

USANDO LINGUAGEM DE MÁQUINA

APLICAÇÕES EM ASSEMBLY Z80

MARIO SCHAEFFER

Para:
SINCLAIR
ZX-81
TS-1000
RINGO R-470
APPLY-300
CP 200
TK 82 C
AS 1000
TK 8
TK 8


EDITORA
MODERNA

URANIA

**USANDO
LINGUAGEM
DE MÁQUINA**
APLICAÇÕES EM ASSEMBLY Z80

MARIO SCHAEFFER



Este livro foi editado pela
Urania Publicações
e **Assessoria Pedagógica Ltda.**

Os programas
foram digitados e testados
no Núcleo de Orientação de Estudos
sob a coordenação
da professora Betty Fromer Piazzi.

Avaliação e Revisão Técnica:
Pierluigi Piazzi e Nancy Mitie Ariga
Arte e Capa: Cassiano Roda
Pastup: Fátima Rossini e Osmère Sarkis

Todos direitos reservados

URANIA

Publicações e Assessoria Pedagógica Ltda.

PREFÁCIO

Como toda pessoa que sonhou (e sonha...) viajar pelo espaço, fico maravilhado quando, nos raros momentos que o ritmo alucinante dos nossos dias me permite, paro para meditar e admirar os fantásticos avanços da arte tecnológica... Os passos gigantescos e velozes que estamos dando nos colocam cada vez mais perto de um mundo até ontem pertencente à fantasia da ficção científica e que, aparentemente, deveria enobrecer a vida dos homens e tornar o mundo mais justo, um lugar melhor de se viver.

Um fato que obrigatoriamente deve estar presente para que este "sonho" aconteça, é que o conhecimento atinja todos os níveis sócio-econômicos da humanidade... a distribuição piramidal do conhecimento deve achatarse cada vez mais! E, felizmente no mundo da Informática, o carro-chefe desse turbilhão de inovações, apareceu um "pequeno" computador, do qual este livro explora mais uma de suas *infinitas* potencialidades... É impressionante notar como um computador de fácil acesso se espalhou pelo mundo e deu asas à imaginação de *milhões* de pessoas... E aqui, a magia da linguagem de máquina é apresentada com exemplos aplicativos de rara beleza possibilitando a excitação da imaginação, abrindo muitas portas novas que conduzem a caminhos de evolução sem fim, que apenas esperam por alguém para explorá-las...

Enfim, parece que temos na mão a "revolução"... No entanto, deixo no ar perguntas para as quais ainda não encontrei uma resposta: por que os momentos de meditação são tão raros e sufocados por essa enorme roda viva? Por que esquecemos tão facilmente que estamos num pequeno barquinho a navegar pelo espaço com um limitado reservatório de água que continuamos a estragar? Por que as manchetes de jornais, retratos de nossa vida, são sempre as mesmas histórias de terror, tristeza e Injustiça? Já li que a resposta é o conhecimento... Já li que a resposta é o amor... Eu considero a tecnologia uma arte, e arte só existe com

Estas linhas, as últimas a serem escritas neste livro, são para agradecer ao Pierluigi, pela idéia do livro e pelo apoio e sugestões dados durante sua execução; à minha mulher, Márcia, pela infinita paciência e pelos conselhos quanto à forma, bem como pela datilografia inicial; aos meus filhos, André e Eduardo, pelo estímulo.

Mario

INTRODUÇÃO

Este livro destina-se àqueles leitores interessados em explorar o seu micro-computador através do uso da linguagem de máquina. Presume-se que o leitor já conheça as instruções de máquina do microprocessador Z-80 e que tenha um micro-computador compatível com o ZX-81 da SINCLAIR, ou seja, RINGO, NEZ-8000, TK-82, TK-83, TK-85, CP-2000, TIMEX 1000, etc.

Há uma vasta bibliografia sobre essa classe de máquina, mas muito pouca coisa foi escrita de forma a efetivamente conduzir o interessado do simples conhecimento das instruções do Z-80 ao uso prático das mesmas. Assim sendo, procuramos apresentar a matéria partindo do mais simples programa possível até um ponto que dará ao leitor a necessária auto-confiança para, então, desenvolver seus próprios programas.

Praticamente todos os programas apresentados neste livro podem ser rodados com 1K de memória RAM, e em nenhum caso serão necessários mais do que 2K.

