GOTO 160
 640 POKE A,H
 650 POKE (A+1),H
 660 POKE (A+2),H
 670 FOR X=0 TO 1000
 680 NEXT X
 690 RETURN
```

```
700 STOP
710 POKE 16418,0
720 PRINT AT 17,0;"
```




```
";
730 LET L=USR 16823
740 PRINT AT 10,0;"FIM DE JOGO
 N/L PARA CONTINUAR"
750 IF CODE INKEY$<>118 THEN GO
TO 750
760 CLS
770 GOTO 710
```

SOLUÇÕES TÉCNICAS

Se você digitou o programa Space-I, deve ter percebido uma mudança de velocidade quando o invasor desce ou quando o seu míssil sobe, apesar do programa não mudar. Novamente o problema é na rotina PRINT do computador.

Nas posições baixas da tela, a impressão é muito mais rápida que nas posições altas. Isto se dá pelo fato da rotina de PRINT trabalhar de baixo para cima. Note que o Space-II não se ressente desses problemas, pois o POKE não se importa com o endereço, nem se ele é tela ou não.

Outra novidade nesses dois programas é o controle de tiro. É reservada para isso uma variável que pode assumir vários estados, dependendo da ocasião. Neste caso, a variável assume três estados:

Ø — míssil acaba de ser disparado.

1 — míssil já foi disparado e não pode ser requisitado.

2 — míssil já está reativado e pode ser requisitado.

Porém, em outros programas, outros estados podem ser atribuídos a diversas ocasiões.

Uma ocorrência que não pode ser chamada de defeito é a velocidade do Space-II, muito rápido para se jogar. Por causa disso foi incluído um loop fechado, que não controla nada, apenas para perder tempo. Este loop pode ter o seu valor mudado para uma velocidade a gosto do freguês.

Com este capítulo, foi dado mais um passo no aprendizado da programação de jogos, a *Execução Controlada Condicional.* Nestes dois jogos isto pode parecer fácil de fazer de outra maneira, mas em jogos mais complexos é a maneira mais fácil e prática de se fazer a ECC.

CAPÍTULO 6

UMA PONTE PARA O ASSEMBLY

Você já deve ter notado, a essa altura do campeonato, a velocidade fantástica que o compilador pode proporcionar aos seus jogos em BASIC.

Mas você deve ter observado, também, que a velocidade do compilador não é infinita.

Assim, certas rotinas longas, complexas, ou repetitivas, podem gerar atrasos durante a execução do jogo, fato este que constitui uma falha técnica apreciável. A única solução viável para estes atrasos é o uso de sub-rotinas em Assembly puro, cujo tempo de execução é até 5 vezes menor que o BASIC compilado. Para que o uso de Assembly puro seja possível, discutiremos agora onde e como armazená-lo, para não causar problemas de conflito.

Entendemos que a melhor maneira é a menos convencional, já que foi integralmente desenvolvida para coexistir com o compilador e o material compilado.

As técnicas sugeridas são:

I — RAMTOP — O sistema tradicional de armazenamento, seguro quando não se deseja entrar em conflito com o programa BASIC, tem como principal desvantagem que o que fica nessa área não é gravado em fita, tendo que ser armazenado em outro lugar para ser salvo.

II — **Linha REM** — Este sistema é mais prático que o anterior, já que pode ser gravado e carregado diretamente da fita junto com o resto do programa. Infelizmente, o compilador e o material compilado devem estar, respectivamente, na 1ª e 2ª linhas do programa. Deste modo, a linha REM, com os nossos códigos de máquina ficaria obrigatoriamente após a linha 2.

— Qual é o problema? Perguntaria o leitor.

Ocorre que a linha 2 não tem comprimento fixo (seu tamanho depende do programa BASIC que foi compilado), e isso poderia provocar o deslocamento dos códigos de máquina após cada nova compilação (lembre-se de que um programa em desenvolvimento pode requerer dezenas de novas compilações até ficar pronto). Esse fato é inaceitável para um programa em Assembly, pois ele é um programa absoluto, e, com raras exceções, não pode ficar "passeando" pela memória do computador.

III — **Linha 2Ø PRINT compilada** — Este processo é o mais complexo. No entanto, torna-se bastante seguro pelas vantagens que oferece, pessoalmente é o que julgamos o mais eficiente.

Consiste em uma região de memória reservada dentro da linha 2 REM, que se forma a cada nova compilação sempre no mesmo endereço. Isto torna o programa bastante confiável, pois não existe o problema do código de máquina "passear" pela memória, variando os seus endereços. O uso da linha 2Ø PRINT é complexo e deve obedecer às seguintes condições:

1) — A linha 2Ø PRINT deve ser a segunda linha do programa em BASIC. A única linha que *pode e deve* existir antes dela, além das linhas Ø REM e 2 REM é uma linha 1Ø GOTO 3Ø (o número 3Ø é optativo, porém terá que estar entre 21 e 99, nunca maior ou menor que 2 dígitos), que impede que a linha 2Ø PRINT seja executada.

2) — O número de caracteres entre as aspas não pode ser menor que o tamanho da sub-rotina em Assembly. É aconselhável inserir mais caracteres para ter bytes livres que podem ser aproveitados em uma futura rotina. Exemplo: para uma rotina de 22 bytes, faríamos o seguinte programa:

```
10 GOTO 30
20 PRINT "XXXXXXXXXXXXXXXXXXXXX
XX"( 22 CARACTERES ENTRE ASPAS )
30 ... ( AQUI COMECA O BASIC )
```

Para o exemplo acima, após o programa ter sido compilado, o primeiro endereço da área reservada é 18934, e o último é 18955. É bom lembrar que

a cada nova compilação, é necessário refazer os POKEs na região reservada.

Vamos demonstrar este sistema com um joguinho chamado Intruder.

Neste jogo, seu objetivo é atingir as bases "▣" com suas bombas "ıı", e desviar dos meteoros "*". Seus comandos são:

tecla 6 — desce a nave
tecla 7 — sobe a nave
tecla Ø — solta a bomba

Na parte inferior da tela, você verá seu score e o combustível restante. O jogo termina quando seu combustível acaba, ou quando a nave colide com algum objeto.

A rotina em Assembly usada pelo Intruder tem como função realizar um SCROLL para a esquerda, movendo o cenário e dando a impressão da nave estar se movendo para a direita.

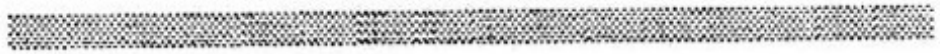
DIGITAÇÃO

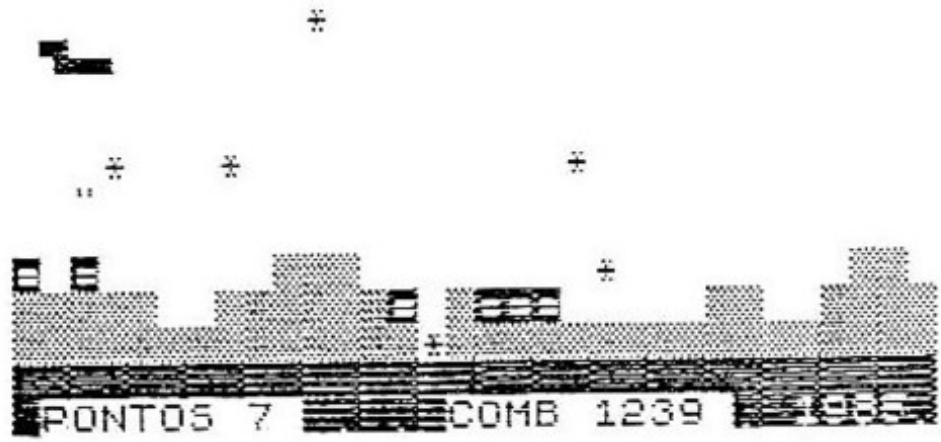
Com o compilador na memória, inicie a digitação com um 1Ø GOTO 3Ø. Esta linha desvia a execução para que a linha 2Ø PRINT não seja executada. Prossiga normalmente com o programa a seguir.

```
MAPA DE VARIAVEIS INTRUDER

A-POSICAO DA NAVE
B-PONTOS
C-COMBUSTIVEL
D-CONTROLE DA BOMBA
E-POSICAO DA BOMBA
F-ALTURA DA MONTANHA
G-ENDERECO DO D-FILE
H-POSICAO INICIAL MONTANHA
I-
J-
K-
L-TECLAS E USR
M-
N-
O-
P-
Q-
R-
S-
T-
U-
V-
W-
```

```
X-1* LOOP E RND
Z(...)-

  10 GOTO 30
  20 PRINT "Digite 23 pontos"
  30 REM INICIALIZACAO
  40 LET G=PEEK 16396+256*PEEK 1
6397
  50 CLS
  60 LET A=G+365
  70 LET B=0
  80 LET C=2000
  90 LET F=5
 100 LET D=2
 110 LET H=G+692
 120 REM MONTAGEM TELA
 130 PRINT AT 0,0;"
                    ";AT 19,0;"
                              ";
"
 PONTOS          COMB        1985
";
 140 REM ACAO CENTRAL
 150 POKE A,0
 160 POKE (A+1),0
 170 POKE (A+2),0
 180 IF C=0 THEN GOTO 540
 190 LET L=CODE INKEY$
 200 IF L=34 THEN LET A=A+33
 210 IF L=35 THEN LET A=A-33
 220 IF L=28 THEN IF D=2 THEN LE
T D=0
 230 LET L=USR 18934
 240 IF PEEK A<>0 THEN GOTO 540
 250 IF PEEK (A+1)<>0 THEN GOTO
540
 260 IF PEEK (A+2)<>0 THEN GOTO
540
 270 POKE A,132
 280 POKE (A+1),131
 290 POKE (A+2),4
 300 REM TIRO
 310 IF D=2 THEN GOTO 410
 320 IF D=0 THEN LET E=A+34
 330 LET D=1
 340 LET E=E+33
 350 IF PEEK E=148 THEN LET B=B+
1
 360 IF PEEK E=148 THEN POKE E,0
 370 IF PEEK E<>0 THEN LET D=2
 380 IF D=1 THEN POKE E,11
```

```
 390 IF D=1 THEN POKE (E-34),0
 400 REM ▒▒▒▒MONTAGEM MONTANHA▒▒▒
 410 IF RND>20000 THEN IF F<15 T
HEN LET F=F+1
 420 IF RND<14000 THEN IF F>1 TH
EN LET F=F-1
 430 FOR X=1 TO F
 440 POKE (H-33*X),8
 450 NEXT X
 460 POKE (G+32),8
 470 IF RND>30000 THEN POKE (H-3
3*X),148
 480 LET X=RND/4096
 490 IF (X+F)>18 THEN LET X=0
 500 IF RND>28000 THEN POKE (H-3
3*(X+F)),23
 510 PRINT AT 21,7;B;AT 21,19;C;
"■";
 520 LET C=C-1
 530 GOTO 150
 540 STOP
 550 LET L=USR 16623
 560 IF INKEY$<>CHR$ 118 THEN GO
TO 560
 570 GOTO 550
```





Agora que você acabou, NÃO rode ainda o programa. Ele ainda não foi compilado, e nem possui a rotina em Assembly.

Para a compilação, proceda como já visto anteriormente, e a seguir entre com os códigos em HEXADECIMAL. Use o monitor de entrada chamado *Carregaprint*.

```
9010 PRINT "QUANTOS BYTES ?"
9020 INPUT A
9030 DIM A$(A)
9040 FOR X=1 TO A
9050 SCROLL
9060 INPUT B$
9070 PRINT B$
9080 LET A$(X)=CHR$ (16*CODE B$+
CODE B$(2)-476)
9090 NEXT X
9100 STOP
9110 FOR X=1 TO A
9120 POKE 18933+X,CODE A$(X)
9130 NEXT X
9140 PRINT "PODE RODAR"
```

O Carregaprint funciona da seguinte maneira:

Da linha 9010 à 9090, ele guarda na matriz A$ e na variável A, respectivamente, o programa ASSEMBLY e o seu tamanho.

Da linha 9110 à 9140, após compilado o programa principal, ele descarrega o programa em Assembly na área reservada.

Após digitar o Carregaprint, proceda da seguinte maneira:

- Digite GOTO 9010
- Quando o programa pedir "QUANTOS BYTES", digite 23, isto é, o tamanho da rotina Assembly.
- Entre com os números em HEX, cada um seguido de NEW LINE.

```
2A 0C 40 3E 14 01 1F
00 23 E5 D1 23 C5 ED
B0 2B 70 C1 3D C8 23
18 F1
```

- Grave o programa para evitar imprevistos.
- Certifique-se que o Intruder já foi compilado; caso contrário compile-o.
- Digite GOTO 9110 para transferir os códigos de máquina para o jogo.

Pronto. Para rodá-lo, digite LET L = USR 18823, e tenha uma boa invasão.

SOLUÇÕES TÉCNICAS

Já basta de comparações entre a velocidade do PRINT e POKE, falaremos agora de outro defeito. No Intruder, durante a ação central, a única vez que foi usado PRINT foi para a impressão dos pontos e combustível restante. Nota-se nessa linha, que foi impresso logo após o combustível, um espaço inverso. Isto foi feito, pois o combustível é decrementado, e, quando passa de 1000 para 999, de 100 para 99 e, de 10 para 9, "encolhe" um dígito, deixando um espaço indesejável atrás de si.

APRIMORANDO O SISTEMA

Como você já deve ter percebido, a área reservada não passa de uma réplica da linha 20 PRINT, ou mais exatamente, do conteúdo entre suas aspas. E você então, poderia perguntar:

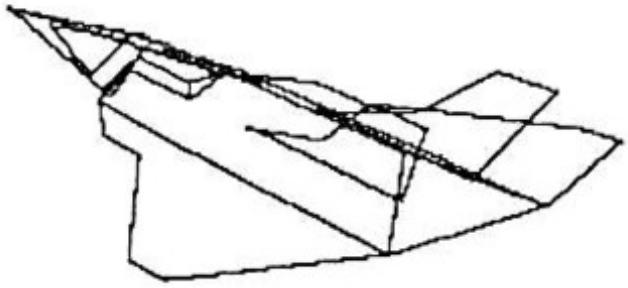
— Porque ficar POKando os códigos de máquina na linha 2, se ela deve ser apagada quando se deseja compilar novamente?

Na verdade, parece-nos mais sensato dar os POKEs dentro da linha 20 PRINT, para quando for criada a área reservada na linha 2 REM, ela já contenha os códigos de máquina na posição correta.

Na prática, o sistema é bastante simples, e para usá-lo, apague a linha 2 e altere o Carregaprint para:

```
9120 POKE 18835+X,CODE A$(X)
```

Digite, então GOTO 9110 e LIST 20. Note que a linha 20 PRINT está cheia de caracteres estranhos entre as aspas. Se você compilar agora, não mais precisará dar GOTO 9110, pois a área reservada será preenchida automaticamente durante a compilação. Com isso, o Carregaprint torna-se desnecessário e pode ser apagado.



CAPÍTULO 7

COMPILE O SEU JOYSTICK

Você deve ter percebido que os jogos anteriores não envolviam grande complexidade de movimentos. Também deve ter percebido no Intruder, que não se pode soltar bombas enquanto se movimenta. Isto se dá pelo fato da rotina INKEY$ (tanto no BASIC SINCLAIR como no compilador), não reconhecer quando mais de uma tecla é pressionada (exceto SHIFT).

Isto é um grande problema para jogos de alta complexidade, onde é necessário ler duas ou mais teclas de uma vez.

Uma maneira simples, mas não muito sensata, seria a leitura dos bytes 16421 e 16422, que guardam uma codificação especial do teclado.

O número 65535 corresponde a nenhuma tecla pressionada, e este valor é decrementado, dependendo da tecla ou das teclas pressionadas. A simples comparação do valor obtido com valores previamente estabelecidos seria uma maneira.

Exemplo:

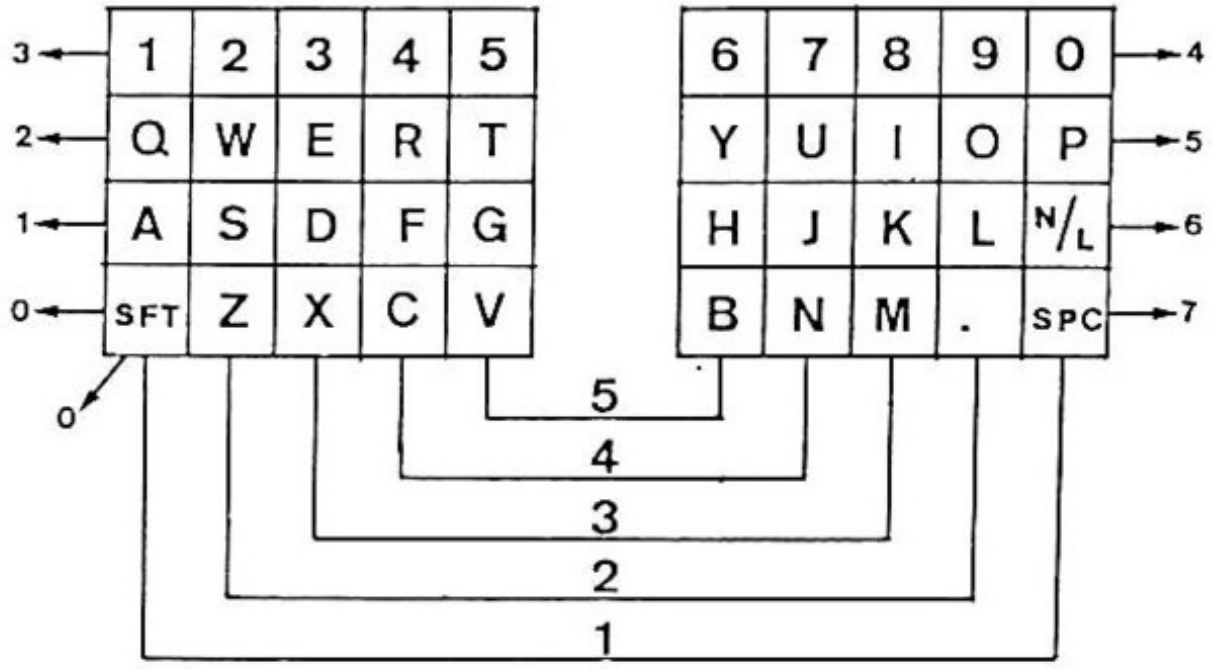
```
LET A=PEEK 16421+256*PEEK 16422
IF A=65007 THEN ( TIRO )
IF A=57335 THEN ( ESQUERDA )
IF A=63471 THEN ( DIREITA )
IF A=56807 THEN ( TIRO )
IF A=56807 THEN ( ESQUERDA )
IF A=62959 THEN ( TIRO )
IF A=62959 THEN ( DIREITA )
```

Esta é uma simples rotina em BASIC compilado que distingue duas teclas pressionadas ao mesmo tempo, que poderia perfeitamente ser usada num programa tipo Space Invaders.

Até esse limite de teclas esta rotina é aceitável, pois gasta apenas 260 bytes aproximadamente. Porém, para o controle de 5 teclas (5, 6, 7, 8 e Ø: o joystick), a situação complica-se, pois são necessárias 34 linhas, e não mais 8, o que leva o gasto de memória para 195Ø bytes aproximadamente. Gastar quase 2Kbytes de memória somente para ler teclas não é um bom negócio, porém esta mesma rotina pode ocupar apenas 81 bytes! Quer saber como? Então analise conosco como funciona o controle do teclado.

O TECLADO

Particularmente, o teclado do Sinclair trabalha como uma matriz de 5 colunas por 8 linhas (8 x 5 = 4Ø teclas), dispostas da seguinte maneira:



Convém aqui uma explicação, que diferenciará o que o teclado originalmente faz do que a rotina de teclado faz.

Fisicamente nenhuma tecla possui setores especiais para ela, pois cada uma faz parte da matriz como qualquer outra.

Porém, quando o programa de teclado recebe as informações do mesmo, através de um circuito integrado específico para isto, ele faz certas modificações. Ele separa a tecla SHIFT dos setores Ø horizontal e 1 vertical, e cria para ela um único setor vertical Ø.

Após esta pequena explicação sobre setorização, explicaremos o que faz a rotina de teclado.

Para armazenar o estado de cada setor, o programa usa 2 registradores da Z-80A para guardá-los, bit a bit. Normalmente, quando nenhuma tecla é pressionada, todos os bits estão em 1, porém se pressionarmos uma tecla os bits correspondentes ao seu setor horizontal e vertical serão levados a 0. Isto também acontece quando mais de uma tecla é pressionada ao mesmo tempo.

Pronto, achamos a outra maneira.

Porém, há um problema, pois em BASIC não temos uma instrução para testar bits, e sua simulação talvez seria mais difícil que a nossa rotina anterior de 2Kbytes. O que fazer?

Talvez a nossa única solução seria apelar para o Assembly, que possui instruções de manipulação de bits, em especial, a instrução BIT, que testa bits "acesos" e "apagados".

Explicaremos detalhadamente as rotinas em Assembly que se seguem. Elas são denominadas JOY 1, JOY 2 e JOY 3.

A rotina JOY 1, mais simples, destina-se a jogos tipo Space Invaders e Racers, que necessitem de movimento apenas na horizontal e tiro ou acelerador ao mesmo tempo.

```
      XXXX NOP          ; estes três bytes e' que indicarão que
           NOP          ; teclas foram pressionadas.
           NOP          ;
entrada    LD HL,XXXX   ; XXXX guarda o endereço dos 3 bytes.
           PUSH HL      ;
           LD B,3       ; limpa os 3 buffers.
           LD (HL),FF   ;
           INC HL       ;
           DJNZ,-5      ;
           CALL 02BB    ; chama a rotina de teclado e arruma os
           PUSH HL      ; registradores .
           POP DE       ;
           POP HL       ;
           BIT 5,D      ; tecla 5.
           JRNZ,+2      ;
           LD (HL),00   ;
           INC HL       ;
           BIT 3,D      ; tecla 8.
           JRNZ,+2      ;
           LD (HL),00   ;
           INC HL       ;
           BIT 1,D      ; tecla 0.
           JRNZ,+2      ;
           LD (HL),00   ;
           RET          ; retorno.
```

```
códigos:
   00 00 00 21 xx xx E5 06 03 36 FF 23 10 FB
   CD BB 02 E5 D1 E1 CB 6A 20 02 36 00 23 CB
   5A 20 02 36 00 23 CB 4A 20 02 36 00 C9
```

A rotina JOY 2, igualmente simples, reconhece movimentos na vertical e tiro. Pode ser usada em jogos tipo Scrambler.