1.ª PARTE

	Pág.
Capítulo 1 — Os primeiros passos	11
Capítulo 2 — O início	13
Capítulo 3 — Idéias para montar um programa	23
Capítulo 4 — Utilizando as variáveis do sistema	27
Capítulo 5 — Usando a ROM	37
Capítulo 6 — Calculando em linguagem de Máquina	53
Capítulo 7 — As Sub-rotinas de Cálculo	67

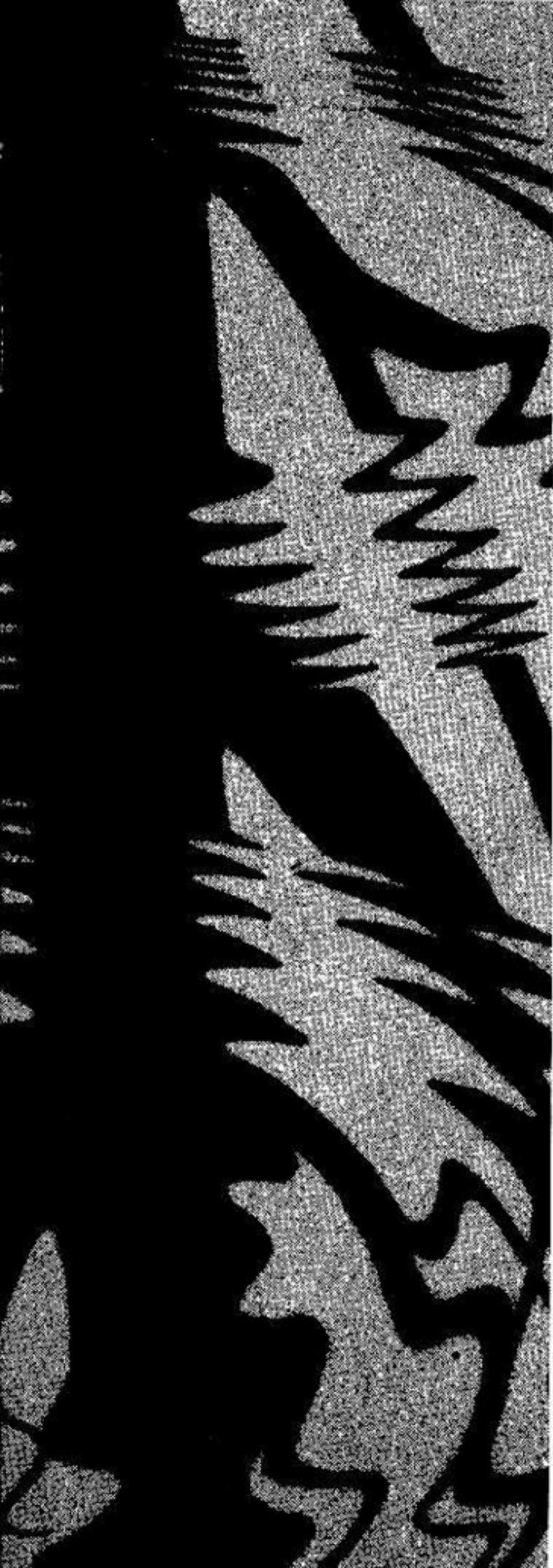
2.ª PARTE

Capítulo 8 — Mestre-2	85
Capítulo 9 — Jogo dos Palitos	95
Capítulo 10 — Um Plot Rápido	107
Capítulo 11 — Caleidoscópio	115
Capítulo 12 — Um jogo Autoprogramável	123

APÊNDICES

Apêndice A — Como criar uma Linha REM	133
Apêndice B — Tabela de Caracteres	136
Apêndice C — Tabela de leitura do Teclado	142





1ª PARTE
DOS
PRIMEIROS
PASSOS
ÀS SUB-ROTINAS
DE CÁLCULO

OS PRIMEIROS PASSOS

1

Toda pessoa que resolve estudar linguagem de máquina esbarra inicialmente no terrível dilema: “Devo ou não ir adiante?”, tamanha é a complexidade e hermeticidade dessa linguagem quando comparada ao BASIC. Realmente, o homem vem, desde o primeiro computador, criando linguagens de nível cada vez mais alto, e já criou centenas delas, justamente para que o usuário não precisasse entender tão profundamente a máquina e as firmas pudessem operar seus computadores com pessoal menos especializado e, portanto, mais barato.

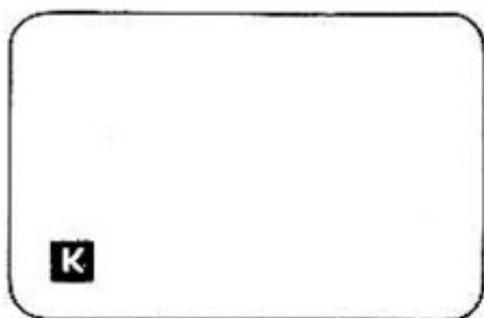
O BASIC é um belo exemplo dessa tendência. No entanto — e aqui procuramos dar uma boa dose de estímulo ao leitor ainda hesitante —, no caso de microcomputadores que são normalmente de uso pessoal, a linguagem de máquina oferece vários atrativos, tais como:

- aumenta enormemente a velocidade de processamento, permitindo, assim, a realização de programas impossíveis de serem rodados em linguagens de alto nível;
- economiza memória, fator importante principalmente em máquinas do porte dessa que estamos abordando;
- finalmente, para aqueles que sentem um sabor de “hobby” ao utilizar o seu micro, a linguagem de máquina é muito mais positiva e dá uma sensação de domínio maior da máquina, ou, em outras palavras, é mais “gostosa” de ser usada.

Ao se aprender uma língua, passa-se normalmente por uma fase inicial mais teórica, que é a da gramática, com todos os problemas de sintaxe, caso, gênero, número, etc. (que, em nosso caso corresponde exatamente ao repertório de instruções do Z-80). Depois, fazem-se os primeiros exercícios e uso efetivo da língua e, finalmente, ao dominá-la, começa-se a pensar nesta língua. No caso de linguagem de computador ocorre um fenômeno muito semelhante inicialmente, estudam-se as áridas instruções da linguagem de máquina; depois, procura-se adaptar a solução da linguagem que conhecemos para a de máquina, até que, finalmente, ao se abordar um problema novo, já se começa a buscar uma solução pensando nos recursos da linguagem de máquina. Pretendemos, pois, que o leitor, razoável conhecedor de “gramática”, percorra conosco este livro e, ao final, nos abandone já “pensando” em linguagem de máquina.

INÍCIO 2

O Sinclair, como quase todos os micros pessoais, foi criado para rodar programas em BASIC. Dessa forma, ao se ligar a máquina, ela entra no programa armazenado em sua ROM e percorre uma série de sub-rotinas de iniciação e verificação do tamanho da memória e, ao final, quando coloca o cursor à nossa disposição, já está em BASIC. Assim, para irmos para a linguagem de máquina, temos que usar — e não existe outra forma — a função `USR X`, onde `X`, um valor entre 0 e 65535, é o endereço inicial do programa em linguagem de máquina.



Quando o cursor é colocado à disposição, o computador já está esperando o BASIC

Existem várias maneiras de se armazenar programas em linguagem de máquina na RAM, cada uma apresentando suas particularidades. Entre as várias alternativas, optamos pelo uso da instrução `REM`. Assim sendo, todos os programas em linguagem de máquina apresentados neste livro estão colocados na primeira linha do programa BASIC, que é uma instrução `REM`. Por conseguinte o leitor, ao fazer um programa em linguagem de máquina, deve calcular quantos “bytes” o programa ocupa e colocar, como primeira instrução do programa (não importa o número dessa linha), a instrução `REM` seguida de no mínimo tantos caracteres (qualquer caracter pode ser usado para esse fim) quantos são os “bytes” do programa na linguagem de

O passo seguinte é escrever nessa instrução REM o programa. Ora, como o endereço na RAM do início da área de programa é 16509, se se colocar como primeira instrução REM, teremos a seguinte ocupação para os primeiros "bytes" do programa em BASIC:

16509	} número da linha
16510	
16511	} comprimento da instrução
16512	
16513	código da instrução (no caso de REM será 234)
16514	primeiro "byte" do programa em linguagem de máquina

Para carregar um programa escrito em linguagem de máquina nessa instrução REM recomendamos o programa em BASIC apresentado a seguir. Se o leitor tiver expansão, poderá optar por programas bem mais sofisticados, apresentados na bibliografia. Recomendamos a gravação desse programa em fita, pois ele terá que ser usado toda vez que se tiver que carregar um programa em linguagem de máquina.

```

1 REM 00000000000000000000000000000000
000000000000000000000000000000000000
0000000000
8990 STOP
9000 INPUT A$
9010 FOR J=1 TO LEN A$-1 STEP 2
9020 POKE 16514+INT (J/2) ,16*CODE
E A$(J)+CODE A$(J+1)-476
9030 NEXT J
9040 STOP
9900 SAVE "MESTRE"
9910 STOP