```
 XXXX    NOP           ; estes três bytes e' que indicarão que
         NOP           ; teclas foram pressionadas.
         NOP           ;
entrada  LD HL,XXXX    ; XXXX guarda o endereço dos 3 bytes.
         PUSH HL       ;
         LD B,3        ; limpa os 3 buffers.
         LD (HL),FF    ;
         INC HL        ;
         DJNZ,-5       ;
         CALL 02BB     ; chama a rotina de teclado e arruma os
         PUSH HL       ; registradores .
         POP DE        ;
         POP HL        ;
         BIT 5,D       ; tecla 6.
         JRNZ,+2       ;
         LD (HL),00    ;
         INC HL        ;
         BIT 4,D       ; tecla 7.
         JRNZ,+2       ;
         LD (HL),00    ;
         INC HL        ;
         BIT 1,D       ; tecla 8.
         JRNZ,+2       ;
         LD (HL),00    ;
         RET           ; retorno.
```

```
códigos:
   00 00 00 21 xx xx E5 06 03 36 FF 23 10 FB
   CD BB 02 E5 D1 E1 CB 6A 20 02 36 00 23 CB
   62 20 02 36 00 23 CB 4A 20 02 36 00 C9
```

As duas rotinas JOY 1 e JOY 2, são praticamente idênticas, a não ser por um único byte, que distingue a tecla 7 da tecla 8, que são de setores verticais diferentes, pois tanto a tecla 5 quanto a tecla 6 pertencem ao mesmo setor vertical, e o teste é o mesmo.

A rotina que se poderia chamar de genérica é a rotina JOY 3. Ela é quase que uma mistura das rotinas JOY1 e JOY 2, pois lé 5 teclas ao mesmo tempo (5, 6, 7, 8 e 0, ou Joystick), possibilitando movimentos múltiplos em oito sentidos (esquerda. direita, para cima, para baixo e as quatro diagonais \ / / \ ) simultaneamente com o tiro.

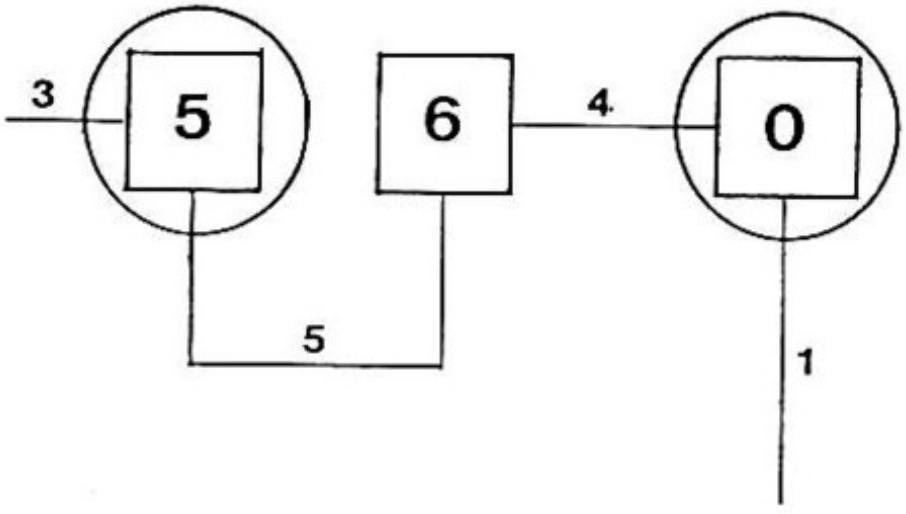
```
XXXX    NOP          ; estes cinco bytes e' que indicarão que
        NOP          ; teclas foram pressionadas.
        NOP          ;
        NOP          ;
        NOP          ;
entrada LD HL,XXXX   ; XXXX guarda o endereço dos 3 bytes.
        PUSH HL      ;
        LD B,5       ; limpa os 5 buffers.
        LD (HL),FF   ;
        INC HL       ;
        DJNZ,-5      ;
        CALL 02BB    ; chama a rotina de teclado e arruma os
        PUSH HL      ; registradores .
        POP DE       ;
        POP HL       ;
        BIT 1,D      ; tecla 0.
        JRNZ,+2      ;
        LD (HL),00   ;
        INC HL       ;
        BIT 3,D      ; tecla 8.
        JRNZ,+2      ;
        LD (HL),00   ;
        INC HL       ;
        BIT 4,D      ; tecla 7.
        JRNZ,+2      ;
        LD (HL),00   ;
        INC HL       ;
        BIT 3,E      ; tecla 5.
        JRNZ,+2      ;
        LD (HL),00   ;
        INC HL       ;
        JR Z,+6      ; tecla 6.
        BIT 5,D      ;
        JRNZ,+2      ;
        LD (HL),00   ;
        BIT 3,E      ; diagonal 5 e 6.
        JRNZ,+18     ;
        BIT 5,D      ;
        JRNZ,+14     ;
        BIT 4,E      ;
        JRNZ,+10     ;
        BIT 1,D      ;
        JR Z,+6      ;
```

```
BIT 4,D      ;
JR Z,+2      ;
LD (HL),00   ;
RET          ; retorna.
```

```
códigos:
   00 00 00 00 00 21 xx xx E5 06 05 36 FF 23
   10 FB CD BB 02 E5 D1 E1 CB 4A 20 02 36 00
   23 CB 5A 20 02 36 00 23 CB 62 20 02 36 00
   23 CB 5B 20 02 36 00 23 28 06 CB 6A 20 02
   36 00 CB 5B 20 12 CB 6A 20 0E CB 63 20 0A
   CB 4A 28 06 CB 62 28 02 36 00 C9
```

A rotina JOY 3, apesar de ler apenas duas teclas a mais, tem o dobro do tamanho de JOY 1 ou JOY 2. Isto ocorre pelo problema das teclas 5 e 6 pertencerem ao mesmo setor vertical, e a identificação do pressionamento simultâneo dessas duas teclas é mais complicado que parece.

A rotina JOY 3 reconhece todas as combinações simultâneas de teclas, a não ser o pressionamento simultâneo das teclas 5, 6 e Ø. Isto não é uma deficiência desta rotina, pois é **IMPOSSÍVEL** distinguir do acionamento das teclas 5 e Ø o das teclas 5, 6 e Ø, pois no primeiro, já foram usados todos os setores da tecla 6, e o computador fica indiferente ao seu acionamento.



Quando acionamos a tecla 5,usamos o setor 5 vertical e o 3 horizontal.
Quando acionamos a tecla Ø,usamos o setor 1 vertical e o 4 horizontal.
Note que os dois setores usados pela tecla 6 (o 4 hor.e o 5 vert.) já foram usados e tanto faz você pressionar ou não a tecla 6.

Porém, não será esta pequena deficiência que atrapalhará o seu programa.

COMO USAR AS ROTINAS JOY

Todas as rotinas são completamente auto-relocáveis, podendo ficar em qualquer parte da memória, apenas pela mudança de XXXX, dois bytes que marcam o endereço inicial dos buffers.

E são justamente estes buffers que ajudarão a identificação das teclas. Basta um PEEK no buffer: se ele contiver Ø, a tecla foi pressionada, se ele contiver 255, a tecla não foi pressionada. Com isso baixam para apenas 6 linhas a leitura independente de cinco teclas.

O uso das rotinas JOY será demonstrado em quase todos os jogos daqui para frente. Em caso de dúvida, dê uma olhada no programa Galactica, neste capítulo.

IMPRESSÃO DE NÚMEROS

Se você alguma vez já teve que imprimir números na tela em BASIC Sinclair, deve ter notado que é um processo sonolento demais. No compilador este é um processo bastante rápido, cerca de 56 vezes mais veloz que o BASIC Sinclair, porém ainda se mostra lento quando queremos imprimir simultaneamente vários números na tela.

Apresentaremos agora uma pequena rotina em Assembly que imprime números de até 5 dígitos que não sejam maiores que 65535.

Esta rotina é extremamente rápida, cerca de 232 vezes mais rápida que BASIC Sinclair, porém ainda lenta quando queremos imprimir simultaneamente vários números na tela.

A principal deficiência da rotina impressora de números do compilador é o fato do comprimento do número variar com o seu valor (ela também coloca um espaço de cada lado do número, o que, às vezes não é interessante).

A rotina apresentada a seguir é genérica apenas para exemplificar como imprimir os números.

```
LD BC,10000 : seleciona e chama 5- digito
CALL conta  :
LD BC,1000  : seleciona e chama 4- digito
CALL conta  :
LD BC,100   : seleciona e chama 3- digito
CALL conta  :
LD BC,10    : seleciona e chama 2- digito
CALL conta  :
LD BC,1     : seleciona e chama 1- digito
```

```
conta SUB A          ; rotina que calcula o digito
      LD A,28        ; e o imprime na posicao correta.
      SBC HL,BC      ;
      JR C,+3        ;
      INC A          ;
      JR, -7         ;
      ADD HL,BC      ;
      LD (DE),A      ;
      INC DE         ;
      RET            ;
```

Para se entrar na rotina, deve-se ter no registrador HL o número que se quer imprimir e no registrador DE o início de onde se quer imprimir, para posteriormente entrar na rotina dos dígitos desejados.

OBS.: Deve-se entrar numa chamada em que caibam os dígitos do número contido no registrador HL (não se deve chamar uma rotina de quatro dígitos, por exemplo, se quisermos imprimir o número 15ØØØ).

Apresentamos agora, um jogo que utiliza os recursos mostrados neste capítulo: Galactica.

Galactica:

Sozinho em seu caça "Víbora", você está em uma missão de patrulhamento de rotina quando avista algo no quadrante Épsilon. Mesmo fazendo contato com a estranha nave, ela se recusa a identificar-se. Só então você reconhece o símbolo estampado em suas asas: o símbolo de Cylon, povo inimigo que tenta exterminar a raça humana desde que a guerra eclodiu há mil anos atrás.

Eis que de repente, o sinal de alerta de seu painel acende: você acaba de ser detectado pelo inimigo. Não há sequer tempo para reportar à Galactica, pois a bateria de lasers dos cylônios já está sendo carregada. Você dispõe de poucos segundos para destrui-los, antes que eles dêem seu tiro certeiro e o "Víbora" se transforme em fragmentos e poeira cósmica.

Seus controles são as teclas 5, 6, 7, 8 e Ø, ou o joystick. No centro da tela existe uma mira na qual a nave cylônica deve ser enquadrada para ser atingida. Na linha inferior é mostrado o número de lasers restantes, quantas naves foram interceptadas, e a carga do laser inimigo. Seja rápido, pois se esta atingir a marca 1ØØ, não haverá escapatória. Sua missão encerra-se quando seus lasers acabam, ou quando a sua nave é destruída.

Digitação:

Carregue o compilador e monitor que você armazenou em fita conforme as instruções do Apêndice I. Comande BREAK e digite a seguir o seguinte programa em BASIC, da linha 10 a 1020.

Chame o monitor digitando GOTO 9580 e escolha a opção 2 (entrar dados). Ao ser perguntado sobre o endereço inicial responda 18836 e intrc-duza os códigos hexadecimais listados antes do programa em BASIC que você já digitou.

```
*18836 00 00 00 00 00 21 F6 279
 18843 49 E5 06 05 36 FF 23 657
*18850 10 FB CD BB 02 E5 D1 1099
 18857 E1 CB 4A 20 02 36 00 590
*18864 23 CB 5A 20 02 36 00 416
 18871 23 CB 62 20 02 36 00 424
*18878 23 CB 5B 20 02 36 00 417
 18885 23 28 06 CB 6A 20 02 424
*18892 36 00 CB 5B 20 12 CB 601
 18899 6A 20 0E CB 63 20 0A 496
*18906 CB 4A 28 06 CB 62 28 664
 18913 02 36 00 C9 2A 0C 40 375
*18920 11 9C 02 19 EB 2A BD 666
 18927 49 CD 88 4A 2A D5 49 784
*18934 06 08 13 10 FD CD 88 611
 18941 4A 2A 91 49 05 0A 13 369
*18948 10 FD 01 0A 00 CD 71 598
 18955 4A 01 01 00 97 3E 1C 317
*18962 ED 42 38 03 3C 18 F9 695
 18969 09 12 13 C9
```

```
  10 GOTO 30
  20 PRINT "Digite 137 pontos"
  30 REM ▒▒INICIALIZACAO▒▒
  40 LET O=PEEK 16396+256*PEEK 1
6397+1
  50 LET N=0
  60 LET T=50
  70 REM ▒▒DEFINE CYLONIO▒▒
  80 LET X=RND/1214
  90 LET Y=RND/1725
 100 LET C=0
 110 REM ▒▒PAINEL DA NAVE▒▒
 120 PRINT AT 19,0;"
                    ▒NAVES=";AT
20,9;"▒LASER=";AT 20,19;"▒ALER
TA=0";AT 20,31;"▒";
 130 LET J=USR 19015
 140 IF T=0 THEN GOTO 940
 150 LET P=O+33*Y+X
```

```
 160 REM ▒▒▒DESENHA CYLONIO▒▒▒
 170 POKE P,22
 180 POKE (P+1),23
 190 POKE (P+2),52
 200 POKE (P+3),23
 210 POKE (P+4),22
 220 PRINT AT 10,15;"<#>"
 230 LET C=C+1
 240 IF C=100 THEN GOTO 740
 250 LET A=X
 260 LET B=Y
 270 LET J=USR 18939
 280 REM ▒▒▒LEITURA TECLAS▒▒▒
 290 IF PEEK 18934=0 THEN GOTO 5
30
 300 IF PEEK 18937=0 THEN LET X=
X+1
 310 IF PEEK 18935=0 THEN LET X=
X-1
 320 IF PEEK 18938=0 THEN LET Y=
Y+1
 330 IF PEEK 18936=0 THEN LET Y=
Y-1
 340 PAUSE 1
 350 REM ▒▒▒MOVE CYLONIO▒▒▒
 360 IF RND>10000 THEN GOTO 390
 370 LET X=X+INT (RND/10923)-1
 380 LET Y=Y+INT (RND/10923)-1
 390 IF X>27 THEN LET X=27
 400 IF X<0 THEN LET X=0
 410 IF Y<0 THEN LET Y=0
 420 IF Y>18 THEN LET Y=18
 430 IF X=A THEN IF Y=B THEN GOT
O 130
 440 LET P=0+33*B+A
 450 REM ▒▒▒APAGA CYLONIO▒▒▒
 460 POKE P,0
 470 POKE (P+1),0
 480 POKE (P+2),0
 490 POKE (P+3),0
 500 POKE (P+4),0
 510 GOTO 130
 520 REM ▒▒▒LASER▒▒▒
 530 FOR M=1 TO 8
 540 POKE (0+33*(19-M)+7+M),6
 550 POKE (0+33*(19-M)+25-M),134
 560 NEXT M
 570 LET T=T-1
 580 IF Y=10 THEN IF X>=12 THEN
IF X<=16 THEN GOTO 650
 590 FOR M=1 TO 8
 600 POKE (0+33*(19-M)+7+M),0
 610 POKE (0+33*(19-M)+25-M),0
```

```
 620 NEXT M
 630 GOTO 120
 640 REM ▒▒▒NA MOSCA▒▒▒
 650 PRINT AT Y,X;"▒▒▒▒▒▒▒";AT Y,X
;" ▒▒▒ ";AT Y,X;"  ▒▒▒  ";AT Y,(X+
2);"  ";AT 10,15;"<*>"
 660 LET N=N+1
 670 REM ▒▒▒APAGA LASER▒▒▒
 680 FOR M=1 TO 8
 690 POKE (0+33*(19-M)+7+M),0
 700 POKE (0+33*(19-M)+25-M),0
 710 NEXT M
 720 GOTO 80
 730 REM ▒▒▒ + MORREU + ▒▒▒
 740 PRINT AT 20,28;"100"
 750 LET Q=P+1
 760 LET R=P+3
 770 IF Q=0 THEN GOTO 810
 780 LET Q=Q+32
 790 IF PEEK Q>0 THEN LET Q=0
 800 POKE Q,6
 810 IF R=0 THEN GOTO 850
 820 LET R=R+34
 830 IF PEEK R>0 THEN LET R=0
 840 POKE R,134
 850 IF (Q+R)<>0 THEN GOTO 770
 860 FOR M=1 TO 2000
 870 NEXT M
 880 FOR M=1 TO 1000
 890 PLOT (RND/512),(5+RND/850)
 900 NEXT M
 910 PRINT AT 10,8;"NAVE DESTRUI
DA"
 920 GOTO 950
 930 REM ▒▒▒FIM DOS LASERS▒▒▒
 940 PRINT AT 10,8;"FIM DA ENERG
IA"
 950 STOP
1000 REM ▒▒▒JOY3▒▒▒
1010 SLOW
1020 LET L=USR 16823
```

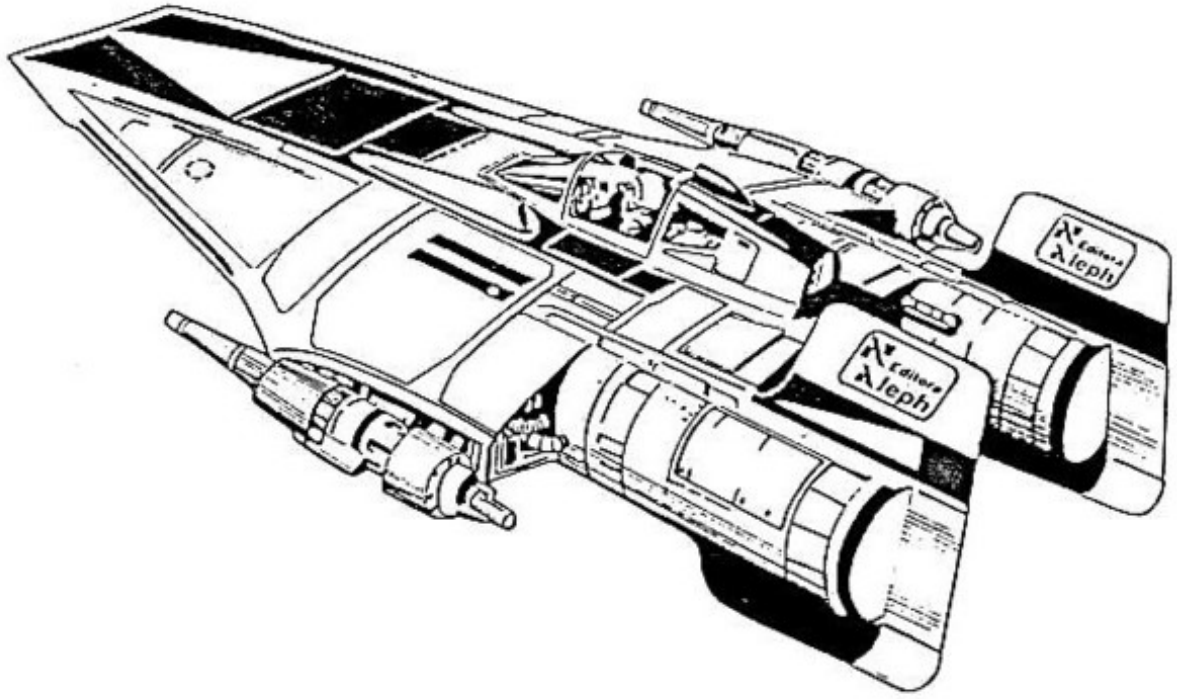
SOLUÇÕES TÉCNICAS – GALACTICA

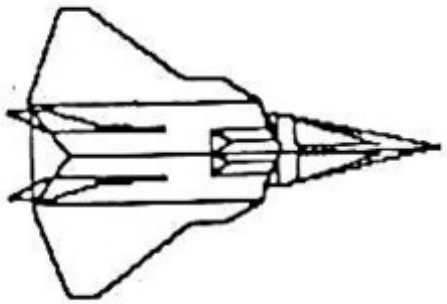
O jogo está utilizando duas rotinas deste capítulo: a JOY 3 e o impressor para 2 dígitos. Este último, em particular, está operando de uma forma muito interessante, pois imprime todos os três números com uma única chamada USR, e seus valores são pesquisados diretamente das variáveis do compilador. Por falar nelas, o Apêndice I traz explicações detalhadas de como pesquisar e alterar seus valores com PEEK e POKE.

A maioria dos leitores já deve ter percebido como foi possível imprimir o valor de "ALERTA = 100" com a rotina para apenas dois dígitos. Se você não descobriu ainda, dê uma olhada mais atenta para as linhas 230, 240 e 740. Esta pequena manobra serviu para que não fosse necessário definir uma rotina impressora de três dígitos (mais longa) por causa de apenas um número impresso em um instante.

A rotina JOY 3, usada entre as linhas 270 e 330, não oferece maiores segredos para quem já a viu listada neste mesmo capítulo. Quanto à linha 340, ela só serve para liberar a tecla BREAK durante a execução do jogo, e pode ser removida sem prejuízo para o mesmo.

O resto da listagem não passa de trivialidades do Basic, e a esta hora todos já devem ter entendido seu funcionamento.





CAPÍTULO 8

ELEMENTOS DE TEMPORIZAÇÃO

Na medida em que nos aprofundamos no conjunto das técnicas de programação, vamos progressivamente adotando sistemas de desenvolvimento de jogos que facilitam nosso trabalho, dando-nos o poder de criar cada vez mais. Você já conhece o compilador, a rotina de joystick, a rotina de placar, manipulação de sub-rotinas em Assembly, e agora vai aprender a temporizar seus jogos.

O processo de contagem de tempo é o que denominamos "tempo real", assim chamado porque marca um intervalo de tempo pré-determinado sem sofrer interferência do restante do processamento. Se você programar para que um programa marque um minuto, ele o fará com precisão, mesmo que sofra desvios, caia em loops ou sofra atrasos de todo o tipo.

Como funciona:

Muitos leitores já devem conhecer uma variável do sistema chamada "FRAMES", contida nos bytes 16436 e 16437. O primeiro é o byte menos significativo, o segundo é o mais significativo.

Ela serve para temporizar o PAUSE do BASIC, mas pode ser usada

pelo operador para fazer suas próprias temporizações concomitantemente com o processamento geral.

Quando o computador é ligado, FRAMES inicia com o valor 65535, ou seja, os dois bytes estão com o valor 255. A cada quadro enviado para a televisão (1/60 de segundo), o byte 16436 perde uma unidade até chegar a 0, quando então é reciclado para 255 e o byte 16437 perde uma unidade. O curioso é que quando 16437 chega a 128, ele é automaticamente reciclado a 255. A explicação disto é que o PAUSE usa a FRAMES com valores de 16437 menores que 128, e as outras operações usam valores superiores a 127.

Convém aqui colocar um alerta: *nunca* "pokeie" valores menores que 128 em 16437, pois neste caso o computador pensa que está realizando um PAUSE. Aparentemente nada acontece no momento, mas quando o valor de 16437 atinge 0, o sistema "sai do ar" e o programa é perdido.

Utilizando a temporização:

Os dois bytes podem ser usados para temporização, sendo que o byte 16436 serve melhor para temporizar tempos pequenos (até 4 s), e o byte 16437 serve para tempos maiores (até 9 min). Para marcar-se o tempo desejado, basta testar regularmente o valor de um dos bytes, e verificar se perdeu tantas unidades quantas sejam necessárias para o dado tempo (no byte 16436, cada unidade corresponde a 16,67 milissegundos, e no byte 16437, cada unidade corresponde a 4,27 segundos).

A listagem a seguir cria um relógio digital, na tela, com horas, minutos e segundos. Os números são impressos com a rotina de placar do capítulo 7, e os ":" são impressos em Assembly. A temporização, por sua vez, é bastante simples: coloca-se um valor conhecido em 16436 (255, no caso) e espera-se até que ele atinja o valor 195 (60 unidades a menos). Lembre-se que 60 unidades deste byte correspondem a 1 segundo de tempo que o micro aguarda antes de incrementar o relógio e reimprimir seus valores.

```
*18836 2A 0C 40 11 55 01 19 246
 18843 EB 2A A5 49 CD 15 4A 815
*18850 3E 0E 12 13 2A B9 49 413
 18857 CD 15 4A 3E 0E 12 13 413
*18864 2A D1 49 01 0A 00 CD 540
 18871 1E 4A 01 01 00 97 3E 319
*18878 10 ED 42 38 03 3C 18 474
 18885 F9 09 12 13 C9
```

```
 10 GOTO 30
 20 PRINT "Digite 54 pontos"
 30 LET H=0
 40 LET M=0
 50 LET S=0
 55 REM ▒▒▒▒PAUSA 1 SEGUNDO▒▒▒▒
 60 POKE 16436,255
 70 IF PEEK 16436>195 THEN GOTO
70
 80 LET J=USR 18934
 90 LET S=S+1
 95 REM ▒▒▒▒INCREMENTA RELOGIO▒▒▒▒
100 IF S<60 THEN GOTO 130
110 LET S=0
120 LET M=M+1.
130 IF M<60 THEN GOTO 160
140 LET M=0
150 LET H=H+1
160 IF H=24 THEN LET H=0
170 LET J=CODE INKEY$
180 GOTO 60
190 STOP
```

Observações:

1) O maior tempo que pode ser marcado é 9 minutos, aproximadamente. Neste tempo, o byte 16437 parte de 255 e chega a 128, seu último valor.

2) A temporização deve ser feita no modo SLOW, e neste período não se deve chamar o comando PAUSE (nem mesmo o compilado), pois ele altera a variável FRAMES e interfere na temporização.

Super-Máquina

O jogo a seguir é uma aplicação prática do sistema de tempo real. Trata-se de um carro que deve percorrer uma pista sinuosa e esburacada, e como se isto não bastasse, manter uma média de 900 km por minuto (só a Super Máquina é capaz de fazer isto...).

Os controles do carro são:

5 ..... esquerda
8 ..... direita
Ø ..... acelera

É aconselhável manter a velocidade máxima, mas cuidado para não colidir com os buracos! Você ganha um minuto extra a cada 900 km percorridos. O jogo acaba quando você não conseguir manter os 900 km cada minuto.

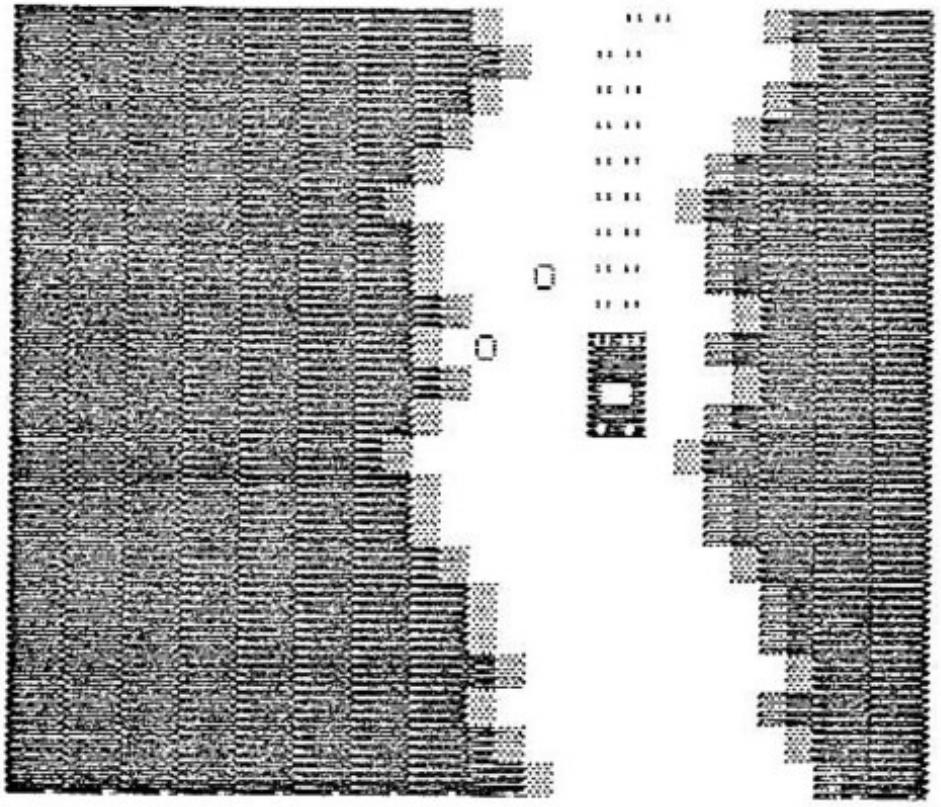
Compile-o e rode-o do modo usual.