```

A instrução do REM está colocada na linha 0 por duas razões: para garantir ser a primeira linha do programa BASIC e evitar sua edição e possível apagamento. Para colocar 0 no número dessa linha, entre inicialmente com a instrução na linha 1 e faça, depois, POKE 16510,0. A instrução STOP em 8990 isola esse programa dos que venham a ser criados. A instrução em 9000 faz entrar uma "STRING", que é o programa em linguagem de máquina escrito em hexadecimal. Como para cada "byte" do programa temos dois dígitos hexadecimais — um, o mais significativo, numa posição ímpar da "STRING" e outro, menos significativo, numa posição par

fazemos, na instrução 9010, o "J" apontar sempre para o dígito mais significativo de cada "byte" do programa e convertemos o código de hexadecimal em decimal, pois é assim que a máquina exige que esteja o dado a armazenar na instrução "POKE". Como os códigos dos caracteres de 0 a F são seqüências e começam em 28, temos que subtrair 476 para obter o valor decimal do "byte", ou seja, temos que descontar $16 \times 28 + 28 = 476$ (veja apêndice B).

Bem, agora estamos preparados para começar a escrever programas. Como já dissemos, a função `USR` é a que nos transporta do BASIC para a linguagem de máquina. Quando o interpretador BASIC se depara com essa função, ele pega o argumento da mesma e coloca no par de registradores BC, pulando, a seguir, para o endereço indicado por esse argumento.

De outro lado, para voltarmos para o BASIC, temos que usar uma das instruções de `RETurn` (`RET`). Mais tarde veremos que nem toda instrução `RET` faz voltar para o BASIC, pois podemos ter sub-rotinas dentro do programa em linguagem de máquina.

Da mesma forma, quando o programa volta para o BASIC, ele traz no par de registradores BC um valor que é o resultado da função `USR` aplicada ao argumento X. Vamos entender isso melhor: imagine um programa em BASIC que, a certa altura, tenha a seguinte linha:

510 LET L=USR 16514

O interpretador BASIC vai fazer a variável L igual ao resultado da aplicação da função `USR` ao argumento 16514. Os seguintes eventos então ocorrem:

- o programa sai do BASIC e vai para o endereço 16514, levando no par de registradores BC o valor 16514;
- o programa em linguagem de máquina, com início nesse endereço, é executado até a instrução de `RETURN`, quando então, o valor que estiver no par de registradores BC volta para o BASIC como sendo o resultado da função `USR`.

Normalmente, não nos interessa o resultado da função `USR`, mas, sim, o que o programa em linguagem de máquina faz. Os programas em linguagem de máquina serão apresentados neste livro de acordo com o esquema abaixo:

End. (decimal)	End. (hexadecimal)	Rótulo	Instrução mnemônica	Cód. inst. (hexadecimal)	Nota
-------------------	-----------------------	--------	------------------------	-----------------------------	------

Os endereços decimal e hexadecimal, bem como o Rótulo e a Nota existirão ou não, dependendo da instrução.

Vamos, então, escrever o nosso primeiro programa:

```
16514 RET C9
```

Usamos o programa "MESTRE" para carregar o programa em linguagem de máquina na instrução REM, e, para rodá-lo, temos duas opções: através de um comando direto como RAND USR 16514, ou PRINT USR 16514, ou, então, colocamos uma instrução no BASIC como:

```
10 PRINT USR 16514
```

e fazemos RUN. O que vamos obter como resultado é o número 16514 escrito na tela, pois o par de registradores BC foi carregado com o endereço 16514 e, como não existe instrução alguma alterando o conteúdo desses registradores, o resultado é o próprio valor 16514.

Como segunda experiência, façamos o seguinte:

```
16514 LD B,0 0600  
RET C9
```

A resposta agora é 130. Por quê?

Ora, o par BC foi carregado pela instrução USR com 16514 ou 4082, em hexadecimal, isto é (B) = 40H = 16384 e (C) = 82H = 130. Como fazemos (B) = 0 com a instrução LDB, 0, então voltando para o BASIC teremos 130, que é o conteúdo de C.

O programa que segue lê o conteúdo de duas posições de memória carregadas pelo BASIC, faz a soma e coloca o resultado no par de registradores BC:

```
16514 (4082) ... 00  
16515 (4083) ... 00  
16516 LD A, (16514) 3A8240 (A) = X  
LD H, A 67 (H) = X  
LD A, (16515) 3A8340 (A) = Y  
16523 ADD A, H 8C  
LD C, A 4F  
LD A, 0 3E00  
ADC A, A 8F  
LD B, A 47  
RET 00
```

As posições de memória 16514 e 16515 foram reservadas para receber os números a serem somados; em seguida, o acumulador é carregado com o primeiro número (X) e o conteúdo de A transferido para H. Em 16523, fazemos a soma de X e Y e, em seguida, o resultado é transferido para C. As três instruções seguintes carregam o registrador B com 1 se houver "carry" na soma, isto é, se ela for maior que 255, e 0, se não houver "carry".

Para rodar essa rotina, use o programa em BASIC abaixo:

```

10 PRINT "X=";
20 INPUT X
30 PRINT X
40 POKE 16514,X
50 PRINT "Y=";
60 INPUT Y
70 PRINT Y
80 POKE 16515,Y
90 PRINT USR 16516

```

Repare-se que, desta vez, o argumento de USR é 16516, pois o programa começa efetivamente nessa posição, uma vez que em 16514 e 16515 estamos armazenando dados.

Aproveitando o programa anterior em BASIC, façamos agora a multiplicação de X por Y.

16514	(4082)	...	00
16515	(4083)	...	00
		LD A,(16514)	3A8240
		LD B,A	47
		CP A,0	FE00
		JR Z (A)	2810
		LD HL,0	210000
		LD D,0	1600
		LD A,(16515)	3A8340
		LD E,A	5F
(B)		ADD HL,DE	19
		DJNZ (B)	10FD
		LD B,H	44
		LD C,L	4D
		JR (C)	1803
(A)		LD BC,0	010000
(C)		RET	C9

O programa inicialmente verifica se o conteúdo de 16514 é zero. Se for, pula para o final, uma vez que a resposta é zero. Se não for, carrega o registrador B com esse valor, para servir de contador da instrução DJNZ. Isto é, soma o conteúdo de DE (previamente carregado com o valor de Y) em HL (inicialmente nulo) tantas vezes quantas for o número X. Finalmente, transfere o resultado de HL para BC, a fim de retornar com a resposta para o BASIC.

Vejamos, ainda nesta parte inicial, mais dois pequenos programas. No primeiro, calculamos o fatorial de um número e, no segundo, usamos de outra forma o resultado de USR X.

No cálculo do fatorial de N, carrega-se, através do BASIC, o número desejado na posição de memória 16507 (407B). No capítulo seguinte, vamos ver porque usamos essa posição para guardar dados.

O programa em linguagem de máquina começa lendo o valor N e verificando se é zero ou não. Se for zero, o programa volta para o final, onde é colocado 1 como resposta. Isso é feito porque o fatorial de 0 vale 1 por definição, não seguindo o algoritmo de cálculo do fatorial. Como vamos obter a resposta do fatorial através do par de registradores BC, o maior N possível é 8, cujo fatorial dá 40320.

O cálculo de fatorial é feito multiplicando-se o valor N por N-1, usando o método descrito no programa anterior. O resultado que fica no par de registradores HL é transferido para o par DE e, fechando o "loop", multiplicado por N-2 e, assim, sucessivamente, até que $N-P = 0$, quando a resposta é transferida para o par de registradores BC e retorna para o BASIC. Vamos ver, passo a passo:

16514	LD A, (16507)	3A7B40	Carrega N no acumulador.
	CP A, 0	FE00	Compara N com 0.
	JRZ (A)	2819	Salta 25 "bytes" se N = 0.
	LD HL, 0	210000	Zera o par HL para a multiplicação.
	LD D, 0	1600	Carrega o par DE com N.
	LD E, A	5F	
	DEC A	3D	Decrementa A.
	JRZ (A)	2810	Salta para o final se $N-1 = 0$, i. é, se $N=1$.
(C)	LD B, A	47	Carrega B com N-1, que é o contador da instrução DJNZ
(B)	ADD HL, DE	19	Soma N, N-1 vezes, i. é, faz $N(N-1)$
	DJNZ (B)	10FD	
	PUSH HL	E5	Troca o conteúdo do HL para DE
	POP DE	D1	

LD HL,0	210000	Zera HL, preparando para a próxima multiplicação.
DEC A	3D	Acha o novo valor de B para o "loop" de multiplicação.
JRNZ (C)	20F4	Volta ao "loop" de multiplicação se o contador B for diferente de zero.
PUSH DE	D5	Coloca a resposta no par BC para a volta ao BASIC.
POP BC	C1	
JR (D)	1803	
(A) LD BC,1	010100	Carrega BC com 1.
(D) RET	C9	Retorna ao BASIC.