```
*18836 3D 3D 3D 21 F6 49 35 589
 18843 FF 23 35 FF 23 36 FF 943
*18850 CD BB 02 11 F6 49 3E 792
 18857 00 CB 4C 20 01 12 13 349
*18864 CB 5C 20 01 12 13 CB 568
 18871 6C 20 01 12 13 C9 2A 421
*18878 0C 40 11 F8 02 19 ED 605
 18885 5B B1 49 EB 01 10 27 632
*18892 CD 4A 4A 01 E8 03 CD 794
 18899 4A 4A 01 64 00 CD 4A 528
*18906 4A 01 0A 00 CD 4A 4A 438
 18913 01 01 00 CD 4A 4A C9 556
*18920 97 3E 1C ED 42 38 03 603
 18927 3C 18 F9 09 12 13 C9 580
```

```
  10 GOTO 30
  20 PRINT "Digite 98 pontos"
  30 REM ▒▒INICIALIZACAO▒▒
  40 LET T=1000
  50 LET G=PEEK 16396+256*PEEK 1
6397
  60 LET O=G+694
  70 LET D=10
  80 LET H=128
  90 LET A=G+378
 100 LET K=0
 110 REM ▒▒MONTAGEM TELA▒▒
 120 CLS
 130 FOR L=1 TO 22
 140 PRINT "▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒
▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒";
 150 NEXT L
 160 PRINT AT 8,11;"▒▒▒▒▒▒▒▒▒▒";
 170 POKE 16437,255
 180 REM ▒▒ACAO CENTRAL▒▒
 190 POKE A,0
 200 POKE (A+1),0
 210 POKE (A-33),0
 220 POKE (A-32),0
 230 POKE (A-66),11
 240 POKE (A-65),11
 250 SCROLL
 260 LET L=USR 18937+USR 18975
 270 LET L=CODE INKEY$
 280 IF PEEK 18936=0 THEN LET A=
A-1
 290 IF PEEK 18935=0 THEN LET A=
A+1
 300 IF PEEK A>0 THEN GOTO 700
 310 IF PEEK (A+1)>0 THEN GOTO 7
00
 320 POKE A,155
```

```
 330 POKE (A+1),155
 340 POKE (A-33),7
 350 POKE (A-32),132
 360 POKE (A-66),139
 370 POKE (A-65),139
 380 IF PEEK 16437<=240 THEN GOT
O 630
 390 REM ▒▒▒IMPRESSAO PISTA▒▒▒
 400 IF K=890 THEN LET H=6
 410 FOR L=0 TO (O+31)
 420 POKE L,H
 430 NEXT L
 440 LET H=128
 450 POKE (O+D),136
 460 POKE (O+D+10),136
 470 FOR L=(O+D+1) TO (O+D+9)
 480 POKE L,0
 490 NEXT L
 500 IF RND>31000 THEN POKE (O+D
+1+RND/4600),52
 510 LET L=RND
 520 IF D>2 THEN IF L>20000 THEN
 LET D=D-1
 530 IF D<19 THEN IF L<14000 THE
N LET D=D+1
 540 REM ▒▒CONTROLE VELOCIDADE▒▒
 550 LET K=K+1
 560 LET T=T+50
 570 IF PEEK 18934=0 THEN LET T=
T-100
 580 IF T<0 THEN LET T=0
 590 IF T>1000 THEN LET T=1000
 600 FOR L=0 TO T
 610 NEXT L
 620 GOTO 190
 630 IF K<900 THEN GOTO 670
 640 POKE 16437,255
 650 LET K=K-900
 660 GOTO 400
 670 PRINT AT 13,10;"FIM DO TEMP
O"
 680 GOTO 800
 690 REM ▒▒▒EXPLOSAO▒▒▒
 700 POKE A,6
 710 POKE (A+1),129
 720 POKE (A-33),7
 730 POKE (A-32),135
 740 POKE (A-66),139
 750 POKE (A-65),139
 760 FOR N=1 TO 5000
 770 NEXT N
 780 LET T=1000
 790 GOTO 320
```

```
 800 STOP
 810 LET L=USR 18823
 820 IF INKEY$<>"9" THEN GOTO 82
0
 830 GOTO 810
```



00575

Incrementando o jogo:

Que tal colocar um relógio que conta o tempo no programa anterior? Isto não é difícil, pois basta "misturar" a listagem "Relógio" com a "Super Máquina" e compatibilizar as variáveis.

O primeiro passo é substituir a temporização de 1 minuto pela de 1 segundo. Depois coloca-se a rotina de relógio e cria-se as variáveis para minutos e segundos (as horas não serão contadas). Tudo isso está na listagem a seguir:

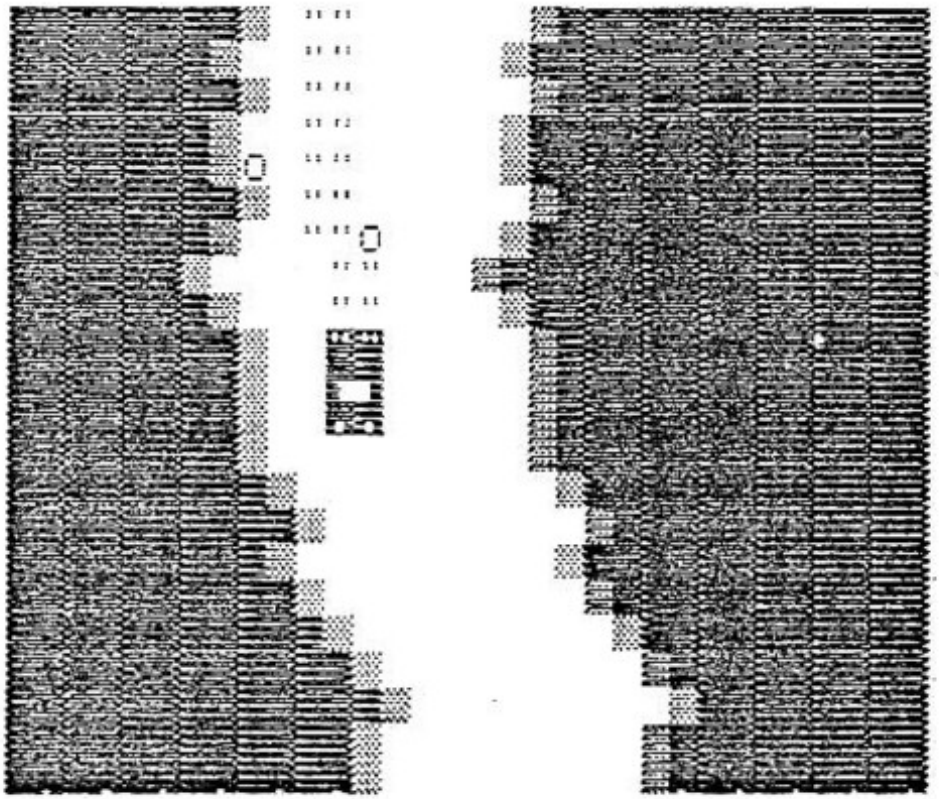
```
*18836 3D 3D 3D 21 F6 49 35 589
 18843 FF 23 36 FF 23 36 FF 943
*18850 CD 5B 02 11 F6 49 3E 792
```

```
 18857  00  CB  40  20  01  12  13  349
*18864  CB  5C  20  01  12  13  CB  568
 18871  6C  20  01  12  13  C9  2A  421
*18878  0C  40  11  F8  02  19  ED  605
 18885  56  51  49  EB  01  10  27  532
*18892  CD  4A  4A  01  E8  03  CD  794
 18899  4A  4A  01  64  00  CD  4A  528
*18906  4A  01  0A  00  CD  4A  4A  438
 18913  01  01  00  CD  4A  4A  C9  556
*18920  97  3E  1C  ED  42  38  03  503
 18927  3C  18  F9  09  12  13  C9  580
*18934  2A  0C  40  11  FF  02  19  417
 18941  EB  2A  B9  49  CD  3D  4A  875
*18948  3E  0E  12  13  2A  D1  49  437
 18955  CD  3D  4A  C9
```

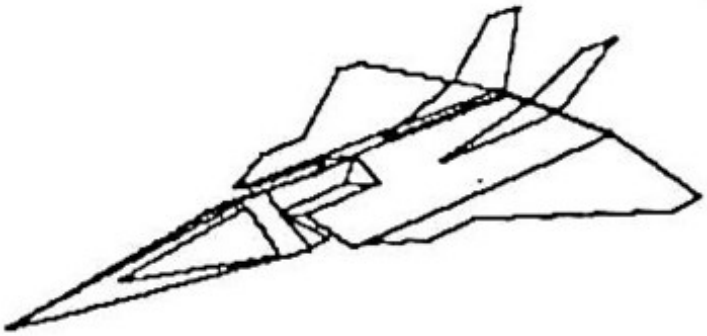
```
  10 GOTO 30
  20 PRINT "Digite 123 pontos"
  30 REM ▒▒▒INICIALIZACAO▒▒▒
  40 LET T=1000
  50 LET G=PEEK 16396+256*PEEK 1
6397
  60 LET O=G+694
  70 LET D=10
  80 LET H=128
  90 LET A=G+378
 100 LET M=0
 110 LET S=0
 120 LET Y=0
 130 LET K=0
 140 REM ▒▒▒MONTAGEM TELA▒▒▒
 150 CLS
 160 FOR L=1 TO 22
 170 PRINT "▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒
▒▒▒▒▒▒▒▒▒▒▒▒▒▒";
 180 NEXT L
 190 PRINT AT 8,12;"▒▒▒▒▒▒▒▒▒▒";
 200 POKE 16436,255
 210 REM ▒▒▒ACAO CENTRAL▒▒▒
 220 POKE A,0
 230 POKE (A+1),0
 240 POKE (A-33),0
 250 POKE (A-32),0
 260 POKE (A-66),11
 270 POKE (A-65),11
 280 SCROLL
 290 LET L=USR 18937+USR 18975
 300 LET L=CODE INKEY$
 310 IF PEEK 18936=0 THEN LET A=
A-1
 320 IF PEEK 18935=0 THEN LET A=
A+1
 330 IF PEEK A>0 THEN GOTO 760
```

```
 340 IF PEEK (A+1)>0 THEN GOTO 7
60
 350 POKE A,155
 360 POKE (A+1),155
 370 POKE (A-33),7
 380 POKE (A-32),132
 390 POKE (A-66),139
 400 POKE (A-65),139
 410 IF M>0 THEN IF S=0 THEN GOT
O 680
 420 REM ▒▒▒IMPRESSAO PISTA▒▒▒
 430 IF Y=890 THEN LET H=6
 440 FOR L=0 TO (O+31)
 450 POKE L,H
 460 NEXT L
 470 LET H=128
 480 POKE (O+D),136
 490 POKE (O+D+10),136
 500 FOR L=(O+D+1) TO (O+D+9)
 510 POKE L,0
 520 NEXT L
 530 IF RND>31000 THEN POKE (O+D
+1+RND/4600),52
 540 LET L=RND
 550 IF D>2 THEN IF L>20000 THEN
 LET D=D-1
 560 IF D<19 THEN IF L<14000 THE
N LET D=D+1
 570 REM ▒▒CONTROLE VELOCIDADE▒▒
 580 LET Y=Y+1
 590 LET K=K+1
 600 LET T=T+50
 610 IF PEEK 18934=0 THEN LET T=
T-100
 620 IF T<0 THEN LET T=0
 630 IF T>1000 THEN LET T=1000
 640 GOSUB 870
 650 FOR L=0 TO T
 660 NEXT L
 670 GOTO 220
 680 IF C=1 THEN GOTO 430
 690 IF Y<900 THEN GOTO 730
 700 LET Y=Y-900
 710 LET C=1
 720 GOTO 430
 730 PRINT AT 13,10;"FIM DO TEMP
O"
 740 GOTO 960
 750 REM ▒▒▒EXPLOSAO▒▒▒
 760 POKE A,5
 770 POKE (A+1),129
 780 POKE (A-33),7
 790 POKE (A-32),135
```

```
 800 POKE (A-66),139
 810 POKE (A-65),139
 820 FOR N=1 TO 500
 830 GOSUB 870
 840 NEXT N
 850 LET T=1000
 860 GOTO 350
 870 LET L=USR 19032
 880 IF PEEK 16436>195 THEN RETU
RN
 890 POKE 16436,255
 900 LET S=S+1
 910 IF S<60 THEN RETURN
 920 LET S=0
 930 LET M=M+1
 940 LET C=0
 950 RETURN
 960 STOP
1000 SLOW
1010 LET L=USR 18823
1020 IF INKEY$<>"9" THEN GOTO 10
20
1030 GOTO 1000
```



```
00209  00:17
```



CAPÍTULO 9

EFEITOS VISUAIS EM SEUS JOGOS

Uma das partes mais importantes de um jogo é, sem dúvida, a parte estética ou visual. O sucesso ou fracasso de um jogo depende de muitos fatores, mas um deles está intimamente ligado ao senso crítico geral: a apresentação.

Todos sabem que um jogo formado por elementos de imagem ricos e efeitos visuais mirabolantes é muito mais emocionante que o clássico "feijão com arroz" das imagens pobres e sem criatividade. Lembre-se de que a nossa intenção é a de explorar ao máximo os recursos disponíveis, por isso, a parte de efeitos visuais ganhou um capítulo exclusivo.

Nós vamos tratar de vários tipos de efeitos, alguns com puro espírito estético, outros mais voltados para uma necessidade mais específica. De um modo geral, são rotinas em Assembly escritas para manipular o vídeo, obtendo o efeito desejado.

Usos práticos:

Apesar de não parecer, os efeitos visuais têm uma importância muito grande para a programação de jogos. Os "scrolls", por exemplo, servem para dar uma impressão de movimento de fundo, como se pode ver no "Intruder" ou nos dois "Super Máquina". O único problema dos "scrolls" é que nem o

BASIC compilado tem velocidade suficiente para executá-los satisfatoriamente. Deste modo, optou-se por fazê-los em Assembly, e suas listagens estão impressas a seguir (todos os quatro scrolls são auto-relocáveis, isto é, podem ser colocados em qualquer lugar sem sofrer alterações.

```
 Scroll para cima:
2A  0C  40  E5  11  21  00  19  D1  01  D6
02  ED  B0  C9

   Scroll para baixo:
2A  0C  40  01  B4  02  09  11  21  00  E5
19  D1  EB  ED  B8  AF  06  20  23  77  10
FC  C9

   Scroll para a direita:
2A  0C  40  01  1F  00  3E  16  C5  09  E5
D1  13  ED  B8  23  70  0E  20  09  3D  C1
C8  18  EF


   Scroll para a esquerda:
2A  0C  40  3E  16  01  1F  00  23  E5  D1
23  C5  ED  B0  2B  70  C1  3D  C8  23  18
F1
```

Você pode também combinar dois scrolls "perpendiculares" para obter movimentação diagonal da tela.

Operando os apontadores de imagem:

Para quem não sabe, o arquivo de imagem é mapeado em memória, e o seu endereço inicial é indicado pelos bytes 16396 e 16397 (variável D-FILE). O funcionamento básico da tela do vídeo é explicado com detalhes no final do livro, no Apêndice II.

Um efeito muito interessante pode ser obtido se mudarmos convenientemente o conteúdo de D-FILE. Quando isto é feito, o software que envia os sinais para a televisão começa a "colher" os dados em outro lugar. Deste modo, podemos definir várias telas ao longo da memória e chamá-las uma a uma, bastando para isto seguir a fórmula:

```
10 LET A=E-256*INT (E/256)
20 LET B=INT (E/256)
30 POKE 16396,A
40 POKE 16397,B
```

onde E é o primeiro endereço da tela.

É importante memorizar os valores de D-FILE para os quais a tela funciona normalmente, no contrário você não conseguirá retornar para o vídeo original.

Outro aspecto importante é a formatação da tela. Ela deve seguir rigorosamente a forma descrita no Apêndice II, e o endereço E deve marcar exatamente a posição do primeiro NEWLINE da tela. Se uma destas regras for desrespeitada, há uma boa chance do computador sofrer um colapso total, com a conseqüente destruição do programa.

Para ter uma idéia melhor, digite o programa que é listado a seguir (não é necessário compilar).

```
 10 LET A=PEEK 16396
 20 LET B=PEEK 16397
 30 LET C=0
 40 LET D=100
 50 LET E=A+256*B
 60 LET F=C+256*D
 70 FOR L=0 TO 793
 80 POKE (L+F),PEEK (L+E)
 90 NEXT L
100 SLOW
110 FOR L=0 TO 21
120 PRINT "**********************
**********";
130 NEXT L
140 POKE 16397,D
150 POKE 16396,C
160 POKE 16396,A
170 POKE 16397,B
180 GOTO 140
```

Arquivos de tela Intermediários:

Em jogos que envolvem a movimentação simultânea de muitos elementos na tela, podemos notar uma certa defasagem de tempo entre eles. Percebemos que as movimentações não são tão "simultâneas" quanto pareceriam em jogos mais simples, pois a ação do jogo é dividida entre o processamento de cada parte isoladamente, e é isso que causa a defasagem.

Para devolvermos a desejada simultaneidade ao jogo, temos que imprimir todos os elementos ao mesmo tempo, em vez de controlá-los e imprimi-los um a um. Isto pode ser feito com a mudança da seqüência das linhas de programação, mas é quase impraticável em jogos longos e estruturados. Nós encontramos dificuldades desse tipo em um jogo que movia um helicóptero,

cinco balas, duas bombas, um cenário de fundo, um placar e vários inimigos. Era simplesmente impossível sincronizar a impressão de tantas coisas ao mesmo tempo!

Felizmente, nós temos uma solução alternativa que quase não altera a lógica de funcionamento do programa. Constitui-se de um "rascunho" da tela onde as imagens são atualizadas uma a uma, e ao final do ciclo, este "rascunho" é copiado em cima da tela definitiva. Desta maneira, visualizamos toda a ação acontecer ao mesmo tempo, sem nos darmos conta de que cada movimento é executado em um instante.

Outra aplicação deste sistema é para o caso de desenhos que "piscam" demais, como o carro de "Super Máquina", ou a nave de "Intruder". Basta copiar a tela-rascunho para a definitiva no instante em que os mesmos estão acesos. É isto que muitos jogos comerciais, como o excelente "Defensor 3D" e o "Redalert" fazem para melhorar o visual.

A desvantagem deste sistema é a perda de velocidade (a cada ciclo deve-se copiar a tela) e o gasto adicional de memória (uma tela a mais).

Como trabalhar com o "Buffer":

O "Buffer" (é assim que chamamos a tela-rascunho) nada mais é que uma cópia fiel do arquivo de imagem original. Assim, basta proceder com ele normalmente, como se fosse a tela do D-FILE, e quando toda a imagem estiver pronta nele, deve-se copiá-lo sobre a tela normal.

Este trabalho exige uma velocidade muito elevada, e por isto resolvemos deixá-lo a cargo de uma rotina em Assembly. Tal rotina é tão simples, que os conhecedores da linguagem Assembler compreenderão imediatamente. Sua listagem é dada a seguir:

```
Buffer Tela->  Ld HL,31800      21007C
               Ld DE,(16396)    ED5B0C40
               Ld BC,793        011903
               LDIR             EDB0
               RET              C9
```

Ela copia um bloco de 793 bytes iniciado em 31ØØ em cima da tela de imagem. O endereço 31ØØ é o início do "buffer" — não foi escolhido por acaso. Um endereço mais baixo significaria maior ocupação da memória, e um endereço mais alto fatalmente levaria o "buffer" a invadir a área do stack.

Note que o seu programa não pode ser maior que 15,2 bytes, pois se isto acontecer, as últimas posições de memória ficarão conflitantes com o

"buffer", e isto causará danos na área do BASIC. Felizmente, são poucos os programas que superam os 12 Kbytes.

Outro ponto importante a ser frisado é que o "buffer" deve estar corretamente formatado, como já foi visto neste capítulo. Para fazer isso, foi criada uma rotina oposta à anterior, que copia a tela em cima do "buffer", deixando-o formatado. Aqui está ela:

```
Tela Buffer->  Ld HL,(16396)    2A0C40
               Ld DE,31800      11007C
               Ld BC,793        011903
               LDIR             EDB0
               RET              C9
```

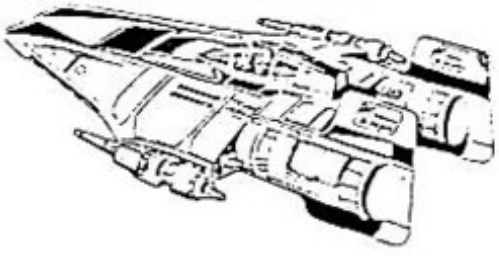
Ela SEMPRE deve ser chamada no início de cada execução para "colocar o buffer em ordem". Você pode inicializar o "buffer" com qualquer conteúdo, imprimindo este conteúdo na tela normal, e chamando a rotina de cópia. Se desejar um "buffer" em branco, execute a rotina após um CLS.

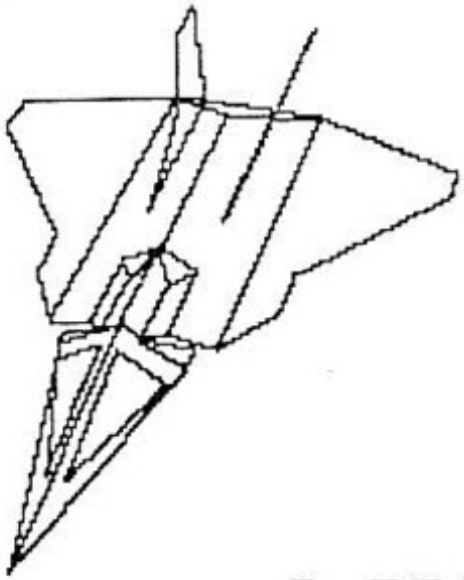
O próximo passo constitui-se em desviar os locais de destino dos POKEs. Lembre-se de que todos os POKEs devem agora ser feitos no "buffer", e não mais na tela, devendo-se então mudar a forma de referenciar o endereçamento. Nos nossos jogos, basta trocar:
LET A = PEEK 16396 + 256 * PEEK 16397 por:
LET A = 31800

Os "scrolls" também devem ser mudados. Por exemplo: LD HL,(16396) deve ser mudado para: LD HL,31800.

Experimente combinar esta técnica com o "Intruder" ou o "Super Máquina". Veja que neste último, o "scroll" para cima deverá ser alterado para movimentar o "buffer".

Embora o efeito visual melhore muito com o recurso do "buffer", devemos admitir que isto implica em um sacrifício da velocidade de processamento. Por isso, quando for escrever um jogo, considere este fato e pese a importância da velocidade. Não se esqueça também de retirar todos os "loops" de espera e racionalizar o processamento para ganhar tempo, pois a conciliação não é impossível de ser alcançada.





CAPÍTULO 10

A MATEMÁTICA NO COMPILADOR

A principal limitação do compilador é a ausência de números fracionários. Os jogos em geral utilizam apenas números inteiros para contar pontos, recordes, naves disponíveis, posições, etc., mas existem casos particulares em que eles não são suficientes.

Um caso típico de rotina que usa números fracionários é a plotagem de segmentos de reta oblíquos, que se pode encontrar em jogos como o "Missile Command" (Atari), "Cosmic Fighter" e "Galaxy Invasion" (ambos para o TRS-80). E, para os matemáticos, mostraremos mais adiante como compilar seno e cosseno, com suas devidas limitações.

Para que tudo isto seja possível, tornar-se necessária uma técnica que seja capaz de simular números decimais a partir da ferramenta de que dispomos: os números inteiros.

Multiplicando a escala:

Multiplicar a escala nada mais é do que "expandir" o intervalo matemático ao qual está restrito o número que estamos operando. Um exemplo prático disto é a função RND. Normalmente, o RND situa-se na faixa

entre Ø e 1 (não incluído), mas o compilador o coloca no intervalo que vai de Ø a 32767 (incluído). Este é o único meio que nos possibilita ter um elevado número de RNDs diferentes, já que todos são inteiros.

Demultiplicando a escala:

A demultiplicação da escala é a operação inversa, e tem por objetivo reduzir o intervalo expandido para o que desejamos. É o caso do RND, que é dividido por uma constante para seu intervalo ser "rebaixado" à faixa certa, como se pode notar em jogos mostrados anteriormente.

Como usar:

Bem, após tantos conceitos teóricos, vamos a um exemplo prático. Suponha que desejemos traçar uma reta oblíqua de coeficiente angular de valor 0,4 e que passa pelo ponto (0,0) da tela. Em BASIC normal, o programa seria o seguinte: LISTAGEM 1

```
10 FOR X=0 TO 63
20 LET Y=2/5*X
30 PLOT X,Y
40 NEXT X
50 STOP
```

Este programinha funciona perfeitamente com o BASIC normal, mas, paradoxalmente, traça uma reta horizontal quando é compilado. A causa do erro é o modo de se calcular a ordenada Y. Lembre-se que o compilador só trabalha com números inteiros, logo a razão 2/5 é interpretada como sendo 0 e não como 0,4. Por isso a reta saiu errada.

Para tornar o programa compatível com o sistema do compilador, faça as seguintes alterações:

a) Para multiplicar a escala, altere a linha 20 para:

```
20 LET Y=10*2/5*X
```

Ou, para ser mais prático,

```
20 LET Y=4*X
```

b) Para demultiplicar a escala, altere a linha 30 para:

```
30 PLOT X,(Y/10)
```

Tente agora usar o programa na forma compilada. Sucesso! Ficou igual ao BASIC comum! O segredo consiste em deixar os arredondamentos para o final, pois assim seu efeito fica minimizado.

Aplicação prática:

A rotina que mostraremos a seguir tem como função unir dois pontos dados de coordenadas (A,B) e (C,D) com um segmento de reta. Ela pode ser muito útil para desenhar, ou mesmo para animar jogos que necessitem deste recurso.

Seu funcionamento é baseado na equação da reta $y - yo = m(x - xo)$, onde m é o coeficiente angular da reta, e (xo, yo) é um ponto pertencente à reta. Por trabalharmos apenas com segmentos de reta, adotamos tais coordenadas como sendo a extremidade inicial deste segmento. O que temos a fazer é calcular o coeficiente angular através da própria equação da reta, na qual (xo, yo) assume o valor de (A,B) e (x,y) assume o valor de (C,D). Depois, variamos x de A até C, calculamos y e plotamos o ponto (x,y).

Na prática, entretanto, notamos que para valores de m maiores que 1 os pontos começam a se distanciar demais entre si.. E o que é pior, não dá para desenhar retas verticais, pois neste caso m tende a infinito, e é impossível fazer cálculos nesta base. E agora?

Que nos perdoem os matemáticos, mas o jeito é fazer a "barbaridade" de trocar os x pelos y. É isto mesmo! Quando fazemos isso, tudo o que é horizontal passa a ser vertical e vice-versa. Deste modo, a rotina que traçava tão bem retas tendendo à horizontal agora só traça retas que tendem para a vertical. Para desenhar retas em todas as direções é necessário então utilizar as duas rotinas seletivamente, cada uma para a sua faixa de aceitação.

O programa que faz tudo isso está listado a seguir. Após compilá-lo, execute-o com GOTO 3000 e entre com as coordenadas A, B, C e D respectivamente. O seu segmento (A,B) (C, D) será desenhado.