Para rodar esse programa, utiliza-se, em BASIC, o que segue:

```

10 PRINT "N=";
20 INPUT N
30 PRINT N
40 POKE 16507,N
50 PRINT USR 16514
60 GOTO 10

```

O próximo programa utiliza o resultado da função USR X de forma diferente daquela que temos usado até agora. O programa, muito simples, verifica se um número é par ou ímpar, analisando a posição 0 do acumulador, onde foi carregado o número dado. Se o acumulador for par, então o "bit" 0 será 0, o que faz a instrução seguinte JRZ verdadeira. Então, dependendo se o número é par ou ímpar, carrega-se o par de registradores BC com os valores 70 ou 90, respectivamente, que são exatamente os números das linhas no programa em BASIC onde estão as mensagens "É PAR" e "É IMPAR". Assim, estamos usando a função USR X associada à instrução GOTO. Aqui também usamos a posição de memória 16507, para guardar o número dado.

É o seguinte o programa em linguagem de máquina:

16514	LD A, (16507)	3A7B40	Carrega o acumulador com o número dado N.
	BIT 0,A	C647	Analisa o "bit" menos significativo de A.
	JRZ (A)	2E04	Salta se o "bit" 0 de A for 0.
	LD BC,90	015A00	Carrega BC com 90, se N for ímpar.
	RET	C9	Retorna ao BASIC.
(A)	LD BC,70	014500	Carrega BC com 70, se N for par.
	RET	C9	Retorna ao BASIC.

O programa em BASIC é o seguinte:

```

10 PRINT "ENTRE COM O NUMERO"
20 INPUT N
30 CLS
40 PRINT N;
50 POKE 16507,N
60 GOTO USR 16514
70 PRINT " E" PAR"
80 GOTO 10
90 PRINT " E" IMPAR"
100 GOTO 10

```

Finalmente, como é de praxe em livros que tratam de linguagem de máquina, vamos mostrar dois programas que bem demonstram a velocidade com que se consegue fazer o PRINT. O primeiro programa utiliza uma sub-rotina do ROM que será mais adiante explicada. O segundo programa só pode ser rodado em computadores com mais de 3,5K de RAM. Esse segundo programa é bem mais poderoso do que o primeiro, e recomendamos ao leitor rodá-lo em SLOW e FAST e comparar com o tempo gasto por um programa similar em BASIC. Se o leitor ficou ansioso por ver esse segundo programa rodando em seu computador e se esse dispõe somente de 1K ou 2K de RAM, damos um programa que vai permitir rodar essa segunda demonstração do PRINT.

O primeiro programa é simplesmente:

16514	LD A, 128	3E80
(A)	RST 16	07
	JR (A)	18FD

Para rodá-lo, basta fazer RAND USR 16514.

O segundo programa é o seguinte:

16527	LD HL, (16396)	2A0C40
	INC HL	23
	LD A, 128	3E80
	LD C, 22	0E16
	(B) LD B, 32	0620
	(A) LD (HL), A	77
	INC HL	23
	DJNZ (A)	10FC
	DEC C	0D
	RET Z	C8
	INC HL	23
	JR (B)	18F5

Para aqueles que têm menos do que 3,5K de RAM, damos a seguir uma rotina que prepara a memória RAM para essa segunda demonstração.

16514	LD A, 0	3E00
	LD C, 4	0E04
	(B) LD B, 176	0680
	(A) RST 16	D7
	DJNZ (A)	10FD
	DEC C	0D
	JRNZ (B)	20F8
16526	RET	C9

A parte em BASIC é a seguinte:

```
10 RAND USR 16514
20 INPUT X$
30 RAND USR 16527
```

O programa pára na linha 20, já com a memória preparada para a rotina que começa em 16527 aguardando uma entrada. Aperte-se, então, NEW LINE e tela-se

IDÉIAS PARA MONTAR UM PROGRAMA

3

Escrever um programa em linguagem de máquina é bem diferente do que escrever um programa em BASIC ou em qualquer linguagem de alto nível, e as principais razões são as seguintes:

- as instruções em linguagem de máquina são bem mais elementares que as instruções em BASIC. Por essa razão, o programa tem que ser dissecado muito mais que em BASIC;
- o programador determina completamente o andamento do programa, tendo que se preocupar com todos os detalhes, tais como a locação de espaços na memória para o programa e para os dados, áreas de manobra, etc.
- como em linguagem de máquina comanda-se diretamente o microprocessador, pequenos erros costumam ter conseqüências catastróficas, levando o programa, na maioria das vezes, à auto-destruição.

Em outras palavras, quando se programa em linguagem de máquina, fica-se totalmente exposto aos erros do programa, contrariamente ao que ocorre com as linguagens de alto nível. Numa linguagem de alto nível como o BASIC, os interpretadores preocupam-se com erros de sintaxe e, quando uma falha ocorre ao rodar o programa, param a execução sem destruir o programa, emitindo ainda um relatório (normalmente codificado), indicando o tipo de erro e a linha em que o mesmo ocorreu.

A conclusão é que devemos ser ultra meticolosos ao elaborarmos um programa em linguagem de máquina se quisermos programar rapidamente e com confiabilidade.

As recomendações que damos a seguir, se bem que possam parecer óbvias e desnecessárias, mostram-se na prática de enorme utilidade:

- Analise profundamente o problema, pensando exclusivamente nos elementos do problema e não em um possível método de solução. Por exemplo, quando se trata de implementar jogos, é fundamental se estabelecer as regras e a lei de formação das jogadas ou movimentos.
- Determine os elementos fundamentais do problema, tais como velocidade de execução, necessidade ou não de movimentos rápidos na tela, número

- Tendo já analisado todos os aspectos conceituais do problema, verifique a conveniência de se fazer o programa em linguagem de máquina.
- Uma vez definido que se trata de uma programação em linguagem de máquina, faça um diagrama de blocos do problema onde apareçam claramente os pontos em que devem ser tomadas decisões, pontos onde o usuário deve entrar com dados, pontos onde algo vai ser escrito na tela. Determine, também, o andamento do tempo, ou seja, se alguma sub-rotina ou mesmo o programa todo deve fluir sincronizado com o tempo real, ou se a velocidade de andamento do programa deve ser feita de outra forma, como, por exemplo, usando contadores para gerar atrasos.
- Determine todas as variáveis envolvidas no problema, inclusive os dados. Determine para cada variável a gama de variações da mesma, isto é, numérica ou alfanumérica; se numérica, pode ela ser representada por um número inteiro ou não. Se puder ser representada por um número inteiro, é ela sempre positiva ou não? Finalmente, no caso de ser um número inteiro positivo, tem-se que saber se ele é menor que 256 (1 "byte"), menor que 65536 (2 "bytes"), ou, se for maior que 65535, terá então que ser representado por vários "bytes" ou por uma variável em ponto flutuante.
- Feito tudo isso, determine as regiões da memória RAM onde vão ser guardados os dados do problema, as variáveis usadas na solução do problema e o programa em si. Lembre-se que os dados ou variáveis não podem estar no meio do programa, pois seriam interpretados como instruções e não como dados.
- Nesse ponto, comece a escrever o programa seguindo o esquema do diagrama de blocos e colocando em cada bloco o endereço do início do segmento de programa que implementa a função do bloco. Isso vai ser muito útil mais tarde, para achar falhas no programa.
- Ao final de cada trecho importante do programa, deixe espaço, isto é, coloque dois zeros (instrução NOP), para, futuramente, se necessário, colocar uma instrução de retorno ao BASIC.
- Nos pontos de retorno ao BASIC, certifique-se de que o número total de PUSHs é igual ao número total de POPs, até aquele ponto. Se isso não ocorrer, você perde o programa. Quando o programa tem muitas sub-rotinas, fica às vezes difícil analisar todos os caminhos possíveis para fazer essa contagem de PUSHs e POPs, mas faça, porque é fundamental.
- Certifique-se de que as instruções estão corretas, verificando se elas têm o número de "bytes" necessários. Quando se usam as instruções JUMP e JUMP RELATIVE e o salto é para um ponto do programa não escrito ainda, tem-se que deixar em aberto o "byte" no JUMP RELATIVE que indica a extensão do salto, ou os dois "bytes", no JUMP, que indicam o endereço do salto. A fim de evitar erros na contagem do número de "bytes" do

programa, não deixe em branco os lugares que indicam os saltos. Coloque, por exemplo, XX para cada "byte", e no final do programa substitua esses XX pelos valores corretos.

- Finalmente, quando o programa estiver pronto, carregue-o na instrução REM e, antes de rodar (o que é muito importante), grave-o. É importante gravar antes de rodar porque muitos erros alteram ou destroem o próprio programa.
- Se o programa apresentar defeito e você fizer algumas alterações no mesmo, não se esqueça de renumerar a linha de instrução afetada pela alteração e de conferir todas as instruções de salto, relativo e absoluto, para ver se seus argumentos ainda são válidos.
Boa sorte.

UTILIZANDO AS VARIÁVEIS DO SISTEMA

4

Como já dissemos anteriormente, o programa monitor administra totalmente as atividades do micro e, para tal, fora as sub-rotinas do ROM, ele necessita de um espaço da memória RAM para armazenar uma série de variáveis que ficam sob seu controle (veja apêndice E).