```
10 GOTO 30
20 PRINT "XXXX"
30 LET A=PEEK 18934
40 LET B=PEEK 18935
50 LET C=PEEK 18936
60 LET D=PEEK 18937
```

```
 100 LET E=C-H
 110 LET F=D-B
 120 IF (ABS E)>(ABS F) THEN GOT
O 200
 130 REM ▒▒▒DELTAY>DELTAX▒▒▒▒▒▒
 140 IF B>D THEN GOSUB 1000
 150 FOR Y=B TO D
 160 LET X=100*E/F*(Y-B)/100
 170 PLOT (A+X),Y
 180 NEXT Y
 190 GOTO 2000
 200 REM ▒▒▒DELTAX>DELTAY▒▒▒▒▒▒
 210 IF A>C THEN GOSUB 1000
 220 FOR X=A TO C
 230 LET Y=100*F/E*(X-A)/100
 240 PLOT X,(B+Y)
 250 NEXT X
 260 GOTO 2000
1000 LET X=A
1010 LET A=C
1020 LET C=X
1030 LET X=B
1040 LET B=D
1050 LET D=X
1060 RETURN
2000 STOP
3000 INPUT A
3010 INPUT B
3020 INPUT C
3030 INPUT D
3040 POKE 16934,A
3050 POKE 16935,B
3060 POKE 16936,C
3070 POKE 16937,D
3080 LET L=USR 16823
3090 GOTO 3000
```

Compilando seno e cosseno:

Os leitores que se interessam pelos recursos gráficos do micro (que aliás não são grande coisa) vão se interessar por este método de compilar as funções seno e cosseno, pois ficam muito mais rápidas que as originais e possibilitam animações interessantes.

A diferença principal está no intervalo matemático (—128 a 127). Cada seno está associado a 2 graus consecutivos, logo, guardamos os senos de 360 graus em 180 bytes. Além disto, foi estabelecida a seguinte convenção:

byte maior que 127 . . . senos negativos
byte menor que 128 . ...senos positivos

Passemos à listagem 2. Com o compilador na memória, digita-se o comando GOTO 120. Quando o micro terminar, a linha 20 conterá diversos caracteres entre as aspas. Se isto não acontecer, verifique se não sobrou nenhuma linha 2 de alguma eventual compilação anterior. Estando tudo OK, compile o programa, rode-o, e vamos lá...

```
 10 GOTO 30
 20 PRINT "Digite 180 pontos"
 30 REM ▒▒▒▒TRAÇADOR SENOIDAL▒▒▒▒
 40 FOR L=1 TO 6
 50 FOR N=0 TO 60
 60 LET Y=PEEK (16529+N*L/2)
 70 IF Y>126 THEN LET Y=Y-256
 80 PLOT N,(22+Y/6)
 90 NEXT N
100 NEXT L
110 STOP
120 REM ▒▒▒▒GERADOR DE TABELA▒▒▒▒
130 FOR N=0 TO 179
140 LET Y=INT (127*SIN (N/90*PI
))
150 IF Y=11 THEN LET Y=12
160 POKE N+18836,Y
170 NEXT N
```

Algumas considerações sobre o programa:

1) Para se obter o seno de um ângulo x, use a fórmula:
sen x = PEEK (18934 + x/2), sem acerto de escala.

2) Para se obter o cosseno de x, faça y = 90 - x e use a fórmula:
cos x = PEEK (18934 + y/2)

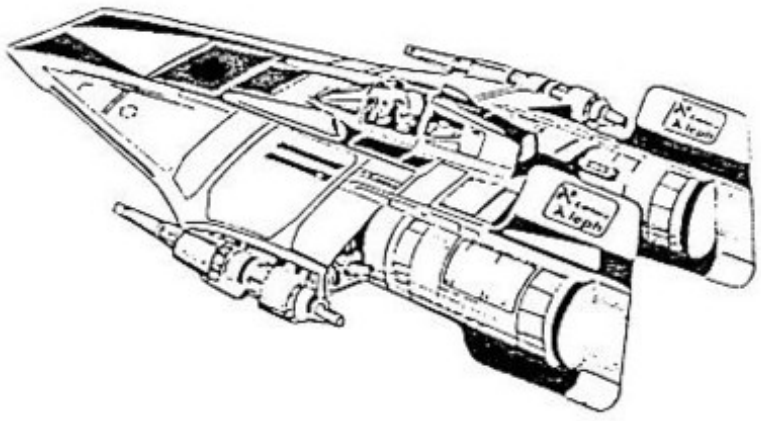
3) Tanto x como y devem estar no intervalo de 0 a 360 graus (primeira determinação positiva).

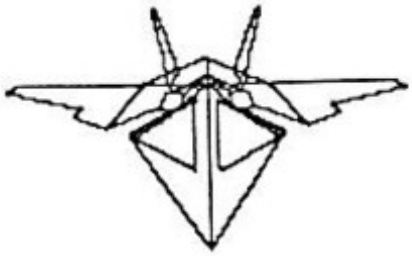
Estando tudo entendido, vejamos a listagem 3. Ela traça uma curva fechada conforme o sistema:

x = 31 + 20 . cos (n + k)
y = 21 + 20 . sen n, onde n é o parâmetro de 0 a 360 graus
k é a defasagem entre os parâmetros em x e y

Na medida em que k varia, a curva muda de circunferência para elipse e depois a reta, para depois regredir. Redigite o BASIC da listagem 3 sobre o da listagem 2, recomplile o programa, e veja com que velocidade isto tudo se processa. . .

```
 10 GOTO 30
 20 PRINT "Digite 180 pontos"
 30 REM ▒▒▒GRAFICO CIRCUNF▒▒▒
 40 FOR K=0 TO 88
 50 FOR N=1 TO 180
 60 LET Y=PEEK (18934+N)
 70 LET A=PEEK (18933+N)
 80 IF Y>128 THEN LET Y=Y-256
 90 IF A>128 THEN LET A=A-256
100 LET X=(18979-N+K)
110 LET B=(18978-N+K)
120 IF X<18934 THEN LET X=X+180
130 IF B<18934 THEN LET B=B+180
140 LET X=PEEK X
150 LET B=PEEK B
160 IF X>128 THEN LET X=X-256
170 IF B>128 THEN LET B=B-256
180 UNPLOT (32+B/6),(22+A/6)
190 PLOT (32+X/6),(22+Y/6)
200 NEXT N
210 LET K=K+1
220 NEXT K
230 STOP
240 REM ▒▒▒MONTADOR TABELA▒▒▒
250 FOR N=0 TO 179
260 LET Y=INT (127*SIN (N/90*PI
))
270 IF Y=11 THEN LET Y=12
280 POKE N+18836,Y
290 NEXT N
```





CAPÍTULO 11

INTELIGÊNCIA ARTIFICIAL

Este é um capítulo muito importante e muito difícil, portanto recomendamos uma leitura atenta e a digitação cuidadosa dos programas.

Importante, porque dele depende todo o resultado de um jogo. Um jogo tipo PAC-MAN perde todo o seu interesse se os seus opositores movimentam-se como idiotas.

Difícil, porque ensinar um computador a pensar é o mesmo que ensinar uma porta a fazer algo mais do que abrir e fechar.

TRON

Este jogo, baseado numa seqüência de cenas do filme homônimo, certamente irá proporcionar as mesmas emoções de quem o viu.

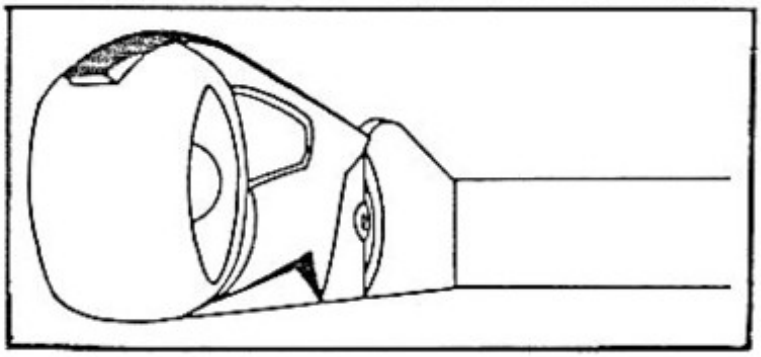
Para quem não viu o filme, aqui vai uma breve explicação do mesmo:

O filme passava-se dentro de um computador onde os programas assumiam a forma física de seu usuário. Porém, todos eles eram dominados por um programa central, chamado MCP. O MCP colocava vários programas inúteis para se destruírem jogando entre si. Um dos jogos era chamado de "Light Cycles", ou seja, "Motos de Luz".

O design do jogo era o seguinte: motos futurísticas (vide desenho) encerradas dentro de uma área livre, chamada de "grade de jogo", que deixavam atrás de si rastros luminosos, que formavam uma espécie de parede sólida. Era com estes rastros que se deveria cercar seu inimigo, pois todas as motos não podiam bater em nada

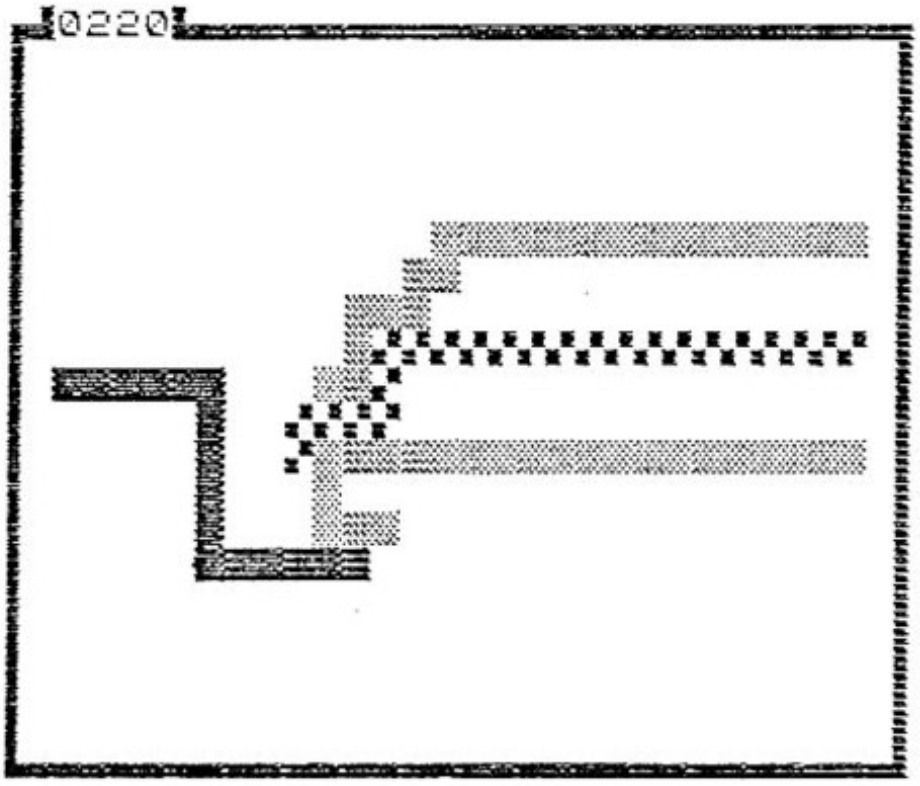
Baseado nesta idéia foi criado este jogo, em que, se a resolução gráfica não ajuda muito, a ação compensa esta deficiência.

Sua moto é a preta, e você competirá contra outras 3 motos(duas motos cinza e uma xadrez). Suas únicas armas são astúcia, habilidade e um acelerador, porém, lembre-se: as motos nunca batem se houver um caminho a seguir. No *"Soluções técnicas"* será comentado como foi feita a rotina de perseguição do TRON.



Seus controles são:

tecla 5 — esquerda
tecla 6 — desce
tecla 7 - sobe
tecla 8 — direita
tecla Ø - acelera



Digitação

Primeiramente, com o compilador e o monitor na memória, digite a linha 1Ø GOTO 3Ø e a linha 2Ø PRINT com os caracteres necessários, e siga

normalmente com a listagem HEXADECIMAL a seguir. Estas rotinas têm por função ler o JOYSTICK, imprimir o placar, e apagar o rastro das motos que batem.

```
#18836 00 00 00 00 00 21 F6 279
 18843 49 E5 06 05 36 FF 23 657
#18850 10 FB CD BB 02 E5 D1 1099
 18857 E1 CB 4A 20 02 36 00 590
#18864 23 CB 5A 20 02 36 00 416
 18871 23 CB 62 20 02 36 00 424
#18878 23 CB 5B 20 02 36 00 417
 18885 23 28 06 CB 6A 20 02 424
#18892 36 00 CB 5B 20 12 CB 601
 18899 6A 20 0E CB 63 20 0A 496
#18906 CB 4A 28 06 CB 62 28 664
 18913 02 36 00 00 ED 5B 0C 396
#18920 40 13 13 13 2A D1 49 445
 18927 01 E8 03 CD 66 4A 01 618
#18934 64 00 CD 66 4A 01 0A 492
 18941 00 CD 66 4A 01 01 00 383
#18948 97 3E 1C ED 42 38 03 603
 18955 3C 18 F9 09 12 13 C9 580
#18962 2A 0C 40 ED 56 10 40 526
 18969 23 7E FE 00 20 02 36 503
#18976 00 E5 97 ED 52 E1 C8 1124
 18983 18 F0
```

Após esta digitação prossiga normalmente com a listagem em BASIC.

```
MAPA DE VARIAVEIS TRON

A-POSICAO DA SUA MOTO
B-VETOR DESLOCAMENTO DA SUA MOTO
C-LINHA DA SUA MOTO
D-COLUNA DA SUA MOTO
E-AUXILIAR
F-ENDERECO DO D-FILE
G-LINHA DAS MOTOS INIMIGAS
H-COLUNA DAS MOTOS INIMIGAS
I-CARACTERE DA MOTO INIMIGA
J-VELOCIDADE DO JOGO
K-FLIP-FLOP DA SUA MOTO
L-CHAMADA USR E TECLAS
M-STATUS DO SEU ACELERADOR
N-
O-
P-
R-
S-
T-
```

```
U-
V-
W-
X-CONTROLADOR INDEXADOR E LOOP
Z(1,2,3)-POSICAO DAS MOTOS
         INIMIGAS
Z(4,5,6)-VETOR DAS MOTOS INIMI-
         GAS
Z(7,8,9)-BATIDA DAS MOTOS INIMI-
         GAS


  10 GOTO 30
  20 PRINT "Digite 149 pontos"
  30 GOSUB 740
  40 REM ▒▒▒▒LEITURA TECLAS▒▒▒▒
  50 LET L=USR 18939
  60 PAUSE 1
  70 LET S=S+10
  80 IF PEEK 18937=0 THEN IF B<>
3 THEN LET B=0
  90 IF PEEK 18938=0 THEN IF B<>
2 THEN LET B=1
 100 IF PEEK 18936=0 THEN IF B<>
1 THEN LET B=2
 110 IF PEEK 18935=0 THEN IF B<>
0 THEN LET B=3
 120 IF PEEK 18934=0 THEN LET M=
0
 130 IF K=1 THEN IF M=1 THEN GOT
O 220
 140 IF Z(7)=1 THEN IF Z(8)=1 TH
EN IF Z(9)=1 THEN GOTO 1000
 150 REM ▒▒▒▒CONTROLE MOTO▒▒▒▒
 160 IF B=0 THEN LET A=A-1
 170 IF B=3 THEN LET A=A+1
 180 IF B=1 THEN LET A=A+33
 190 IF B=2 THEN LET A=A-33
 200 IF PEEK A<>0 THEN GOTO 630
 210 POKE A,128
 220 LET E=A-(F+1)
 230 LET C=INT (E/33)
 240 LET D=INT (E-(C*33))
 250 LET M=1
 260 LET K=-K
 270 REM ▒▒▒▒MOTOS INIMIGAS▒▒▒▒
 280 FOR X=1 TO 3
 290 IF X=1 THEN LET I=6
 300 IF X=2 THEN LET I=8
 310 IF X=3 THEN LET I=136
 320 IF Z(6+X)=1 THEN GOTO 580
 330 LET E=Z(X)-(F+1)
 340 LET G=INT (E/33)
```

```
 350 LET H=INT (E-(G*33))
 360 LET Z(3+X)=4
 370 REM ROTINA INTELIGENCIA
 380 IF ABS (C-G)>ABS (D-H) THEN
 GOTO 420
 390 IF PEEK (Z(X)+1)=0 THEN IF
D>H THEN LET Z(3+X)=3
 400 IF PEEK (Z(X)-1)=0 THEN IF
D<H THEN LET Z(3+X)=0
 410 GOTO 440
 420 IF PEEK (Z(X)+33)=0 THEN IF
 C>G THEN LET Z(3+X)=1
 430 IF PEEK (Z(X)-33)=0 THEN IF
 C<G THEN LET Z(3+X)=2
 440 IF Z(3+X)<>4 THEN GOTO 510
 450 LET Z(3+X)=0
 460 IF PEEK (Z(X)-33)=0 THEN LE
T Z(3+X)=2
 470 IF PEEK (Z(X)+33)=0 THEN LE
T Z(3+X)=1
 480 IF PEEK (Z(X)+1)=0 THEN LET
 Z(3+X)=3
 490 IF PEEK (Z(X)-1)=0 THEN LET
 Z(3+X)=0
 500 REM CONTROLE MOTOS
 510 IF Z(3+X)=0 THEN LET Z(X)=Z
(X)-1
 520 IF Z(3+X)=3 THEN LET Z(X)=Z
(X)+1
 530 IF Z(3+X)=1 THEN LET Z(X)=Z
(X)+33
 540 IF Z(3+X)=2 THEN LET Z(X)=Z
(X)-33
 550 IF PEEK Z(X)<>0 THEN LET Z(
6+X)=1
 560 POKE Z(X),I
 570 IF Z(6+X)=1 THEN GOSUB 680
 580 NEXT X
 590 FOR X=0 TO J
 600 NEXT X
 610 GOTO 50
 620 REM SUB-EXPLOSAO
 630 POKE A,23
 640 POKE 19070,128
 650 LET L=USR 19060
 660 GOTO 1000
 670 REM EXPLOSAO-INIMIGA
 680 POKE 19070,I
 690 LET L=USR 19060
 700 POKE Z(X),23
 710 LET J=J+75
 720 RETURN
 730 REM INICIALIZACAO
```

```
 740 LET F=PEEK 16396+256*PEEK 1
6397
 750 LET A=F+332
 760 LET B=3
 770 LET J=0
 780 LET S=0
 800 LET R=INT (RND/16383)
 810 LET Z(2)=(F+229+33*R)
 820 LET Z(1)=(F+328+33*R)
 830 LET Z(3)=(F+427+33*R)
 840 LET M=1
 850 LET K=1
 860 FOR X=4 TO 9
 870 LET Z(X)=0
 880 NEXT X
 890 REM ▒▒MONTAGEM TELA▒▒
 900 FOR X=1 TO 62
 910 PLOT (63-X),1
 920 PLOT X,42
 930 NEXT X
 940 FOR X=1 TO 42
 950 PLOT 62,(43-X)
 960 PLOT 1,X
 970 NEXT X
 980 PRINT AT 0,0;" ▟0000▙";
 990 RETURN
1000 STOP
1010 LET L=USR 18823
1020 IF INKEY$<>"0" THEN GOTO 10
20
1030 CLS
1040 GOTO 1010
```

Terminada a digitação, grave e rode o programa. Se tudo estiver correto, boa sorte com suas fechadas.

Soluções Técnicas — TRON

Neste programa, realmente, há muitas novidades. Talvez a maior, mas que será explanada melhor no próximo capítulo, é o uso da variável Z, a única dimensionada, permitida na forma indexada. Note que as três motos inimigas realizam o mesmo procedimento, e que foi usada a mesma rotina para as três. Apenas foi mudado o indexador (variável X), para apontar para uma variável Z (. . .) diferente.

Outra novidade é o "FLIP-FLOP", a variável K. A cada ciclo do jogo, a variável assume um valor. Este valor em conjunto com o status do acelerador, controla o efeito de meia-velocidade da sua moto.

Voltando ao assunto principal deste capítulo, explicaremos a rotina de

perseguição usada no TRON. Ela se baseia em aproximação progressiva segundo os eixos X (linha) e Y (coluna) do sistema cartesiano.

A rotina divide-se em duas partes. A primeira é a rotina que faz a aproximação das motos inimigas à sua moto, escolhendo o melhor caminho. A seguir, vem a rotina que acha (caso a primeira rotina não consiga movimentar a moto) uma direção a seguir, se houver.

Já que a localização das motos é feita por endereços, é necessário fazer a conversão para linha e coluna. Isto é facilmente conseguido com três linhas de programa.

```
LET AUX=(D-FILE+1)-(POS. DESEJAD
A)
LET LINHA=(AUX/33)
LET COLUNA=AUX-(LINHA*33)
```

Com isso você tem a comodidade da localização do PRINT AT e a rapidez do POKE.

Burgertime

Como o próprio nome já diz, o jogo tem algo a ver com hamburgers.

Você é um simples esfomeado, louco para devorar um hamburger e acaba decidindo fazer o seu. Porém, certas pessoas não gostam dessa sua idéia; eles são os Mac Monsters (ou Bob Monsters ou Jack Monsters) e não deixarão você realizar seu intento.

Os ingredientes estão espalhados pelo labirinto e o hamburger cresce à medida que você os coleta. Quando você achar que o hamburger está devidamente recheado, procure na parte de baixo do labirinto o pão que fecha seu almoço. Após isso, ele abaixará e descarregará no placar o bônus acumulado. O multiplicador de bônus aumenta consideravelmente os pontos obtidos e pode ser incrementado coletando-se os "$" do labirinto.

Se você estiver sendo perseguido, pode transportar-se para o outro lado do labirinto através das portas  e .

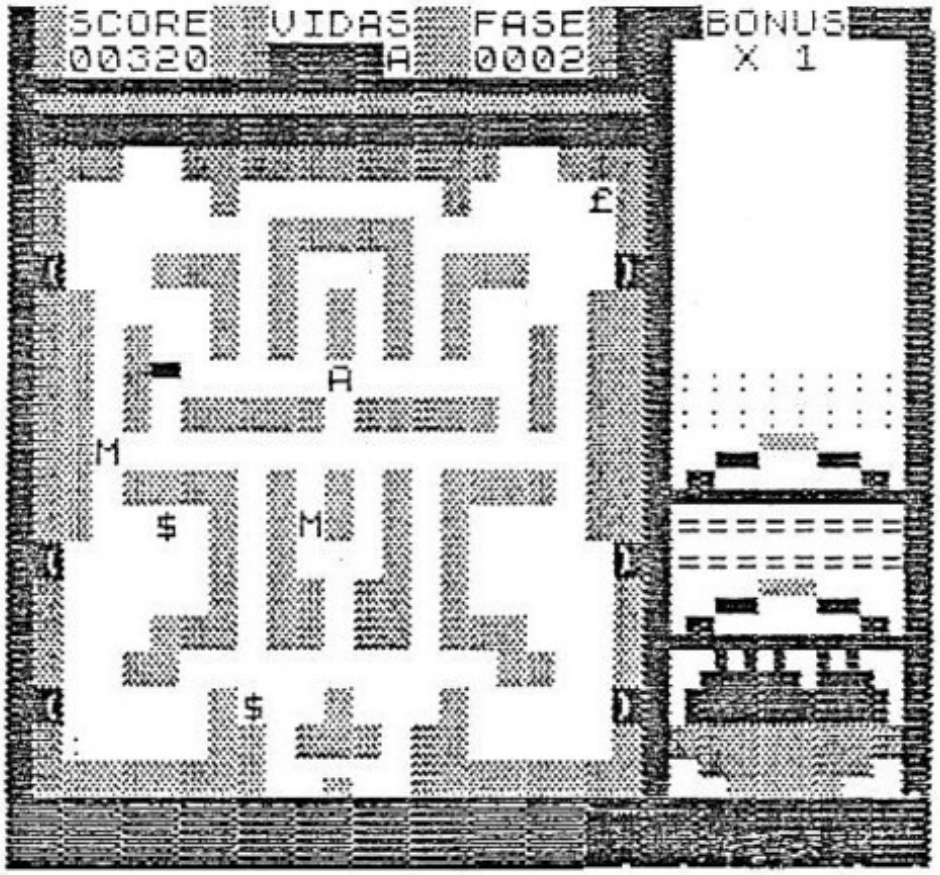
Há mais um personagem nesta estória: é o "chato" comilão, que come tudo que encontra. Você deve o mais rapidamente possível, pegá-lo e anulá-lo, pois se ele conseguir chegar ao fundo do labirinto e comer o pão que fecha o hamburger, sua única solução é o suicídio.

Seus comandos são:

5 – esquerda

6 – desce

7 — sobe
8 - direita



Digitação

Inicie por colocar a listagem HEXADECIMAL a seguir. É uma rotina pequena e controla apenas o placar. Achamos desnecessária a rotina JOY 3, pois nesse jogo não é necessário o acionamento de duas ou mais teclas simultaneamente. Digite antes as linhas 10 GOTO 30 e 20 PRINT quantos caracteres forem necessários.

CUIDADO!!! Neste programa delete as linhas do monitor antes de compilar.

```
*18836 ED 5B 0C 40 06 24 13 465
 18843 10 FD 2A D1 49 CD 1C 826
*18850 4A 06 0A 13 10 FD 2A 420
 18857 D5 49 CD 1C 4A 06 07 606
*18864 13 10 FD 2A D9 49 CD 825
 18871 2E 4A C9 01 E8 03 CD 762
*18878 35 4A 01 64 00 CD 35 486
 18885 4A 01 0A 00 CD 35 4A 417
*18892 01 01 00 CD 35 4A C9 535
 18899 97 3E 1C ED 42 38 03 603
*18906 3C 18 F9 09 12 13 C9 580
```

Digite normalmente a listagem BASIC e após terminar grave; e bom apetite!

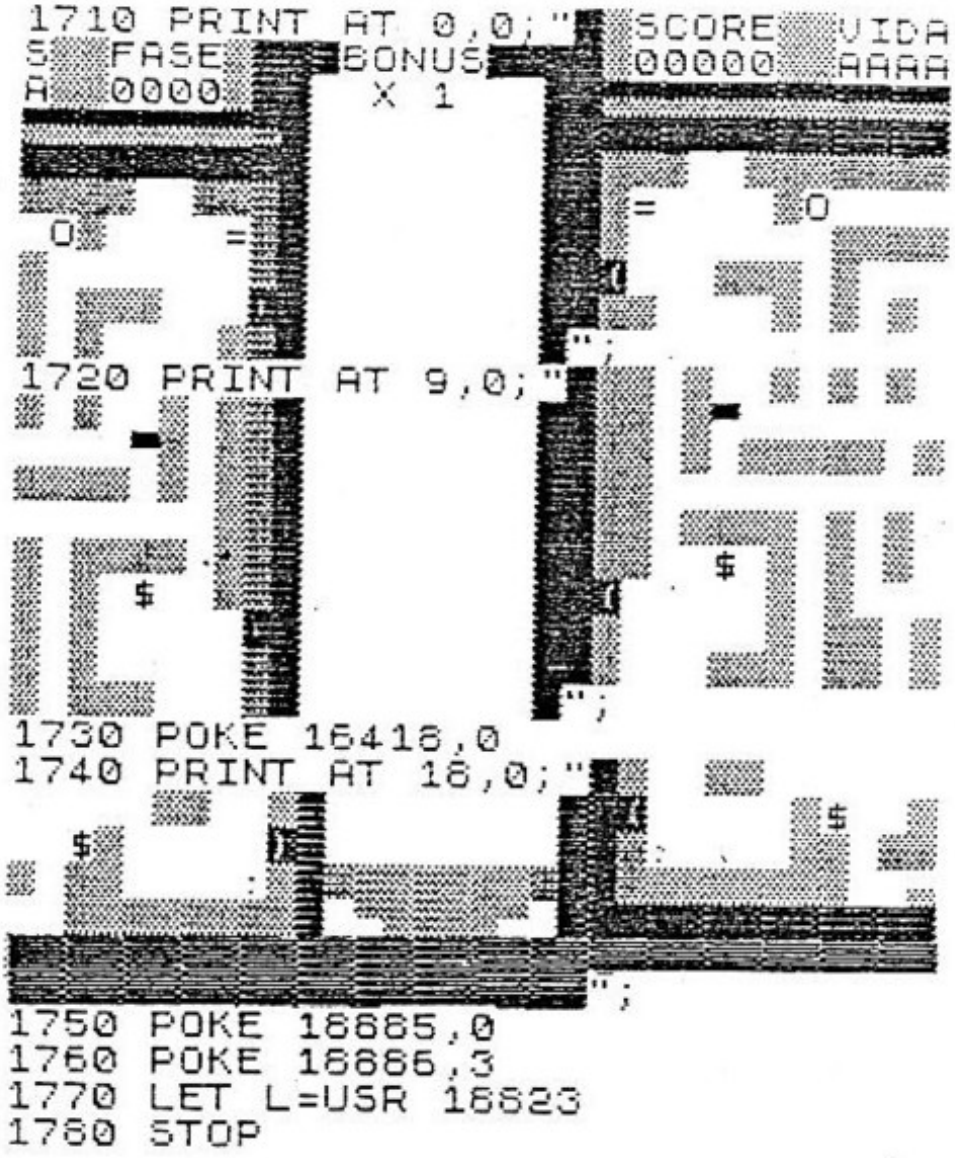
```
MAPA DE VARIAVEIS BURGERTIME

A-SUA POSICAO ATUAL
B-SUA POSICAO ANTERIOR
C-
D-ENDERECO D-FILE
E-
F-
G-
H-
I-
J-POSICAO COMILAO
K-
L-LEITURA DE TECLAS E 2■ LOOP
M-
N-ALTURA DO HAMBURGER
O-PEEK DA SUA POSICAO
P-VELOCIDADE
Q-
R-
S-PLACAR
T-FASE
U-MULTIPLICADOR DE BONUS
V-VIDAS
W-
X-CONTROLADOR INDEXADOR E LOOP
Y-
Z(1,2)-POSICAO ATUAL DOS MONTROS
Z(3,4)-POSICAO ANTERIOR DOS
       MONTROS
Z(5,6)-PEEK DA POSICAO ANTERIOR
       DOS MONSTROS
Z(7,8)-VETOR ATUAL DOS MONTROS


  10 GOTO 30
  20 PRINT "Digite 77 pontos"
  30 GOSUB 1460
  40 REM ▒▒▒ LEITURA TECLAS ▒▒▒
  50 LET L=CODE INKEY$
  60 IF L=33 THEN IF PEEK (A-1) <
>136 THEN LET A=A-1
  70 IF L=34 THEN IF PEEK (A+33)
<>136 THEN LET A=A+33
  80 IF L=35 THEN IF PEEK (A-33)
<>136 THEN LET A=A-33
  90 IF L=36 THEN IF PEEK (A+1) <
>136 THEN LET A=A+1
 100 REM ▒▒▒ TESTE POSICAO ▒▒▒
```

```
 110 LET O=PEEK A
 120 IF O=128 THEN LET A=B
 130 IF O=144 THEN LET A=A+19
 140 IF O=145 THEN LET A=A-19
 150 LET O=PEEK A
 160 IF O<>127 THEN IF O<>0 THEN
 GOTO 730
 170 POKE A,38
 180 IF A<>B THEN POKE B,127
 190 FOR X=1 TO P
 200 NEXT X
 210 LET B=A
 220 REM INTELIGENCIA ARTIFICIAL
 230 FOR X=1 TO 2
 240 REM ▒▒▒TESTE PROXIMIDADE▒▒▒
 250 IF PEEK (Z(X)+1)<>38 THEN I
F PEEK (Z(X)-1)<>38 THEN IF PEEK
  (Z(X)+33)<>38 THEN IF PEEK (Z(X
)-33)<>38 THEN GOTO 320
 260 IF PEEK (Z(X)+1)=38 THEN LE
T Z(X+6)=1
 270 IF PEEK (Z(X)-1)=38 THEN LE
T Z(X+6)=-1
 280 IF PEEK (Z(X)+33)=38 THEN L
ET Z(X+6)=33
 290 IF PEEK (Z(X)-33)=38 THEN L
ET Z(X+6)=-33
 300 GOTO 400
 310 REM ▒▒▒TESTE RASTRO▒▒▒
 320 IF PEEK (Z(X)+1)<>127 THEN
IF PEEK (Z(X)-1)<>127 THEN IF PE
EK (Z(X)+33)<>127 THEN IF PEEK (
Z(X)-33)<>127 THEN GOTO 430
 330 IF PEEK (Z(X)+Z(X+6))=127 T
HEN GOTO 400
 340 LET Z(X+6)=0
 350 IF PEEK (Z(X)+1)=127 THEN I
F RND>10000 THEN LET Z(X+6)=1
 360 IF PEEK (Z(X)-1)=127 THEN I
F RND>10000 THEN LET Z(X+6)=-1
 370 IF PEEK (Z(X)+33)=127 THEN
IF RND>10000 THEN LET Z(X+6)=33
 380 IF PEEK (Z(X)-33)=127 THEN
IF RND>10000 THEN LET Z(X+6)=-33
 390 IF Z(X+6)=0 THEN GOTO 350
 400 LET Z(X)=Z(X)+Z(X+6)
 410 GOTO 530
 420 REM ▒▒▒TESTE BUSCA▒▒▒
 430 IF PEEK (Z(X)+Z(X+6))<128 T
HEN GOTO 400
 440 LET F=Z(X+6)
 450 LET Z(X+6)=0
```

```
 460 IF PEEK (Z(X)+1)<128 THEN I
F F<>-1 THEN IF RND>10000 THEN L
ET Z(X+6)=1
 470 IF PEEK (Z(X)-1)<128 THEN I
F F<>1 THEN IF RND>10000 THEN LE
T Z(X+6)=-1
 480 IF PEEK (Z(X)+33)<128 THEN
IF F<>-33 THEN IF RND>10000 THEN
 LET Z(X+6)=33
 490 IF PEEK (Z(X)-33)<128 THEN
IF F<>33 THEN IF RND>10000 THEN
LET Z(X+6)=-33
 500 IF Z(X+6)=0 THEN GOTO 460
 510 GOTO 400
 520 REM ▒▒CONTROLE MONSTROS▒▒
 530 POKE Z(2+X),Z(4+X)
 540 LET Z(4+X)=PEEK Z(X)
 550 IF Z(4+X)=38 THEN GOTO 1230
 560 IF Z(4+X)=50 THEN LET Z(4+X
)=0
 570 IF Z(X+4)=127 THEN LET Z(X+
4)=0
 580 IF Z(X+4)=12 THEN LET Z(X+4
)=0
 590 POKE Z(X),50
 600 LET Z(2+X)=Z(X)
 610 NEXT X
 620 REM ▒▒CONTROLE COMILAO▒▒
 630 IF J=0 THEN GOTO 50
 640 POKE J,0
 650 IF RND>10000 THEN IF PEEK (
J-1)<128 THEN LET J=J-1
 660 IF RND>10000 THEN IF PEEK (
J+1)<128 THEN LET J=J+1
 670 IF RND>10000 THEN IF PEEK (
J+33)<128 THEN LET J=J+33
 680 IF RND>10000 THEN IF PEEK (
J-33)<128 THEN LET J=J-33
 690 IF PEEK J=38 THEN LET J=0
 700 POKE J,12
 710 GOTO 50
 720 REM ▒▒MONTAGEM HAMBURGER▒▒
 730 IF O=12 THEN LET J=0
 740 IF O=50 THEN GOTO 1230
 750 IF O<>52 THEN GOTO 790
 760 PRINT AT N,23;"  ▬▒▬  ";AT
(N+1),23;"▀▀▀▀▀▀▀▀";
 770 LET S=S+1
 780 LET N=N-2
 790 IF O<>20 THEN GOTO 830
 800 PRINT AT N,23;"========";AT
 (N+1),23;"========";
 810 LET S=S+1
```

```
 820 LET N=N-2
 830 IF 0<>3 THEN GOTO 870
 840 PRINT AT N,23;"  ▚▚▚ ▚▚  ",AT
(N+1),23;"  ▓▓▓▓▓▓▓▓  ";
 850 LET S=S+1
 860 LET N=N-2
 870 IF 0<>14 THEN GOTO 910
 880 PRINT AT N,23;":::::::::";AT
 (N+1),23;":::::::::";
 890 LET S=S+1
 900 LET N=N-2
 910 IF 0<>9 THEN GOTO 950
 920 PRINT AT N,23;"  ▒▒▒▒▒▒  ";AT
(N+1),23;"▒▒▒▒▒▒▒▒▒▒▒▒";
 930 LET S=S+1
 940 GOTO 1010
 950 IF 0<>13 THEN GOTO 980
 960 LET S=S+1
 970 LET U=U+1
 980 GOSUB 1660
 990 GOTO 170
1000 REM ▒▒▒ DESCARREGA BONUS ▒▒▒
1010 FOR X=N TO 18
1020 LET S=S+U
1030 PRINT AT X,23;"          ";AT
  (X+1),23;"          "
1040 GOSUB 1660
1050 NEXT X
1060 LET N=18
1070 REM ▒▒▒ PASSA FASE ▒▒▒
1080 POKE A,0
1090 POKE B,0
1100 POKE J,0
1110 FOR X=1 TO 4
1120 POKE Z(X),0
1130 NEXT X
1140 LET Z(7)=33
1150 LET Z(8)=-33
1160 PRINT AT 21,11;"▒";AT 5,2;"
=";AT 5,8;"O      O";AT 5,20;"=";
AT 10,5;"▬                ▬";AT 14,5;
"$";AT 14,17;"$";AT 19,8;"$  ▒
$";AT 20,2;":";AT 20,20,":";
1170 LET T=T+1
1180 GOSUB 1660
1190 IF P>50 THEN LET P=P-50
1200 GOSUB 1500
1210 GOTO 50
1220 REM ▒▒▒ + MORREU + ▒▒▒
1230 FOR X=0 TO 8
1240 POKE B,0
1250 POKE A,38
1260 FOR L=1 TO 1000
```

```
1270 NEXT L
1280 POKE A,166
1290 FOR L=1 TO 1000
1300 NEXT L
1310 NEXT X
1320 POKE A,0
1330 FOR X=1 TO 10000
1340 NEXT X
1350 POKE A,0
1360 POKE J,0
1370 FOR X=1 TO 4
1380 POKE Z(X),0
1390 NEXT X
1400 LET V=V-1
1410 IF V=-1 THEN GOTO 1700
1420 PRINT AT 1,(13-V);"■"
1430 GOSUB 1500
1440 GOTO 50
1450 REM ▒▒▒INICIALIZA▒▒▒
1460 LET S=0
1470 LET V=5
1480 LET T=1
1490 LET N=16
1500 LET D=PEEK 16396+256*PEEK 1
6397
1510 LET A=D+406
1520 LET B=A
1530 LET J=D+251
1540 LET U=1
1550 LET Z(1)=D+235
1560 LET Z(2)=D+606
1570 LET Z(3)=Z(1)
1580 LET Z(4)=Z(2)
1590 LET Z(5)=0
1600 LET Z(6)=0
1610 LET Z(7)=33
1620 LET Z(8)=-33
1630 GOTO 1660
1640 RETURN
1650 REM ▒▒▒CONTROLE PLACAR▒▒▒
1660 IF S>9999 THEN LET S=S-1000
0
1670 IF T>9999 THEN LET T=T-1000
0
1680 LET L=USR 18934
1690 RETURN
1700 STOP
```



```
1710 PRINT AT 0,0;"
1720 PRINT AT 9,0;"
1730 POKE 16418,0
1740 PRINT AT 18,0;"
1750 POKE 16885,0
1760 POKE 16886,3
1770 LET L=USR 16823
1780 STOP
```

Soluções Técnicas — Burgertime

Após intrigar a todos com os absurdos programados no BURGER, temos agora que explicar, brevemente, como o computador gera o vídeo.

Na parte de SOFTWARE, temos um programa que inicia em $(38)_{16}$ e, que a cada 1/66 de segundo é chamado pelo processador sendo encarregado de mandar o D-FILE para os circuitos externos ao processador Z-80A.

A parte de HARDWARE é um pouco mais complicada. Ela divide-se, basicamente em 2 partes.

Um deles é como um "relógio" que tem por função chamar aquele programa no $(38)_{16}$ no tempo certo. Este não nos interessa por enquanto.

O outro, mais importante agora, é um CI, chamado de 74LS373, que se localiza na placa, ao lado da Z-80 A, ou dentro da famosa ULA.

É este CI que identifica qual o caractere que está na tela e manda-o para a televisão.

Este também é o CI que ocasiona a maioria dos "CRASHES" em que o computador entra.

Isto ocorre porque o nosso computador tem apenas 64 caracteres normais, mais os 64 caracteres inversos. Porém, há 256 códigos. Cuidado, estes 128 restantes são controlados por SOFTWARE, portanto eles não existem, são combinações de outros.

É justamente com esses caracteres que a 74LS373 confunde-se e manda tudo para o espaço.

Mas como toda a regra tem sua exceção, há alguns caracteres com comportamento interessante. Um deles é o de código 127.

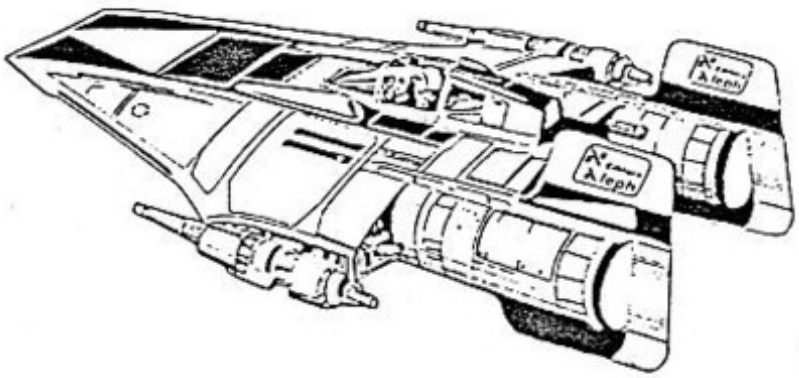
Se ele é POKEado no D-FILE, ele torna-se invisível, porém está lá. E é justamente este efeito que foi usado no BURGER, para deixar um rastro invisível atrás de você, que pode ser seguido pelos monstros.

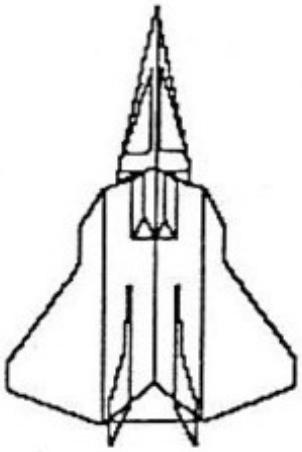
Achamos que agora, após esta explicação, tudo está mais claro. Explicaremos agora a rotina de inteligência artificial do BURGER.

Na verdade, a rotina de inteligência artificial é separada em três distintas.

A primeira parte é apenas para o caso de você estar grudado no monstro. A segunda parte é a que segue o rastro invisível deixado por você. A terceira parte escolhe caminhos aleatórios para o caso de não haver rastros na região vasculhada.

O programa BURGERTIME é o exemplo de que se pode fazer um programa até certo ponto complexo, com pouca memória e poucas variáveis.





CAPÍTULO 12

O JOGO FINAL

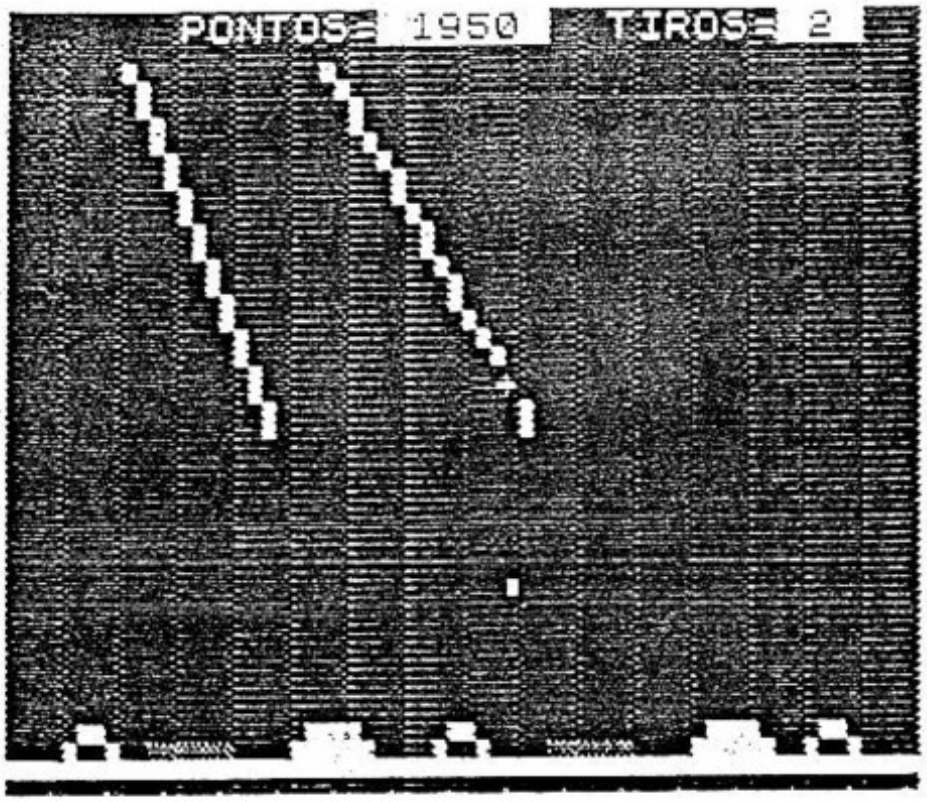
Este capítulo encerra a série de jogos deste livro com o programa Missile Command.

A "Míssil" (com 'a' mesmo), como ficou mais conhecido entre nós, é um programa com "pedigree" que já chegou à sua quinta versão, que será apresentada a vocês.

A história da Míssil:

É uma longa história. Tudo começou com o lançamento da máquina "MISSILE X" nos fliperamas de São Paulo. Este fato motivou parte de nossa turma, o extinto JKF, a fazer um jogo ainda em Basic interpretado que usasse o PLOT e o UNPLOT para traçar o rastro dos mísseis. Imagine você mesmo os resultados: um jogo sonolento e entediante.

Com o surgimento do compilador entre nós, a Míssil recebeu prioridade 1 de desenvolvimento. Entretanto, deparamos com um terrível problema: o compilador não trabalhava com números fracionários. Por causa disto, os mísseis caíam apenas verticalmente ou a 45 graus. Quando Milton descobriu como simular casas decimais, a Míssil ganhou a sua primeira versão definitiva, com o mesmo visual da versão BASIC.



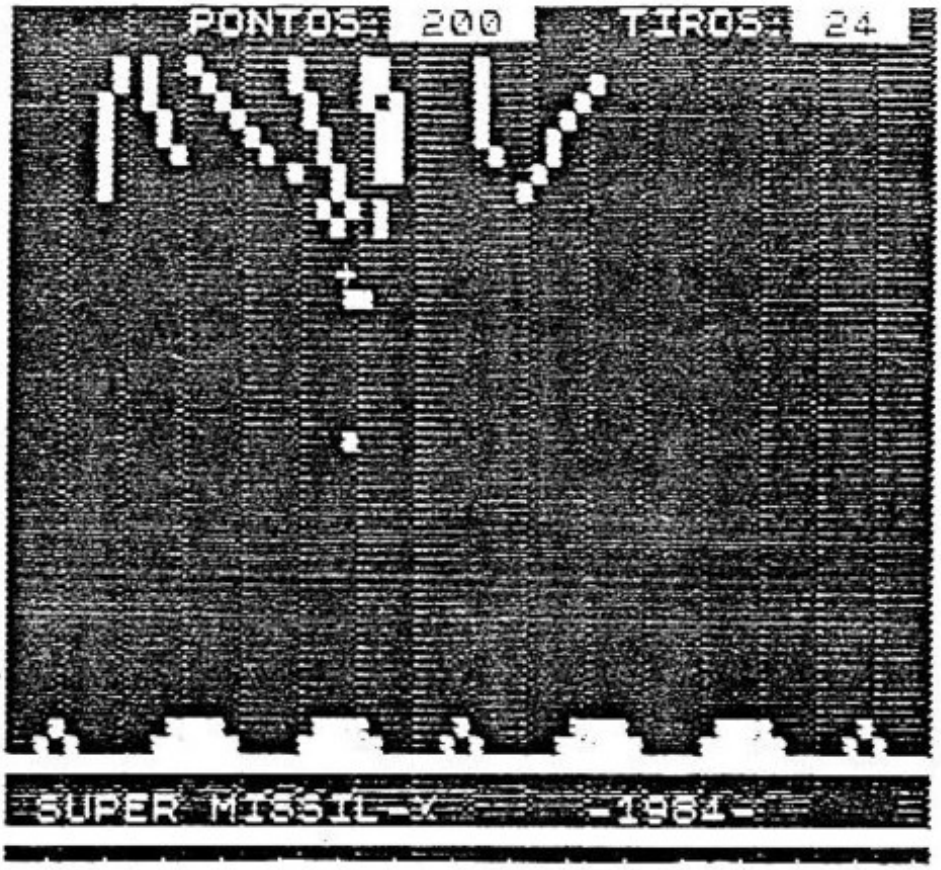
TELA DA VERSAO 1 E 2

Animados com os primeiros resultados (afinal, o jogo já era superior a muitos jogos comercializados), tentamos vendê-lo à empresa Microsoft, atual Multisoft, que nos parecia uma software-house séria. Qual não foi a nossa decepção quando constatamos o total descaso da citada firma ao extraviar as duas cópias que enviamos (o responsável sequer deu satisfação do ocorrido).

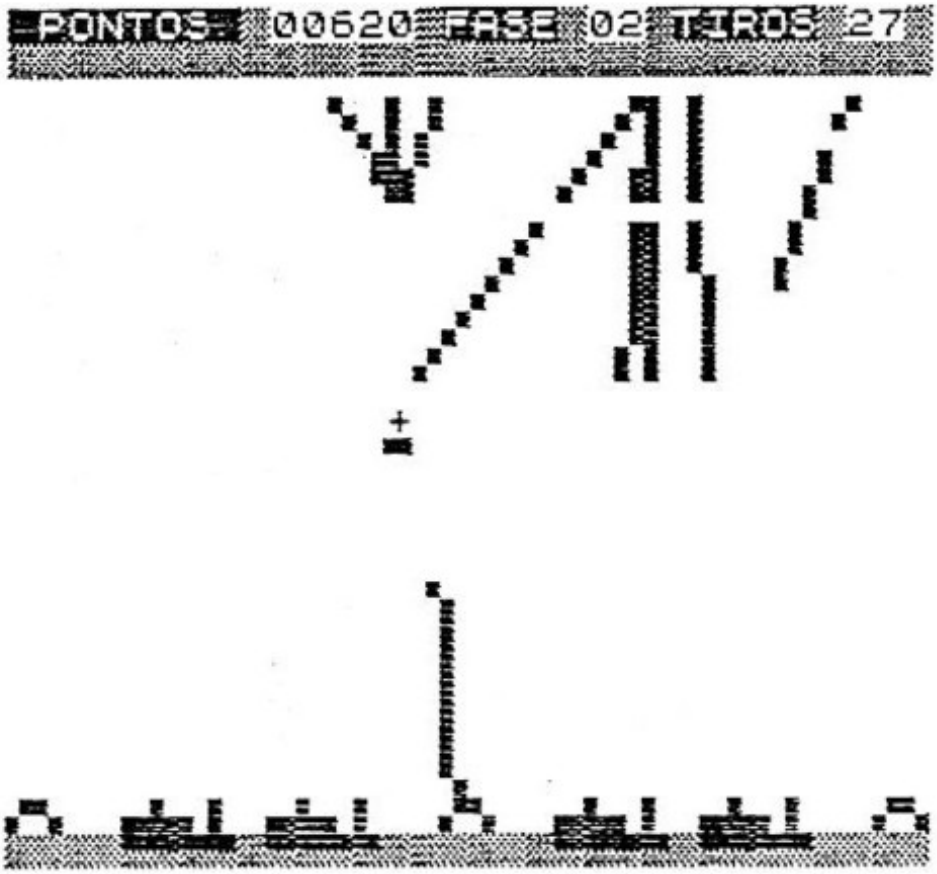
Passada a desilusão, a Míssil sofreu várias adaptações até chegar à versão II, que já era mais fiel ao original do fliperama.

Cerca de um ano depois, entramos em contato com dois estudantes estagiários da empresa Engesoft, produtora dos jogos Micro Game. Fechado o negócio, foi feita com exclusividade a versão III, que tinha como principal diferença de suas predecessoras, bases inteligentes lançadoras de mísseis. Era escolhida a base mais próxima do alvo e lançava-se o anti-míssil dela. Outra grande modificação foi a inclusão da rotina JOY 3, que possibilitou maior agilidade no jogo.

Após um quebra-quebra inicial, causado por uma cópia não autorizada de versão II, em alta resolução gráfica, que se espalhou por diversos pontos do Brasil, vimo-nos na obrigação profissional de criar uma nova versão para a Engesoft. Nasceu então a versão IV, totalmente reestruturada e remodelada, com novo visual e ação mais envolvente. Foi um grande passo na sua história.



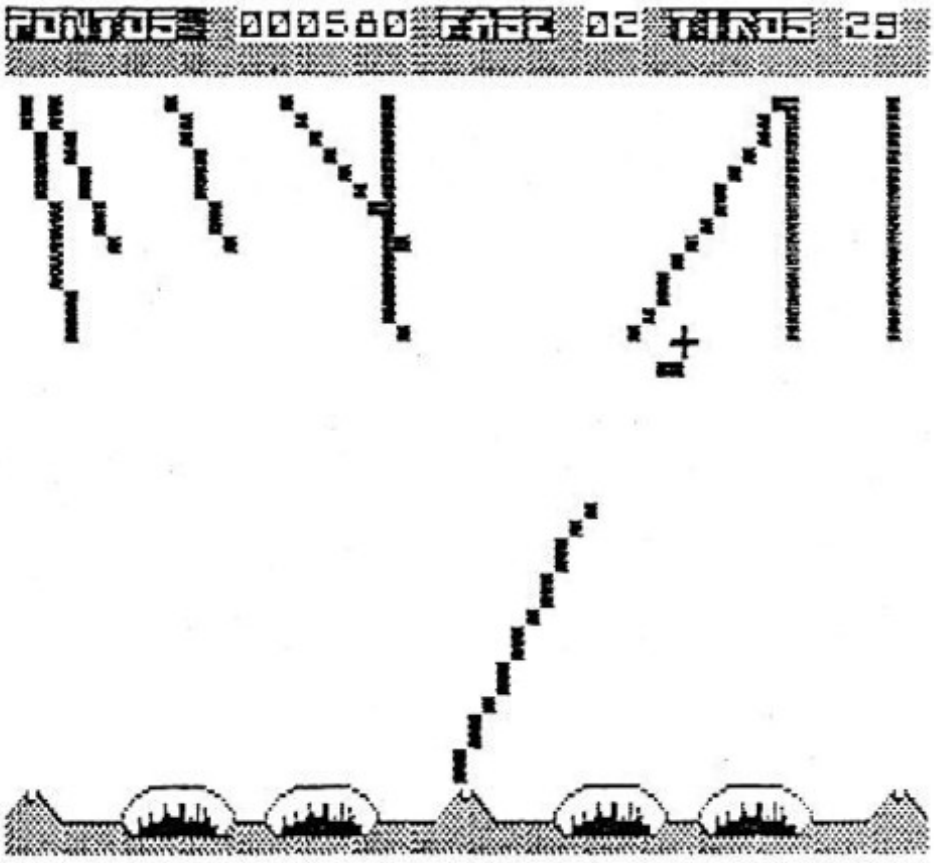
TELA DA VERSAO 3



TELA DA VERSAO 4

Da versão IV para a versão V, pouca coisa mudou em termos de visual. O placar ganhou mais um dígito e o "lay-out" dos cogumelos atômicos foi mudado. Mas por dentro praticamente tudo mudou. O mapa das variáveis foi totalmente refeito, as aproximações fracionárias ficaram dez vezes mais precisas e o programa foi compactado. Isto deu uma mira mais precisa aos mísseis inimigos e ao seu próprio. É a melhor versão feita até o momento para resolução gráfica normal e sem gerador de som.

A história não para aqui. O Missile Command está em constante evolução (lembre-se do que dissemos na introdução) e quem sabe a versão VI não será escrita por você? Fica aqui um convite a todos.



TELA DA VERSAO 5 HGR SOM

A ação do jogo:

Para quem não conhece o "Missile Command" (deve ser muito pouca gente), a ação desenrola-se em um vale no qual estão situadas quatro cidades que são alvos estratégicos do inimigo. Ele tentará destrui-la a todo custo, e a sua missão é proteger o vale contra os mísseis intercontinentais lançados por ele. Para isto, o vale é armado com três bases lançadoras de anti-míssil e programado por uma mira móvel que alcança todo o espaço aéreo da região. Para dispará-lo pressiona-se a tecla Ø ou o botão de tiro do joystick.

A cada fase você ganha um bônus especial, que varia em função do número de cidades destruídas e de tiros consumidos, sendo estes no máximo 3Ø por vez. Se você gastar todos os tiros e não conseguir abater todas as bombas, correrá o sério risco de ver cidades inteiras sendo pulverizadas e milhões de pessoas aniquiladas em frações de segundo. Se uma base for atingida, você perde 1Ø cargas de munição.

Sempre que você consegue rechaçar um ataque, outro mais feroz sucede o anterior e a luta continua até o seu fim. . .

Digitação:

Bem, este é nada menos que o último jogo do livro, logo você já deve estar acostumado com o sistema de digitação.

```
*18836 00 00 00 00 00 21 F6 279
 18843 49 E5 05 05 36 FF 23 657
*18850 10 FB CD B6 02 E5 D1 1099
 18857 E1 CB 4A 20 02 36 00 590
*18864 23 CB 5A 20 02 36 00 416
 18871 23 CB 62 20 02 36 00 424
*18878 23 CB 5B 20 02 36 00 417
 18885 23 28 06 CB 6A 20 02 424
*18892 36 00 CB 5B 20 12 CB 601
 18899 6A 20 0E CB 63 20 0A 496
*18906 CB 4A 28 06 CB 62 28 664
 18913 02 36 00 C9 ED 5B 0C 597
*18920 40 06 0A 13 10 FD 2A 410
 18927 BD 49 CD 6D 4A 06 07 663
*18934 13 10 FD 2A DD 49 CD 829
 18941 79 4A 06 07 13 10 FD 496
*18948 2A E1 49 CD 79 4A C9 941
 18955 01 E8 03 CD 86 4A 01 650
*18962 64 00 CD 86 4A 01 0A 524
 18969 00 CD 85 4A 01 01 00 415
*18976 CD 86 4A C9 97 3E 1C 855
 18983 ED 42 38 03 3C 18 F9 695
*18990 09 12 13 C9 FF
```

CUIDADO!!! Neste programa delete as linhas do monitor antes de compilar.

```
  10 GOTO 30
  20 PRINT "Digite 159 pontos"
  30 REM ▒▒▒INICIALIZACAO▒▒▒
  40 LET Y=PEEK 16396+256*PEEK 1
6397
  50 LET K=2
  60 LET M=Y+727
  70 LET N=0
  80 LET P=10
```

```
 90 LET U=0
100 LET V=1
110 LET W=30
120 REM ▒▒▒▒DIRECIONA MISSEIS▒▒▒▒
130 FOR L=1 TO 8
140 GOSUB 1350
150 NEXT L
160 LET A=31
170 LET B=21
180 LET Q=0
190 REM ▒▒▒▒LOOP CENTRAL▒▒▒▒
200 FOR Q=0 TO P
210 LET C=A
220 LET D=B
230 IF Q=16 THEN GOTO 1470
240 REM ▒▒▒▒LE TECLADO▒▒▒▒
250 LET R=USR 18939
260 PAUSE 1
270 IF PEEK 18934=0 THEN IF W>0
THEN IF K=2 THEN LET K=0
280 IF PEEK 18935=0 THEN LET A=
A+1
290 IF PEEK 18937=0 THEN LET A=
A-1
300 IF PEEK 18936=0 THEN LET B=
B+1
310 IF PEEK 18938=0 THEN LET B=
B-1
320 IF A<1 THEN LET A=1
330 IF A>60 THEN LET A=60
340 IF B>35 THEN LET B=35
350 IF B<1 THEN LET B=1
360 IF K=0 THEN GOTO 910
370 IF K=1 THEN GOTO 1070
380 IF A=C THEN IF B=D THEN GOT
O 410
390 UNPLOT C,D
400 UNPLOT (C+1),D
410 PLOT A,B
420 PLOT (A+1),B
430 IF K<>2 THEN GOTO 460
440 FOR L=0 TO (10*P)
450 NEXT L
460 NEXT Q
470 REM ▒▒▒▒CONTROLA MISSEIS▒▒▒▒
480 FOR L=1 TO 8
490 IF Z(L)=8000 THEN GOTO 540
500 LET Z(L)=Z(L)+Z(L+24)
510 LET Z(L+16)=Z(L+16)-1
520 PLOT (Z(L)/100),Z(L+16)
530 IF Z(L+16)=0 THEN GOSUB 116
0
540 NEXT L
```

```
 550 GOTO 200
 560 REM ▒▒▒EXPLOSAO▒▒▒
 570 LET W=W-1
 580 POKE (S-34),129
 590 POKE (S-33),128
 600 POKE (S-32),130
 610 POKE (S-1),128
 620 POKE S,128
 630 POKE (S+1),128
 640 POKE (S+32),132
 650 POKE (S+33),128
 660 POKE (S+34),7
 670 FOR L=0 TO 400
 680 NEXT L
 690 FOR L=32 TO 34
 700 POKE (S+L),0
 710 POKE (S-L),0
 720 NEXT L
 730 POKE (S-1),0
 740 POKE (S+1),0
 750 POKE S,0
 760 GOSUB 2240
 770 REM ▒▒▒APAGA RASTRO▒▒▒
 780 LET J=0
 790 LET I=(PEEK 19092)*100
 800 UNPLOT (I/100),(J/100)
 810 LET I=I+G
 820 LET J=J+H
 830 IF J<(F*100) THEN GOTO 800
 840 FOR L=1 TO 8
 850 IF Z(L)=800 THEN GOTO 870
 860 IF ABS (E-Z(L)/100)<=3 THEN
 IF ABS (F-Z(L+16))<=3 THEN GOSU
B 1250
 870 NEXT L
 880 LET K=2
 890 GOTO 320
 900 REM ▒▒▒DISPARA MISSIL▒▒▒
 910 LET K=1
 920 LET E=A+1
 930 LET F=B+2
 940 LET S=(21-F/2)*33+1+E/2+Y
 950 LET I=31
 960 IF E<17 THEN LET I=2
 970 IF E>46 THEN LET I=61
 980 LET J=0
 990 IF ABS (E-I)>=F THEN GOTO 1
030
1000 LET G=(100*(E-I))/F
1010 LET H=100
1020 GOTO 1050
1030 LET G=(100*(E-I))/ABS (E-I)
1040 LET H=(100*F)/ABS (E-I)
```

```
1050 POKE 19092,I
1060 LET I=I*100
1070 FOR L=0 TO 3
1080 PLOT (I/100),(J/100)
1090 LET I=I+G
1100 LET J=J+H
1110 IF J>=(F*100) THEN GOTO 570
1120 NEXT L
1130 POKE 6,21
1140 GOTO 380
1150 REM ▒▒▒TESTA=COLIDADE▒▒▒
1160 LET T=(M+Z(L)/200)
1170 IF PEEK T=134 THEN GOSUB 16
30
1180 IF PEEK T=6 THEN GOSUB 1630
1190 IF PEEK T=131 THEN GOSUB 16
60
1200 IF PEEK T=130 THEN GOSUB 16
70
1210 IF PEEK T=4 THEN GOSUB 1690
1220 IF PEEK T=5 THEN GOSUB 1710
1230 GOTO 1260
1240 REM ▒▒▒APAGA=RASTRO=A/I▒▒▒
1250 LET N=N+1
1260 LET Z(L)=Z(L+8)
1270 FOR X=Z(L+16) TO 39
1280 UNPLOT (Z(L)/100),(39-X+Z(L
+16))
1290 LET Z(L)=Z(L)+Z(L+24)
1300 NEXT X
1310 GOSUB 2240
1320 LET O=O+1
1330 IF O>8 THEN GOTO 1450
1340 REM ▒▒▒DIRECIONA=MISSIL▒▒▒
1350 LET Z(L+8)=RND/53
1360 LET Z(L+8)=Z(L+8)*10
1370 LET X=RND/53
1380 LET X=X*10
1390 IF PEEK (M+X/200+1)=0 THEN
GOTO 1370
1400 LET Z(L)=Z(L+8)
1410 LET Z(L+16)=39
1420 LET Z(L+24)=((X-Z(L+8))/41)
1430 IF ABS Z(L+24)>100 THEN LET
 Z(L+24)=Z(L+24)/ABS Z(L+24)
1440 RETURN
1450 LET Z(L)=8000
1460 RETURN
1470 GOSUB 2190
1480 REM ▒▒▒BONUS▒▒▒
1490 IF U=4 THEN IF W<=0 THEN GO
TO 1610
1500 PRINT AT 10,1;"GANHOU BONUS
 X";U
```

```
1510 GOSUB 1970
1520 LET P=P-1
1530 LET V=V+1
1540 IF P<1 THEN LET P=2
1550 LET W=30
1560 LET K=2
1570 PRINT AT 10,1;"
"
1580 IF U=4 THEN GOTO 1610
1590 GOSUB 2240
1600 GOTO 130
1610 PRINT AT 10,11;"FIM DE JOGO
";
1620 GOTO 2310
1630 LET W=W-10
1640 GOTO 2240
1650 REM ▒▒▒EXPLOSAO ATOMICA▒▒▒
1660 GOTO 1720
1670 LET T=T-1
1680 GOTO 1720
1690 LET T=T-2
1700 GOTO 1720
1710 LET T=T-3
1720 LET U=U+1
1730 FOR X=1 TO 3
1740 POKE T,135
1750 POKE (T+1),137
1760 POKE (T+2),137
1770 POKE (T+3),4
1780 FOR R=1 TO 2000
1790 NEXT R
1800 POKE T,0
1810 POKE (T+1),133
1820 POKE (T+2),5
1830 POKE (T+3),0
1840 LET T=T-33
1850 NEXT X
1860 LET T=T+33
1870 POKE T,0
1880 POKE (T+1),0
1890 POKE (T+2),0
1900 POKE (T+3),0
1910 POKE (T+34),0
1920 POKE (T+35),0
1930 POKE (T+67),127
1940 POKE (T+68),127
1950 RETURN
1960 REM ▒▒▒ANIMACAO BONUS▒▒▒
1970 IF U=4 THEN GOTO 2070
1980 FOR Q=0 TO (3-U)
1990 FOR X=1 TO 500
2000 NEXT X
```

```
2010 PRINT AT 12,(Q*5);"   ";AT
 13,(Q*5);"   ";
2020 LET N=N+(5*V)
2030 GOSUB 2240
2040 NEXT Q
2050 FOR X=0 TO 2000
2060 NEXT X
2070 IF W<=0 THEN GOTO 2160
2080 FOR Q=0 TO (W-1)
2090 LET W=W-1
2100 LET N=N+V
2110 GOSUB 2240
2120 PRINT AT 14,Q;" ";
2130 NEXT Q
2140 FOR X=0 TO 2000
2150 NEXT X
2160 GOSUB 2190
2170 RETURN
2180 REM   LIMPA TELA
2190 FOR L=2 TO 21
2200 PRINT AT L,0;"
                              ";
2210 NEXT L
2220 RETURN
2230 REM   CONTROLA PLACAR
2240 IF N<=9999 THEN GOTO 2270
2250 POKE (Y+9),(PEEK (Y+9)+1)
2260 LET N=N-10000
2270 IF W<0 THEN LET W=0
2280 IF V>99 THEN LET V=0
2290 LET R=USR 19015
2300 RETURN
2310 STOP
9000 POKE 16418,0
9010 PRINT AT 0,0;"PONTOS  00000
0 FASE 00 TIROS 30
                       ";
9020 PRINT AT 22,0;"
                              ";
9030 LET L=USR 18823
```

SOLUÇÕES TÉCNICAS – MISSILE COMMAND

Apesar da "Missil" ser um jogo complexo, com várias novidades em termos de programação, a parte que receberá maior atenção será a do indexador de variáveis usado, porém não explicado no capítulo anterior.

A parte de controle de tiro, a ECC, já é uma velha conhecida desde

os jogos "Space" e "intruder". A movimentação da mira é feita com o auxílio da rotina JOY3, que possibilita movimento em todas as direções possíveis no joystick e simultaneidade do tiro.

O controle dos pontos na tela foi dado a uma rotina específica na linha 2240, que tem por função o ajuste de parâmetros e limites de impressão, bem como o de controlar o 6º dígito do placar.

Não foi esquecida a lição do capítulo 11 sobre o caractere invisível; ele foi usado para que a rotina direcionadora de mísseis não ignore as cidades já atingidas, escolhendo somente as cidades restantes, o que deixaria o jogo muito fácil.

Esse programa é o típico exemplo de que às vezes é necessário usar casas decimais em meio a tantos inteiros. Optou-se por 2 casas decimais (divisão por 100), para dar maior precisão aos raios.

Bem, vamos à parte que interessa.

O indexador de variáveis nada mais é do que um "ponteiro" usado para apontar a dimensão desejável da variável Z (...). Nesse caso, na "Missil", o ponteiro seleciona as variáveis de 8 em 8.

— "No que isso pode ajudar?" pergunta você.

É um princípio simples, porém extremamente útil para economia de memória. Nesse programa existe apenas uma rotina de cada tipo: uma para direcionar o míssil, uma para plotá-lo, uma para testá-lo, apagá-lo, e assim por diante. Porém, o jogo controla 8 raios a cada ciclo. Por que?

A resposta é fácil: dentro de cada ciclo de controle do jogo temos um ciclo especial para controlar os raios. Esse ciclo é um loop de 1 a 8 controlado pela variável L. Quando ela está em 1, os valores que serão usados na rotina serão os de Z(1), Z(9), Z(17) e Z(25), que correspondem exatamente a Z(L), Z(8 + L), Z(16 + L) e Z(24 + L), com L valendo 1,2, . . . e assim por diante, até 8.

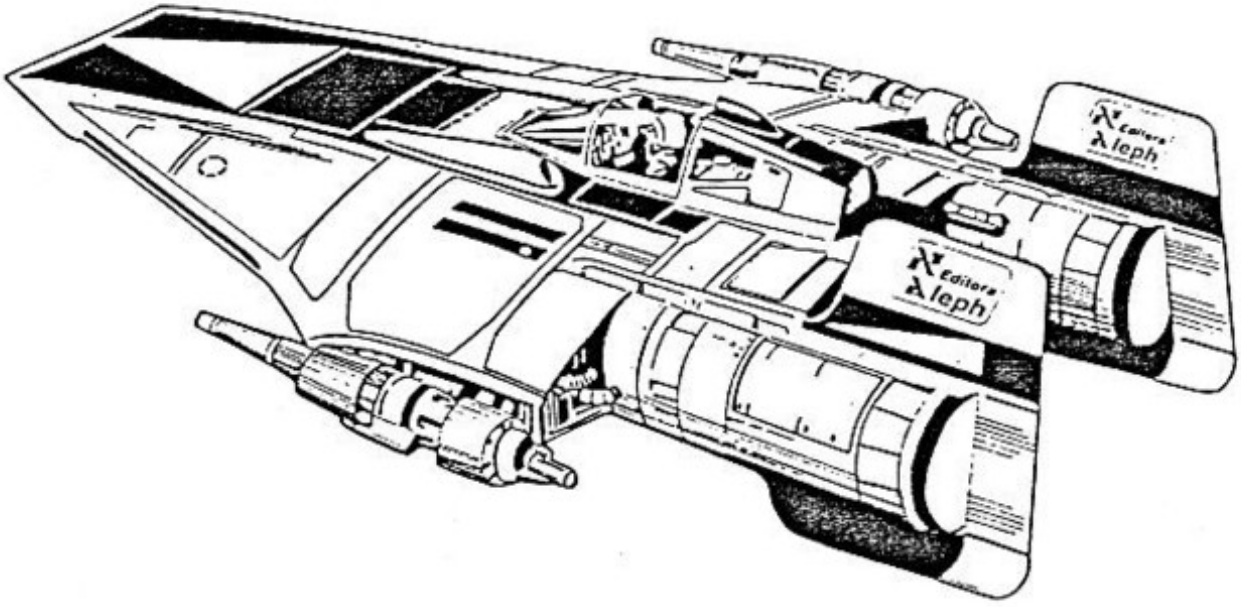
Um pequeno problema, porém praticamente impossível de ser sanado, é que às vezes aparecem caracteres "estranhos" no lugar onde deveria estar plotado um raio.

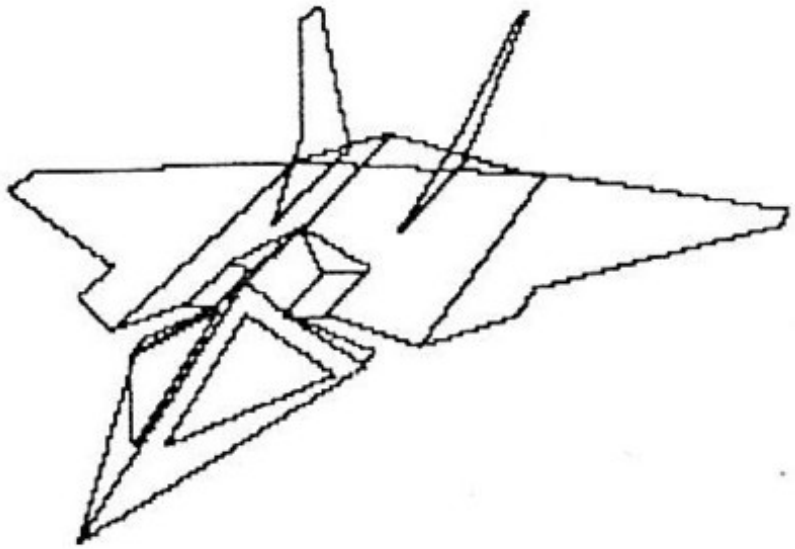
Isto se dá porque a rotina de PLOT do compilador, para poder ser ainda mais rápida, exclui os testes que verificam se na posição do PLOT há um caractere gráfico ou não. Se há, o PLOT se dá normalmente; se não, aparecem caracteres inversos ou estranhos.

Outra boa observação acerca de programação é que a "Míssil" é totalmente modular, permitindo adaptações e mudanças que ficam a gosto do freguês, pois ainda há relativamente bastante memória.

Chegando ao final do último "soluções técnicas" deste livro, espe-

ramos ter deixado a impressão de que nada que se deseja fazer é impossível de conseguir e, mais do que isso, deixado "bagagem" e ensinamentos suficientes para que você próprio crie as suas soluções "malucas" e, dentro das possibilidades gráficas, faça qualquer jogo já existente, ou alguma nova idéia em um campo tão vasto, complexo e fascinante que é o universo de software para o TK.





APÊNDICE 1

Este apêndice destina-se a prover o leitor do material necessário para o acompanhamento do livro. Aqui serão dadas as instruções de como digitar o próprio compilador, bem como um monitor de vários recursos:

- DIGITAÇÃO DO COMPILADOR
  - Criar uma linha REM com 2301 bytes.
  - Entrar os códigos.
  - Verificar os códigos.
  - Gravar partes do seu árduo trabalho.

- DIGITAÇÃO DOS PROGRAMAS A SEREM COMPILADOS
  - Entrar os códigos nas linhas "PRINT".
  - Conferí-los.
  - Gravar partes do seu árduo trabalho.
  - Deletar as linhas do programa em BASIC após estar tudo compilado.

Para uma melhor compreensão deste apêndice, siga rigorosamente os passos a seguir:

A DIGITAÇÃO DO MONITOR

1 – Digite o monitor (listagem 1) cuidadosamente, principalmente a linha 9500, pois ela contém todos os códigos em Assembly das rotinas de apagar e criar linhas. Após digitar, grave uma vez com SAVE via teclado, por segurança, e outra com GOTO 9500.

```
9400 CLS
9402 PRINT "ENDERECO INICIAL ?"
9404 INPUT E
9406 FOR X=E TO 32767 STEP 7
9408 PRINT AT 21,0;X;AT 21,6;
9410 LET T=0
9412 FOR J=0 TO 6
9414 LET M=PEEK (X+J)
9416 LET T=T+M
9418 PRINT CHR$ (INT (M/16)+28);
CHR$ (M-16*INT (M/16)+28);" ";
9420 NEXT J
9422 PRINT TAB 27;T
9423 SCROLL
9424 PAUSE 4E4
9426 IF INKEY$=CHR$ 116 THEN NEX
T X
9428 GOTO 9580
9500 LET A$="FD08CD230F0106002A0
07FE509444D2A2940092229402 10C403
E095E2356D5EB09EB722B7323233D280
3D118EEE1E5017C40A7ED42444DE1ED8
8217D4036002336002 3C10303712 3702
336EA0B0B23110100EB19EB3676ED803
67534CD2B0FC92ABC7FCDD809E5E52A8
E7F23CDD809EB2A0C40A7ED523004ED5
B0C402A1440E5A7ED52E5C1EBF1D1F5E
DB0EBD1A7ED52E5C1E1C3AD09"
9510 FAST
9520 SAVE "MON"
9530 POKE 16388,0
9540 POKE 16389,127
9550 FOR X=0 TO 149
9560 POKE 32512+X,(CODE A$(X*2+1
)*16+CODE A$(X*2+2)-476)
9570 NEXT X
9580 FAST
9585 CLS
9590 PRINT "1-GRAVAR
           2-ENTRAR DADOS
           3-CRIAR REM DO COMPI
LADOR      4-APAGAR LINHAS
           5-VERIFICAR DADOS"
```

```
9600 PAUSE 4E4
9602 IF INKEY$="1" THEN GOTO 950
0
9604 IF INKEY$="2" THEN GOTO 963
0
9606 IF INKEY$="3" THEN GOTO 987
0
9608 IF INKEY$="4" THEN GOTO 990
0
9610 IF INKEY$="5" THEN GOTO 940
0
9620 GOTO 9600
9630 CLS
9640 PRINT "ENDERECO ? ";
9650 INPUT E
9660 PRINT E
9670 SCROLL
9680 SCROLL
9690 PRINT E;" ";
9700 INPUT A$
9710 IF A$="N" THEN GOTO 9850
9720 PRINT A$;" ";
9730 LET S=0
9740 LET F=0
9750 FOR L=1 TO 20 STEP 3
9760 IF A$(L)="X" THEN GOTO 9580
9770 LET X=16*CODE A$(L)+CODE A$
(L+1)-476
9780 LET S=S+X
9790 POKE E+F,X
9800 LET F=F+1
9810 NEXT L
9820 PRINT S
9830 LET E=E+7
9840 GOTO 9680
9850 LET E=E-7
9860 GOTO 9680
9870 IF PEEK 16509*256+PEEK 1651
0=0 THEN PRINT "A LINHA REM JA E
XISTE"
9880 IF PEEK 16509*256+PEEK 1651
0<>0 THEN RAND USR 32514
9890 PAUSE 300
9895 GOTO 9580
9900 PRINT "PRIMEIRA LINHA ?"
9910 INPUT A
9920 POKE 32701,INT (A/256)
9930 POKE 32700,A-256*PEEK 32701
9940 PRINT "ULTIMA LINHA ?"
9950 INPUT A
9960 POKE 32703,INT (A/256)
9970 POKE 32702,A-256*PEEK 32703
```

```
9980 LET L=USR 32608
9990 GOTO 9580
```

A DIGITAÇÃO DO COMPILADOR

2 — Após a gravação em fita deste programa, ele entrará automaticamente numa espécie de menú. Aperte a tecla 3, a tela dará uma breve piscada, aperte N/L. Pronto: já está criada a linha que alojará o compilador.

Novamente no menú, digite a tecla 2 e comece a entrar os dados da listagem 2.

Para parar de entrar os códigos e voltar ao menú, digite N ao iniciar uma linha. Aproveite de tempos em tempos para gravar o que você já digitou, por segurança.

Após terminar, digite RUN 9500 e grave o compilador e o monitor juntos, pois este será o programa que você utilizará durante todo este livro e os futuros (isto é uma ameaça).

```
*16514 A8 B4 B2 B5 76 76 C3 1138
 16521 B4 40 C3 23 41 C3 57 821
*16528 41 C3 4F 41 C3 85 41 797
 16535 C3 97 41 C3 AB 41 C3 1037
*16542 21 42 C3 9C 42 C3 DE 933
 16549 42 C3 39 43 C3 7A 43 769
*16556 C3 57 43 C3 B8 43 CF 1002
 16563 8C E5 D5 AF CD 21 42 1061
*16570 CB 7C 28 08 3E 16 CD 664
 16577 21 42 CD 1B 41 11 10 429
*16584 27 CD 07 41 30 1A 11 407
 16591 E8 03 CD 07 41 30 15 581
*16598 11 64 00 CD 07 41 30 442
 16605 13 1E 0A CD 07 41 30 384
*16612 12 C3 FC 40 CD 0D 41 812
 16619 11 E8 03 CD 0D 41 11 552
*16626 64 00 CD 0D 41 1E 0A 423
 16633 CD 0D 41 1E 01 CD 0D 532
*16640 41 AF D1 E1 C3 21 42 968
 16647 E5 A7 ED 52 E1 C9 3E 1203
*16654 1C A7 ED 52 38 03 3C 633
 16661 18 F9 19 C3 21 42 7C 716
*16668 2F 67 7D 2F 6F 23 C9 669
 16675 2A 0C 40 11 21 00 E5 397
*16682 01 00 00 09 23 E5 19 299
 16689 EB 21 D6 02 ED 42 E3 1014
*16696 C1 EB ED B0 E1 01 B6 1249
 16703 02 09 22 0E 40 3E 21 218
*16710 32 39 40 3E 03 32 3A 344
 16717 40 C9 CD BB 02 7C FE 1037
*16724 FE 20 0E CD BB 02 24 730
```

```
 16731  00 00 CD BB 02 7D FE  773
#16738  FF 28 F8 01 7F FD AF 1099
 16745  E5 ED 42 E1 CA B2 40 1201
#16752  01 EF FC AF E5 ED 42 1199
 16759  E1 28 DD 44 24 C8 4D 867
#16766  CD BD 07 7E FE 00 C9 982
 16773  CB FC 22 34 40 2A 34 599
#16780  40 7C E6 7F B5 C8 CD 1131
 16787  4F 41 28 F3 06 10 4A 523
#16794  7B E8 21 00 00 CB 39 551
 16801  1F 30 01 19 E6 29 EB 516
#16808  10 F5 C9 3E 28 90 DA 929
 16815  DC 42 47 3E 01 CB 28 563
#16822  30 02 3E 04 CB 29 30 408
 16829  02 CB 07 F5 CD EB 41 962
#16836  7E CB 07 FE 10 30 07 551
 16843  CB 0F 30 02 EE 8F 47 720
#16850  11 9E 0C 3A 30 40 93 504
 16857  FA E1 41 F1 2F A0 18 1012
#16864  02 F1 B0 FE 08 38 02 739
 16871  EE 8F 18 36 3E 17 90 600
#16878  DA DC 42 79 E6 1F 4F 965
 16885  C5 C5 C5 AF CB 10 CB 1186
#16892  10 CB 10 68 67 29 29 524
 16899  C1 48 47 09 C1 47 09 518
#16906  ED 4B 0C 40 09 23 22 466
 16913  0E 40 C1 3E 18 90 32 551
#16920  3A 40 3E 21 91 32 39 469
 16927  40 C9 D5 E5 C5 F5 3A 1207
#16934  3A 40 FE 02 28 3A F1 717
 16941  FE 76 28 2A FE 40 30 820
#16948  72 2A 0E 40 77 23 22 422
 16955  0E 40 3A 39 40 3D 32 368
#16962  39 40 7E FE 76 20 4A 725
 16969  3A 3A 40 3D 32 3A 40 413
#16976  23 22 0E 40 3E 21 32 292
 16983  39 40 18 38 2A 0E 40 321
#16990  7E FE 76 28 E6 23 18 827
 16997  F8 2A 0C 40 01 F8 02 517
#17004  09 36 8F CD 4F 41 28 595
 17011  FB FE 28 28 15 FE 3F 923
#17018  28 1B FE 29 20 08 21 435
 17025  10 27 28 7D B4 20 FB 686
#17032  CD 23 41 18 03 CD 6C 645
 17039  44 F1 18 9A C1 E1 D1 1114
#17046  C9 CD 69 08 18 D3 78 874
 17053  B1 C8 1A CD 21 42 13 726
#17060  0B 18 F5 FE 43 38 09 666
 17067  FE C0 CB B7 DA 35 42 1169
#17074  E6 3F 21 11 01 47 04 419
 17081  FE 21 30 04 AF CD 21 752
#17088  42 CB 7E 23 28 FB 10 737
 17095  F9 7E CB 7F 20 06 CD 948
```

```
*17102 21 42 23 18 F5 E6 3F 595
 17109 CD 21 42 AF C3 2D 42 785
*17116 CF 8A 7B B2 28 FA CD 1141
 17123 29 43 C5 7C B2 07 38 570
*17130 F1 48 42 11 00 00 D5 612
 17137 EB 23 29 EB 29 79 95 857
*17144 78 9C EB 30 F6 EB E8 1275
 17151 AF 7C 1F 67 7D 1F 6F 700
*17158 B4 28 18 E6 AF CB 1C 885
 17165 CB 1D 79 95 78 9C FA 1028
*17172 FE 42 79 95 4F 78 9C 945
 17179 47 E3 19 E3 18 DD E1 1020
*17186 C1 CB 78 C2 1B 41 C9 1003
 17193 44 7C 17 DC 1B 41 EB 762
*17200 7C A8 47 CB 7C C2 1B 911
 17207 41 C9 ED 5B 32 40 63 807
*17214 2E FD 7A B7 06 00 ED 847
 17221 52 98 ED 52 98 5F 50 880
*17228 ED 52 30 01 23 22 32 487
 17235 40 CB B0 C9 03 03 03 555
*17242 C5 03 03 03 03 21 80 370
 17249 49 CD A3 09 23 3E 76 565
*17256 71 23 36 02 23 C1 71 545
 17263 23 70 23 36 EA 23 77 624
*17270 23 77 23 C9 21 00 00 423
 17277 E5 E5 CD 57 41 FE 16 1091
*17284 20 09 E1 D1 F5 E5 CD 1154
 17291 21 42 18 F0 FE 76 E1 960
*17298 28 1F 29 E5 29 29 D1 632
 17305 19 4F CD 21 42 79 D6 743
*17312 1C 38 08 4F 06 00 09 186
 17319 FE 0A 38 D3 F1 3E E6 1054
*17326 CD 21 42 18 C7 F1 CC 972
 17333 1B 41 C9 C5 E5 2A 78 884
*17340 40 F5 77 E5 23 22 78 849
 17347 40 CD 4F 41 FE 29 20 740
*17354 12 3E 76 CD 21 42 E1 727
 17361 CD B4 40 F1 26 00 6F 839
*17368 CD B4 40 18 02 E1 F1 941
 17375 E1 C1 AF C9 C9 C9 3E 1258
*17382 0F D7 CD 75 49 18 12 667
 17389 21 00 00 CD 12 44 CD 529
*17396 F7 48 03 CD 78 49 E5 949
 17403 CD 06 49 E1 CD 12 44 800
*17410 CD 59 49 CD 20 44 CD 877
 17417 0D 44 CF 7F 3E C9 C3 873
*17424 7B 49 22 70 40 ED 5B 734
 17431 1C 40 13 21 72 40 CD 527
*17438 09 45 2A 70 40 36 18 374
 17445 23 36 68 23 22 79 40 447
*17452 11 68 00 19 22 7B 40 357
 17459 21 7D 40 CD D4 44 23 742
*17466 23 E5 23 22 15 40 CD 524
```

```
 17473  44  45  FE  F2  CC  CB  44  1108
*17480  FE  FA  CC  C8  48  FE  DE  1456
 17487  28  EF  FE  EC  CC  2B  45  1085
*17494  FE  ED  CC  40  45  FE  EB  1317
 17501  CC  C7  47  FE  F3  CC  F4  1419
*17508  47  FE  F4  CC  B6  47  18  1050
 17515  16  2A  10  40  0E  18  05  188
*17522  20  2B  36  76  2B  36  00  344
 17529  10  FB  0D  20  F3  C3  F5  995
*17536  08  00  00  FE  F6  CC  BC  820
 17543  48  FE  FC  CC  92  48  FE  1254
*17550  F1  CC  03  46  FE  FB  CC  1227
 17557  9F  47  FE  E3  CA  96  47  1134
*17564  FE  F5  CC  55  45  FE  EA  1345
 17571  28  15  FE  E7  CC  A4  47  985
*17578  FE  FE  CC  0D  44  FE  00  1047
 17585  20  0E  CD  44  45  FE  76  760
*17592  20  07  E1  CD  0E  49  C2  750
 17599  36  44  3E  B8  CD  68  49  750
*17606  CD  59  49  CF  9B  CD  4D  1011
 17613  47  21  5F  49  C3  A9  47  707
*17620  E5  56  23  5E  D5  EB  E5  1121
 17627  22  23  40  22  0A  40  CD  446
*17634  EC  45  CD  53  49  E1  CD  1096
 17641  13  45  18  16  EB  D1  CD  791
*17648  09  45  ED  5B  76  40  CD  798
 17655  09  45  E5  ED  5B  72  40  813
*17662  A7  ED  52  E1  38  03  22  804
 17669  72  40  E1  C9  73  23  72  868
*17676  23  C9  5E  23  56  23  C9  687
 17683  E5  C1  2A  1C  40  23  CD  796
*17690  0E  45  EB  A7  ED  42  D0  995
 17697  2A  72  40  ED  52  D8  EB  990
*17704  23  18  ED  3E  C3  F5  CD  1003
 17711  44  45  CD  2A  48  CD  13  680
*17718  45  EB  CD  0E  45  EB  F1  1068
 17725  C3  AB  47  3E  CD  18  E9  951
*17732  E5  E7  FE  76  28  09  FE  1135
 17739  7E  28  05  F5  CD  68  49  798
*17746  F1  E1  C9  CD  44  45  FE  1263
 17753  76  28  77  FE  0B  28  13  601
*17760  FE  C1  28  4C  FE  D6  28  1071
 17767  74  CD  50  47  21  53  49  661
*17774  CD  A9  47  18  4F  3E  18  634
 17781  CD  78  49  2A  76  40  E5  859
*17788  CD  78  49  01  FF  FF  2A  954
 17795  16  40  23  03  7E  F5  CD  700
*17802  68  49  F1  FE  0B  28  05  728
 17809  CD  78  49  18  EF  22  16  720
*17816  40  E1  C5  71  23  3E  11  713
 17823  CD  AB  47  3E  01  E1  CD  940
*17830  AB  47  21  6B  49  CD  A9  829
 17837  47  18  12  CD  37  47  CD  549
```

```
*17844 C1 47 21 43 4D CD AE 820
 17851 47 21 F5 08 CD A9 47 802
*17858 CD 44 45 FE 1A 28 0C 802
 17865 FE 19 28 08 21 EC 45 793
*17872 CD A9 47 2A 16 40 28 E6 816
 17879 22 16 40 AF C9 CD 4D 778
*17886 47 3E 7D CD 7B 49 21 E9 892
 17893 68 49 CD A9 47 18 D6 860
*17900 3E 76 C3 68 49 FE 40 870
 17907 D2 C1 44 D6 26 17 17 769
*17914 ED 4B 79 40 26 00 6F 646
 17921 09 C9 CD 44 45 CD F1 998
*17928 45 FE 64 20 0A CD 2A 712
 17935 49 3E E5 CD 7B 49 26 B03
*17942 FF E5 CD 44 45 FE 14 1100
 17949 C2 C1 44 CD 3D 46 CD 996
*17956 D3 45 E1 7C FE FF 20 1170
 17963 0C 21 D1 EB CD AE 47 939
*17970 21 09 45 C3 A9 47 3E 608
 17977 22 C3 AB 47 AF F5 CD 1096
*17984 4D 47 3E E5 CD 7B 49 840
 17991 CD 44 45 FE 11 28 50 733
*17998 FE 76 28 4C F5 CD 4D 1015
 18005 47 F1 FE 15 28 04 FE 885
*18012 16 20 23 C1 67 78 FE 759
 18019 00 20 04 F5 E5 18 D8 750
*18026 E5 21 44 4D CD AE 47 857
 18033 CD 7E 46 3E E5 21 69 830
*18040 60 CD AB 47 18 C4 F5 1008
 18047 C5 18 1C FE 18 28 10 583
*18054 FE 17 C2 C1 44 CD C6 1135
 18061 46 21 62 49 CD A9 47 719
*18068 18 AC CD C6 46 21 6E 812
 18075 49 18 F3 3E E1 CD 7B 955
*18082 49 F1 FE 00 C8 FE 15 1043
 18089 28 11 CD C6 46 21 A7 730
*18096 EB CD AE 47 21 ED 52 1037
 18103 CD AE 47 18 E7 CD C6 1108
*18110 46 3E 19 CD 7B 49 18 582
 18117 DD 3E D1 C3 7B 49 FE 1137
*18124 D3 20 0F CD 4D 47 21 644
 18131 7E 6F CD AE 47 21 26 758
*18138 00 C3 AE 47 FE D4 20 938
 18145 12 CD 4D 47 21 EB 45 709
*18152 C3 A9 47 01 F1 46 C5 944
 18159 E5 C9 44 4D C9 FE C4 1226
*18166 C2 0F 47 CD 44 45 FE 876
 18173 41 C2 C1 44 21 5C 49 718
*18180 CD A9 47 3E 26 21 00 578
 18187 6F C3 AB 47 FE 40 C2 1050
*18194 1A 47 21 72 49 C3 A9 681
 18201 47 FE D2 20 11 CD 4D 866
*18208 47 21 C8 7C CD AE 47 881
```

```
 18215  3E  C4  21  7A  47  C3  AB  850
*18222  47  FE  CF  C2  C1  44  C3 1182
 18229  4D  47  CD  4D  47  CD  44  774
*18236  45  F5  3E  E5  CD  7B  49 1006
 18243  CD  4D  47  3E  D1  CD  7B  952
*18250  49  F1  C9  CD  44  45  FE 1111
 18257  15  F5  CC  44  45  FE  10  878
*18264  28  16  FE  40  30  0D  FE  695
 18271  26  D4  82  47  DC  8F  47  885
*18278  F1  CC  75  47  C9  CD  CB 1242
 18285  46  18  F5  CD  3D  46  18  700
*18292  F1  21  7A  47  18  2F  7C  552
 18299  2F  67  7D  2F  6F  23  C9  669
*18306  CD  F1  45  FE  64  CA  3F 1134
 18313  49  3E  2A  C3  AB  47  CD  819
*18320  2A  48  3E  21  18  15  CD  459
 18327  0D  44  E1  ED  4B  7B  40  805
*18334  C9  21  6C  44  18  05  21  472
 18341  56  49  18  00  3E  CD  CD  855
*18348  7B  49  7D  CD  7B  49  7C  846
 18355  C3  7B  49  CD  37  47  CD  927
*18362  C1  47  21  EB  73  18  ED  908
 18369  FE  1A  C2  C1  44  C9  CD 1141
*18376  44  45  CD  F1  45  E5  E5 1110
 18383  CD  44  45  CD  37  47  3E  735
*18390  23  CD  7B  49  21  ED  53  789
 18397  CD  AE  47  E1  CD  AE  47 1125
*18404  3E  22  23  23  CD  AB  47  613
 18411  E1  ED  5B  7B  40  CD  09  954
*18418  45  C9  CD  44  45  CD  F1 1058
 18425  45  E5  E5  3E  2A  CD  AB 1007
*18432  47  21  23  22  CD  AE  47  623
 18439  E1  CD  AE  47  23  23  3E  807
*18446  ED  CD  7B  49  3E  5B  CD  996
 18453  AB  47  3E  A7  21  ED  52  823
*18460  CD  AB  47  3E  FA  E1  CD 1189
 18467  0E  45  E8  CD  AB  47  C9  966
*18474  21  00  00  18  0F  CD  44  345
 18481  45  FE  7E  28  1A  F5  11  777
*18488  0A  00  CD  62  49  F1  FE  881
 18495  28  D2  C1  44  FE  1C  DA 1009
*18502  C1  44  D6  1C  06  00  4F  588
 18509  09  18  DF  E5  2A  16  40  513
*18516  11  05  00  19  22  16  40  167
 18523  E1  C9  21  75  49  CD  A9 1023
*18530  47  CD  44  45  CD  F1  45  928
 18537  C3  38  46  3E  9B  F5  CD  988
*18544  37  47  CD  C1  47  21  4B  703
 18551  45  CD  AE  47  3E  3E  CD  848
*18558  7B  49  F1  CD  7B  49  3E  900
 18565  32  21  30  40  CD  AB  47  642
*18572  21  65  49  C3  A9  47  3E  704
 18579  A0  18  D8  3E  2A  21  34  589
```

```
*18586 40 CD AB 47 3E 22 21 540
 18593 32 40 C3 AB 47 FE DD 1025
*18600 28 12 FE 14 28 14 1F 423
 18607 30 01 EB 17 A7 ED 52 793
*18614 37 F8 17 D0 18 08 ED 803
 18621 52 37 C0 18 04 ED 52 676
*18628 37 C8 3F C9 CD 37 47 850
 18635 67 2E 3E CD AE 47 21 694
*18642 A6 48 CD A9 47 2A 0A 735
 18649 40 23 CD 13 45 EB CD 832
*18656 0E 45 EB 3E D2 CD AB 966
 18663 47 CD 44 45 FE DE C2 1083
*18670 C1 44 C9 21 F7 48 C3 1009
 18677 A9 47 CD E7 02 21 3B 770
*18684 40 00 00 C9 21 06 49 377
 18691 C3 A9 47 21 3B 40 00 591
*18698 00 C3 07 02 4E 23 46 387
 18705 09 23 3A 0C 40 9D C0 527
*18712 3A 0D 40 9C C9 CD 44 765
 18719 45 FE 3F C2 C1 44 CD 1046
*18726 4D 47 AF C9 CD 4D 47 877
 18733 21 ED 4B CD AE 47 21 828
*18740 1C 40 CD AE 47 21 29 616
 18747 09 C3 AE 47 CD 2A 49 769
*18754 3E D5 CD 7B 49 21 0E 723
 18761 45 CD A9 47 21 EB D1 991
*18768 C3 AE 47 C3 88 40 C3 1030
 18775 8B 40 C3 8E 40 C3 91 944
*18782 40 C3 94 40 C3 97 40 881
 18789 C3 9A 40 C3 9D 40 C3 1024
*18796 A0 40 EB C3 A3 40 C3 1076
 18803 A6 40 C3 A9 40 C3 AC 1025
*18810 40 C3 AF 40 AF 00
```

Quem possuir um RINGO R-470 deve, ao fim da digitação, comandar diretamente os seguintes POKES:

```
POKE 17267,249
POKE 17476,234
POKE 17481,252
POKE 17509,231
POKE 17539,254
POKE 17544,250
POKE 17549,240
POKE 17554,253
POKE 17569,249
POKE 17578,239
```

A OPERAÇÃO DO MONITOR

As opções do menu são as seguintes:

1 – GRAVAR — grava tudo o que estiver na memória do micro.

2 – ENTRAR DADOS – pede o endereço inicial de onde deseja introduzir os códigos. Após isso, entre com uma linha de 7 bytes em hexadecimal, deixando um espaço entre cada um, como são mostradas todas as listagens heptasintáticas neste livro. Quando desejar parar de introduzir os códigos no início de um INPUT, digite N. Porém, se você estiver no meio de uma linha, no local do próximo byte (que não existe), digite simplesmente X, e ele retornará ao menu.

3 - CRIAR REM DO COMPILADOR – este comando funciona apenas uma vez, enquanto não existe a linha 0, pois somente uma linha 0 é necessária, e mais de uma iria atrapalhar as compilações.

4 - APAGAR LINHAS – uma das rotinas mais úteis do monitor: apaga blocos de linhas. Será útil quando você acabar de digitar e compilar os programas e quiser deletar o BASIC e ficar somente com o chamado "código objeto" (que está na linha 2) lembre-se que o compilador tem que estar na linha 0.

5 - VERIFICAR DADOS – nada mais que o contrário do modo 2. Ao entrar neste modo, ele pede o endereço inicial e mostra uma linha com a soma sintática correspondente. Ao pressionar N/L, ele mostra a próxima linha, qualquer outra, volta ao menu.

A qualquer momento você poderá sair do monitor com BREAK e retornar a ele com GOTO 9580.

APÊNDICE 2

Este apêndice será alvo de muitas consultas, pois irá demonstrar como usar uma grande parte das poderosas funções do compilador.

Primeiramente, trataremos das variáveis do compilador. Elas não são acessáveis pelo Basic externo, isto é, aquele que não é compilador, pela instrução LET. O compilador guarda suas variáveis de A a Y numa área reservada na linha 2 REM entre os endereços 18825 e 18922, que pode ser acessada por PEEK e POKE. Os números são armazenados em dois bytes na forma LSB (Less Significative Byte) e MSB (Most Significative Byte) nesta ordem.

Para pesquisar as variáveis, você deve proceder da seguinte maneira:

```
PRINT PEEK (LSB) + 256 * PEEK (MSB)
```

Para mudá-las:

```
POKE LSB, n-256 * INT (n/256)
POKE MSB, INT (n/256)
```

onde n é um número entre Ø e 65535.

Note também, que estes dois procedimentos valem para as variáveis do sistema, que serão explicadas mais à frente.

TABELA DAS VARIAVEIS

NOME	LSB	MSB
A	18825	18826
B	18829	18830
C	18833	18834
D	18837	18838
E	18841	18842
F	18845	18846
G	18849	18850
H	18853	18854
I	18857	18858
J	18861	18862
K	18865	18866
L	18869	18870
M	18873	18874
N	18877	18878
O	18881	18882
P	18885	18886
Q	18889	18890
R	18893	18894
S	18897	18898
T	18901	18902
U	18905	18906
V	18909	18910
W	18913	18914
X	18917	18918
Y	18921	18922

Quanto à variável Z, ela é a única variável indexada permitida pelo compilador. Porém seus conteúdos não são guardados numa área reservada para isto, eles são armazenados numa área entre o STKEND e o stack de máquina, que em Assembly é fixo (enquanto não se usam as rotinas de cálculo com ponto flutuante) e em Basic é móvel, cobrindo o valor das variáveis. Portanto, não trataremos aqui como pesquisar e alterar as variáveis indexadas.

Variáveis do sistema:

"Variáveis do sistema" é nome dado a uma série de informações manipuladas pelo sistema operacional, e por este motivo não podem ficar guardadas em ROM (Read Only Memory).

Para este fim, o sistema toma parte da memória RAM (Random Access Memory) para guardá-las, mais exatamente dos endereços 16384 a 16508, inclusive.

Suas localizações, nome e significados vêm dados a seguir:

DEC.	*HEX.*	MNEM.	EXPLICAÇÃO
16384	*4000*	ERR-NR	Esta variável guarda o código de erro −1, isto é, o caractere do código de erro é incrementado antes de ser impresso.
16385	*4001*	FLAGS	Bit Ø – indica a supressão do espaço lateral das palavras do Basic. Bit 1 – controla a impressora. Bit 2 – controla cursores K ou G e L ou F. Bit 6 – controla os parâmetros de strings ou ponto flutuante. Bit 7 – controla o check de sintaxe.
16386	*4002* *4003*	ERR-SP	Guarda o endereço do stack do GOSUB.
16388	*4004* *4005*	RAMTOP	Marca o topo da RAM acessável pelo Basic.
16390	*4006*	MODE	Guarda o valor que especifica o cursor K e L ou F e G.
16391	*4007* *4008*	PPC	Contém o número da linha que está sendo executada.
16393	*4009*	VERSN	Marca o começo da RAM que será salva.
16394	*400A* *400B*	E-PPC	Contém o número da linha marcada pelo cursor.
16396	*400C* *400D*	D-FILE	Endereço do início do arquivo de imagem (primeiro Newline).
16398	*400E* *400F*	DF-CC	Endereço do PRINT dentro do D-FILE.
16400	*4010* *4011*	VARS	Marca o início da área de variáveis.
16402	*4012* *4013*	DEST	Endereço da última variável manipulada pelo Basic.
16404	*4014* *4015*	E-Line	Marca o endereço do rascunho de trabalho.
16406	*4016* *4017*	CH-ADD	Contém o endereço do próximo caractere a ser interpretado pelo Basic.

16408	*4018* *4019*	X-PTR	Endereço do caracter precedendo o cursor S.
16410	*401A* *401B*	STKBOT	Endereço do início do STACK do calculador.
16412	*401C* *401D*	STKEND	Endereço do final do STACK do calculador.
16414	*401E*	BERG	Várias utilidades.
16415	*401F* *4020*	MEM	Endereço da área usada no cálculo de ponto flutuante (MEMBOT-16477 a 16506)
16417	*4021*	-----	Não usada.
16418	*4022*	DF-SZ	Número de linhas reservadas na parte inferior da tela.
16419	*4023* *4024*	S-TOP	Número da linha do topo de tela na listagem automática.
16421	*4025* *4026*	LAST-K	Valor binário do estado do teclado após a última leitura.
16423	*4027*	DEBOUNCE	Estado de DEBOUNCE do teclado.
16424	*4028*	MARGIN	Controle de linhas (de um bit de altura) geradas pelo software de vídeo.
16425	*4029* *402A*	NXTLIN	Endereço da próxima linha Basic a ser interpretada.
16427	*402B* *402C*	OLDPPC	Número da linha para a qual o CONT salta.
16429	*402D*	FLAGX	Bit Ø — quando Ø, indica ARRAY Bit 1 — quando Ø, indica que a variável procurada existe. Bit 5 — durante INPUT assume valor 1. Bit 6 — quando 1, indica INPUT numérico.
16430	*402E* *402F*	STRLEN	Comprimento da variável string.
16432	*4030* *4031*	T-ADDR	Endereço da tabela de parâmetros. Também usada p/ distinguir PLOT de UNPLOT.
16434	*4032* *4033*	SEED	Guarda o valor RAND que dará origem ao RND.
16436	*4034* *4035*	FRAMES	Conta os quadros mandados para a televisão. Usada na temporização do PAUSE.
16438	*4036* *4037*	COORDS	Coordenadas X e Y do PLOT e UNPLOT.

16440	*4038*	PR-CC	Contador da impressora, contém o LSB do endereço no PRBUFF.
16441	*4039*	S-POSN	Respectivamente a coluna e a linha do PRINT AT, porém contados inversamente: a posição 1,2 corresponde a última casa do D-FILE (canto inferior direito) e a posição 24,33 corresponde à primeira casa do D-FILE.
	403A		
16443	*403B*	CDFLAG	Bit Ø — assume valor 1 sempre que uma tecla é pressionada. Bit 6 — controle FAST/SLOW. Bit 7 — copia do bit 6.
16444	*403C*	PRBUFF	Buffer da impressora. As linhas são armazenadas aqui até serem completadas e mandadas para a impressora.
⋮	⋮		
16476	*405C*		
16477	*405D*	MEMBOT	Área de memória que pode armazenar até seis números em ponto flutuante. Suas locações recebem o nome de MEM-0 até MEM-5.
⋮	⋮		
16506	*407A*		
16507	*407B*	———————	Não usada.
	407C		
16509	*407D*	PROGRAM	Início da área do Basic.

Como usar o D-FILE:

O D-FILE é uma área de memória RAM, onde ficam guardados os caracteres a serem gerados pelo software de vídeo e mandados para a tela da televisão.

Normalmente, quando queremos imprimir algo na tela, usamos a instrução PRINT. Vamos agora analisar o que o sistema operacional faz quando recebe uma linha PRINT:

```
10 PRINT AT 10,12;"D-FILE"
```

Primeiramente, o software verifica os parâmetros do AT, isto é, se o local pedido permite a impressão, senão fornece o código de erro 5/10.

Posteriormente, o sistema faz certos cálculos para localizar a posição correta da impressão. Após isto, começa seqüencialmente a colocar a partir daquela posição os caracteres da "string" dada. Como o leitor pode notar,

é um processo complicado e demorado, mesmo se compilado. Existe, porém, uma solução.

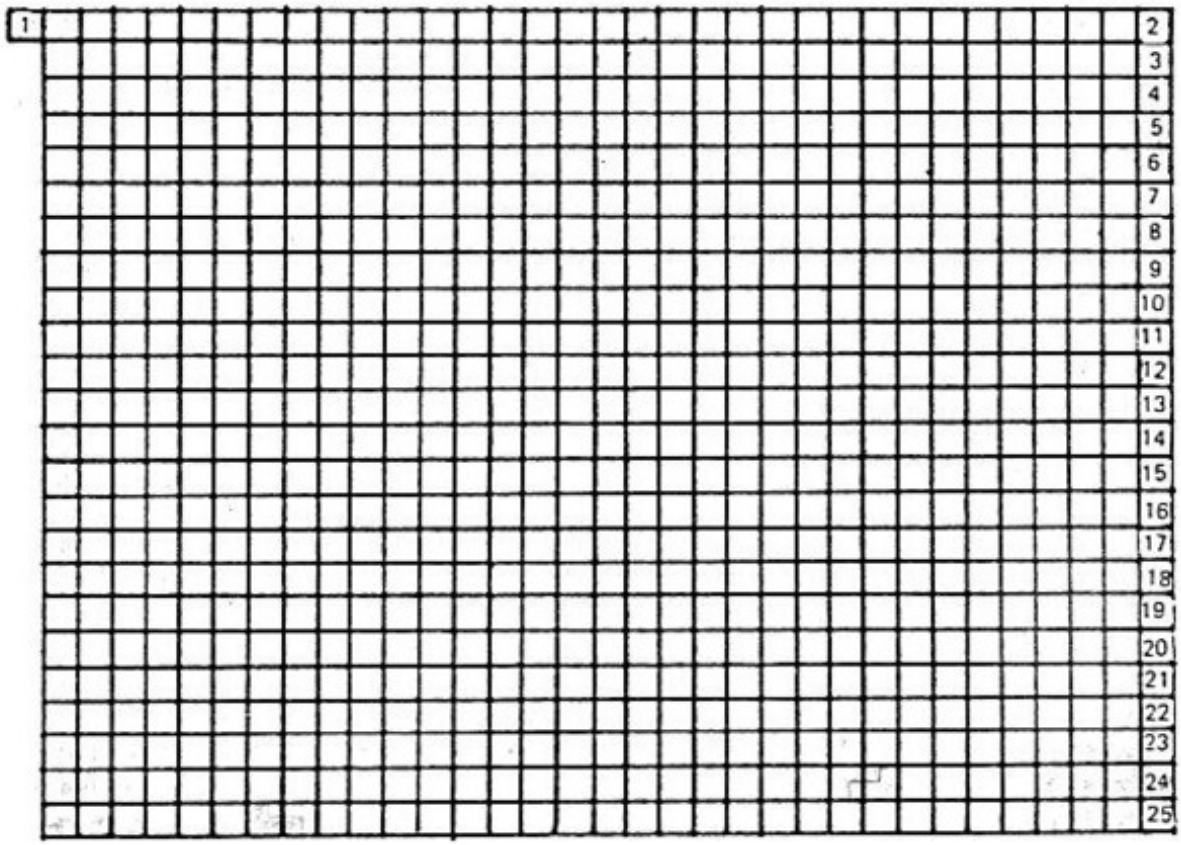
O que é o D-FILE:

Numa primeira análise, o que vemos na tela da televisão são 24 linhas de 32 caracteres cada uma, o que dá o total de 768 posições de tela. Porém, pensando bem, o D-FILE gasta 793 bytes. Aonde e o que são os 25 bytes restantes?

Resposta: cada linha de tela contém 33 colunas!

A organização do D-FILE conta com 32 colunas visíveis e uma coluna que corresponde ao "retorno de carro" (CR), isto é, informa ao hardware do vídeo que deve passar para a linha de baixo.

Para você compreender melhor este aspecto, segue um mapa completo do D-FILE.



A casa numerada com 1 corresponde ao caractere N/L (código 118 ou 76H) e é o primeiro byte invisível do D-FILE.

A variável do sistema do D-FILE guarda o endereço deste N/L e há mais 24 newlines a cada 33 bytes.

Após compreender como funciona o D-FILE é fácil entender porque é tão demorada a instrução PRINT. A solução para a perda de tempo é usar a instrução POKE, porém teremos antes de "POKEar" no D-FILE, fazer o mesmo que o sistema operacional faz; ou seja: achar sua posição correta e verificar se é um caracter permitido, entre Ø e 63 ou 128 a 191, pois senão o sistema irá pelos ares.

Aviso aos navegantes: nunca "POKEie" nos "new lines" do D-FILE, pois se você estragar a "moldura" formada pelos newlines, talvez nunca mais veja o seu programa.

A parte mais complicada realmente é o cálculo da posição do POKE. É o que veremos mais detalhadamente daqui para a frente.

Como calcular a posição correta:

Tomemos como exemplo:

```
10 PRINT AT 10,15;"*";
```

Faça o seguinte cálculo:

1 — pegue o número da coluna e some 1.
2 — pegue o número da linha e multiplique por 33.
3 — some os números obtidos.
4 — o número obtido em 3 deve ser somado ao endereço do D-FILE.

E através de 4 simples operações está achado o endereço onde deve ser POKEado o número 23, que é o código do asterisco:

```
  10 LET A=PEEK 16396+256*PEEK 1
6397
  20 POKE (A+346),136
```

O uso aprimorado deste cálculo será mais do que mostrado em todos os jogos deste livro, que formam um exemplo completo.

Tabela de caracteres:

Esta tabela será muito útil quando você precisar converter PRINTs para POKEs.

DEC	HEX	CARACTER	POKAVEL
0	00		SIM
1	01		SIM
2	02		SIM
3	03		SIM
4	04		SIM
5	05		SIM
6	06		SIM
7	07		SIM
8	08		SIM
9	09		SIM
10	0A		SIM
11	0B	"	SIM
12	0C	£	SIM
13	0D	$	SIM
14	0E	:	SIM
15	0F	?	SIM
16	10	(	SIM
17	11	)	SIM
18	12	>	SIM
19	13	<	SIM
20	14	=	SIM
21	15	+	SIM
22	16	-	SIM
23	17	*	SIM
24	18	/	SIM
25	19	;	SIM
26	1A	,	SIM
27	1B	.	SIM
28	1C	0	SIM
29	1D	1	SIM
30	1E	2	SIM
31	1F	3	SIM
32	20	4	SIM
33	21	5	SIM
34	22	6	SIM
35	23	7	SIM
36	24	8	SIM
37	25	9	SIM
38	26	A	SIM
39	27	B	SIM
40	28	C	SIM
41	29	D	SIM
42	2A	E	SIM
43	2B	F	SIM
44	2C	G	SIM
45	2D	H	SIM
46	2E	I	SIM
47	2F	J	SIM

DEC	HEX	CARACTER	POKAVEL
48	30	K	SIM
49	31	L	SIM
50	32	M	SIM
51	33	N	SIM
52	34	O	SIM
53	35	P	SIM
54	36	Q	SIM
55	37	R	SIM
56	38	S	SIM
57	39	T	SIM
58	3A	U	SIM
59	3B	V	SIM
60	3C	W	SIM
61	3D	X	SIM
62	3E	Y	SIM
63	3F	Z	SIM
64	40	RND	NAO
65	41	INKEY$	NAO
66	42	PI	NAO
67	43	?	NAO
68	44	?	NAO
69	45	?	NAO
70	46	?	NAO
71	47	?	NAO
72	48	?	NAO
73	49	?	NAO
74	4A	?	NAO
75	4B	?	NAO
76	4C	?	NAO
77	4D	?	NAO
78	4E	?	NAO
79	4F	?	NAO
80	50	?	NAO
81	51	?	NAO
82	52	?	NAO
83	53	?	NAO
84	54	?	NAO
85	55	?	NAO
86	56	?	NAO
87	57	?	NAO
88	58	?	NAO
89	59	?	NAO
90	5A	?	NAO
91	5B	?	NAO
92	5C	?	NAO
93	5D	?	NAO
94	5E	?	NAO
95	5F	?	NAO

DEC	HEX	CARACTER	POKAVEL
96	60	?	NAO
97	61	?	NAO
98	62	?	NAO
99	63	?	NAO
100	64	?	NAO
101	65	?	NAO
102	66	?	NAO
103	67	?	NAO
104	68	?	NAO
105	69	?	NAO
106	6A	?	NAO
107	6B	?	NAO
108	6C	?	NAO
109	6D	?	NAO
110	6E	?	NAO
111	6F	?	NAO
112	70	?	NAO
113	71	?	NAO
114	72	?	NAO
115	73	?	NAO
116	74	?	NAO
117	75	?	NAO
118	76	?	NAO
119	77	?	NAO
120	78	?	NAO
121	79	?	NAO
122	7A	?	NAO
123	7B	?	NAO
124	7C	?	NAO
125	7D	?	NAO
126	7E	?	NAO
127	7F	?	SIM
128	80	?	SIM
129	81	?	SIM
130	82	?	SIM
131	83	?	SIM
132	84	?	SIM
133	85	?	SIM
134	86	?	SIM
135	87	?	SIM
136	88	?	SIM
137	89	?	SIM
138	8A	?	SIM
139	8B	?	SIM
140	8C	?	SIM
141	8D	?	SIM
142	8E	?	SIM
143	8F	?	SIM

DEC	HEX	CARACTER	POKAVEL
144	90		SIM
145	91		SIM
146	92		SIM
147	93		SIM
148	94		SIM
149	95		SIM
150	96		SIM
151	97		SIM
152	98		SIM
153	99		SIM
154	9A		SIM
155	9B		SIM
156	9C		SIM
157	9D		SIM
158	9E		SIM
159	9F		SIM
160	A0		SIM
161	A1		SIM
162	A2		SIM
163	A3		SIM
164	A4		SIM
165	A5		SIM
166	A6		SIM
167	A7		SIM
168	A8		SIM
169	A9		SIM
170	AA		SIM
171	AB		SIM
172	AC		SIM
173	AD		SIM
174	AE		SIM
175	AF		SIM
176	B0		SIM
177	B1		SIM
178	B2		SIM
179	B3		SIM
180	B4		SIM
181	B5		SIM
182	B6		SIM
183	B7		SIM
184	B8		SIM
185	B9		SIM
186	BA		SIM
187	BB		SIM
188	BC		SIM
189	BD		SIM
190	BE		SIM
191	BF		SIM

DEC	HEX	CARACTER	POKAVEL
192	C0	"	NAO
193	C1	AT	NAO
194	C2	TAB	NAO
195	C3	?	NAO
196	C4	CODE	NAO
197	C5	VAL	NAO
198	C6	LEN	NAO
199	C7	SIN	NAO
200	C8	COS	NAO
201	C9	TAN	NAO
202	CA	ASN	NAO
203	CB	ACS	NAO
204	CC	ATN	NAO
205	CD	LN	NAO
206	CE	EXP	NAO
207	CF	INT	NAO
208	D0	SQR	NAO
209	D1	SGN	NAO
210	D2	ABS	NAO
211	D3	PEEK	NAO
212	D4	USR	NAO
213	D5	STR$	NAO
214	D6	CHR$	NAO
215	D7	NOT	NAO
216	D8	**	NAO
217	D9	OR	NAO
218	DA	AND	NAO
219	DB	<=	NAO
220	DC	>=	NAO
221	DD	<>	NAO
222	DE	THEN	NAO
223	DF	TO	NAO
224	E0	STEP	NAO
225	E1	LPRINT	NAO
226	E2	LLIST	NAO
227	E3	STOP	NAO
228	E4	SLOW	NAO
229	E5	FAST	NAO
230	E6	NEW	NAO
231	E7	SCROLL	NAO
232	E8	CONT	NAO
233	E9	DIM	NAO
234	EA	REM	NAO
235	EB	FOR	NAO
236	EC	GOTO	NAO
237	ED	GOSUB	NAO
238	EE	INPUT	NAO
239	EF	LOAD	NAO

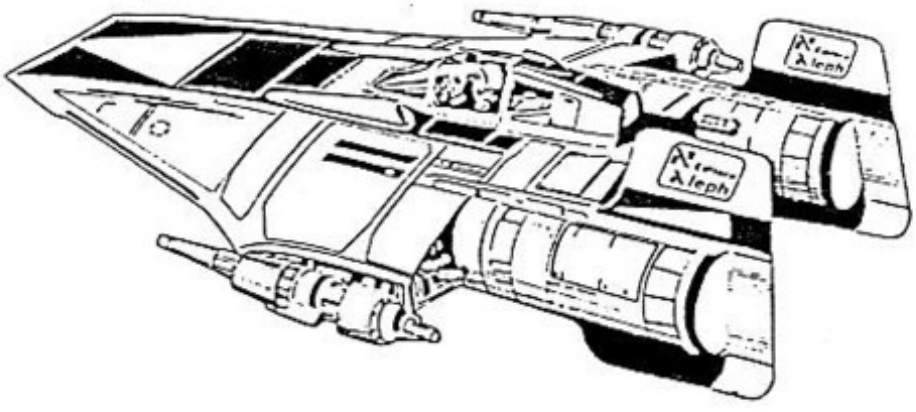
DEC	HEX	CARACTER	POKAVEL
240	F0	LIST	NAO
241	F1	LET	NAO
242	F2	PAUSE	NAO
243	F3	NEXT	NAO
244	F4	POKE	NAO
245	F5	PRINT	NAO
246	F6	PLOT	NAO
247	F7	RUN	NAO
248	F8	SAVE	NAO
249	F9	RAND	NAO
250	FA	IF	NAO
251	FB	CLS	NAO
252	FC	UNPLOT	NAO
253	FD	CLEAR	NAO
254	FE	RETURN	NAO
255	FF	COPY	NAO

DESPEDIDA

É nesta hora que nós, os autores, agradecemos por terem nos agüentado durante todo este livro, apesar das piadinhas e teimosia de nossa parte em insistir em algum ponto ou colocação.

Apesar de tudo, foi uma experiência gratificante escrever esta obra, mostrando que ensaios e experiências de algum tempo atrás se transformaram na única literatura sobre jogos compilados no mercado nacional, e quem sabe, no mundo.

Tudo o que temos a dizer é: muito obrigado, e, prepare-se para o volume 2 (isto é uma ameaça!)



ADENDO

Estamos nestas páginas, talvez, falando sobre o sonho de vários usuários de TK e compatíveis: os acessórios e incrementos.

Nossos micros (dos autores) têm a seguinte configuração mínima:

- 12 Kbytes de ROM, que proporcionam um basic mais rápido e poderoso.
- 4 Kbytes de RAM auxiliar: esta área de RAM, dos endereços 12288 até 16383, é útil para quase tudo, em especial para utilização de alta resolução, com redefinição de 128 caracteres.
- 18 Kbytes de RAM basic: para programas um pouco mais complexos que não caberiam normalmente na memória de 16K.
- Gerador de som: módulo que utiliza o processador de som AY-3-8912 de 3 canais, com capacidade de gerar sons dos mais complexos.
- Teclado semi profissional: proporciona uma digitação muito mais rápida de programas basic e Assembly.

Da combinação de alguns desses incrementos, surgiu uma tela gráfica, onde se pode manipular qualquer dos 49152 pontos de alta resolução gráfica.

De toda essa parafernália de hardware surgiram condições favoráveis para um alto desenvolvimento de software.

Todos os programas deste livro têm a sua respectiva versão em alta resolução e sonorizada, o que dá outro clima aos ataques de vídeo-maníacos que todo programador tem.

O que estamos propondo é que todos escrevam para:

ALEPH Publicações e Assessoria Pedagógica Ltda.
Av. Brig. Faria Lima, 1451 cj. 31
01451 São Paulo SP

aos cuidados dos autores, dando suas idéias, opiniões, e, exprimindo seus desejos sobre tudo o que foi escrito nestas páginas, para que nós, na medida do possível, possamos atender a todos, preenchendo uma vazio no mercado nacional.

Para receber gratuitamente o boletim RAND USR contendo programas, dicas, lançamentos e lista de preços dos livros já publicados, envie seu nome e endereço completos para:

ALEPH PUBL. E ASS. PEDAG. LTDA.
BOLETIM RAND USR
Av. Brig. Faria Lima, 1451 - conj. 31
01451 - São Paulo - SP
TEL. (011) 813-4555

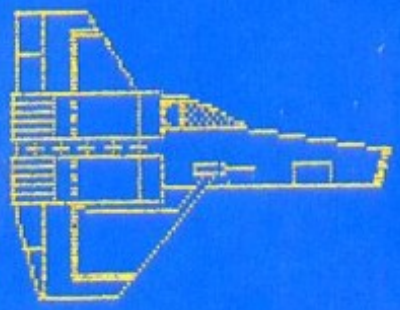
Os originais deste livro foram digitados e arquivados em disco num micro-computador UNITRON ap II com teclado inteligente utilizando o editor de texto JANELA MAGICA 2 acoplado a um monitor monocromático da VIDEOCOMPO. Os originais para revisão foram produzidos numa impressora MONICA PLUS fabricada pela ELEBRA.

unitron
Computadores
elebra informática

VIDEOCOMPO

Este livro foi impresso na
Gráfica Palas Athena
Associação "Palas Athena" do Brasil
Rua Dona Ana Nery, 846
Fone: 279-6288 — CEP 01522
Cambuci — São Paulo

SUPER BASIC TK



Apesar do BASIC ser uma das linguagens de mais fácil aprendizado, tem a desvantagem da lentidão, pois cada linha é traduzida para a linguagem de máquina durante a execução do programa.
Neste volume os autores fornecem um programa COMPILADOR que traduz todas as instruções de uma vez, permitindo rodar o programa inteiro usando a incrível rapidez do ASSEMBLY. Este BASIC "compilável", porém, tem algumas limitações quando comparado com o usado normalmente. Neste volume os autores mostram, através de numerosos e interessantes exemplos, como superar estes inconvenientes montando jogos incríveis e explicando detalhadamente todos os truques empregados. Esta é uma obra importantíssima para quem quer elaborar programas rápidos sem ter que passar pelo árduo aprendizado de uma linguagem árida como o ASSEMBLY.

Editora Aleph

EDITORA MODERNA

Os autores Fernando da Costa Grossi, cursando Processamento de Dados na Universidade Mackenzie, e Milton Maldonado Jr., aluno da Escola Politécnica da USP são dois jovens apaixonados pelos pequenos computadores da linha SINCLAIR. Usando muita criatividade e inteligência elaboraram diversos projetos de "hardware" (o TK 82-C do Fernando já não cabe mais na caixa original devido ao acréscimo de várias "camadas" de circuitos adicionais!) e desenvolveram vários programas em BASIC COMPILÁVEL (SUPER BASIC TK) que estão apresentando neste livro de maneira muito didática e divertida.