Esse pedaço da memória RAM fica justamente no início da mesma e é fixo, ocupando a região entre os endereços 16384 e 16508, inclusive. Ao conjunto de variáveis aí armazenadas chamamos VARIÁVEIS DO SISTEMA. No apêndice D, é dada uma relação de todas essas variáveis, seus endereços e, ainda, alguns comentários.

Não pretendemos aprofundar-nos na análise dessas variáveis, mesmo porque a maioria delas é de uso exclusivo da máquina. No entanto, existem algumas que são muito úteis para o programador e passíveis de serem alteradas sem levar a máquina ao descontrole. É dessas variáveis que vamos nos ocupar agora.

• ESPAÇOS VAZIOS

O programa monitor deixa três “bytes” dessa região de memória sem uso e, portanto, disponíveis para o programador. Esses espaços são:

- 16417 ou 4021 em hexadecimal
- 16507 ou 407B em hexadecimal
- 16508 ou 407C em hexadecimal.

O leitor deve estar se perguntando porque se preocupar com três “bytes” se temos uma memória RAM de vários K! Bom, realmente é muito pouca coisa, mas devemos conhecê-la por duas razões:

- é comum, depois de pronto um programa, descobriremos que nos esquecemos de uma variável e geralmente isso ocorre quando a região que reservamos para as mesmas já está totalmente ocupada. Nesse caso, esses três “bytes” vão parecer muitos K. Lembre-se que não se pode colocar dados

- como esses três “bytes” estão dentro da região de trabalho da máquina, eles não são apagados pelas instruções CLEAR ou RUN, portanto são lugares seguros para guardarmos certos dados.

Ainda mais quando estamos escrevendo pequenos programas, como muitos apresentados neste livro, que não requerem mais do que três “bytes” para guardar dados, é prático usarmos diretamente esses “bytes” para não nos preocuparmos com a alocação dos dados na memória.

Bem, existe ainda um conjunto de 32 “bytes” destinados ao “buffer” da impressora que podemos usar em nosso programa levando em conta que esses “bytes” são apagados quando o programa pára e o comando da máquina volta ao usuário. Assim, essa região é muito útil para variáveis temporárias, isto é, que são criadas pelo programa mas que, ao final do mesmo, possam ser destruídas. Esses “bytes” ficam compreendidos entre os endereços

16444 ou 403C em hexadecimal

e

16475 ou 405B em hexadecimal, inclusive.

• RAMTOP (16388/16389)

Nas posições 16388 e 16389, o computador guarda o endereço seguinte ao último “byte” da memória RAM.

Inicialmente, ao ser ligado, o computador entra nas rotinas de iniciação, sendo que uma das primeiras é verificar o tamanho da memória RAM. Para tal, ele carrega (POKE) e lê (PEEK), “byte” por “byte”, até detectar o fim da memória, quando, então, coloca na variável RAMTOP esse valor somado a 1.

Podemos alterar o valor de RAMTOP e, com isso, enganar a máquina, pois ela lê esse valor toda vez que precisa realocar as regiões da memória. Assim, se colocarmos RAMTOP em um valor mais baixo, a máquina vai ignorar tudo que estiver acima desse valor. Esse artifício seria muito útil para armazenarmos programas em linguagem de máquina se não fosse o inconveniente de que o SAVE não guarda na fita o que está acima do RAMTOP.

Para alterarmos o valor do RAMTOP, devemos fazer POKE nas posições 16388 e 16389 e, depois, dar o comando NEW, para que a máquina percorra novamente as rotinas iniciais e realoque a memória de acordo com esse valor de RAMTOP.

Nós vimos, no fim do Capítulo 2, um programa para a demonstração da velocidade do PRINT, que requeria uma memória maior que 3,5K. Se o seu computador tem menos que 3,5K de RAM, faça POKE 16389,80 e rode aquele programa. Assim, enganamos a máquina, mas cuidado quando enganamos no sentido de aumentar a memória: pode-se criar o caos, dependendo da extensão do programa.

• VARS (16400/16401)

Essa variável contém o endereço do início do setor da memória onde estão guardadas as variáveis do programa em BASIC. As variáveis, sejam elas numéricas, "strings" ou "arrays", vão sendo armazenadas seqüencialmente na memória, a partir de VARS, à medida que vão sendo definidas no programa em BASIC.

A máquina reconhece o início de cada variável e o tipo de variável (numérica, "string", etc.), pela forma diferente como cada tipo é armazenado.

Para finalidade deste livro, interessa-nos unicamente conhecer como são armazenadas as variáveis numéricas e as "strings", ou seja, não vamos nos preocupar com as "arrays".

Mais adiante, faremos uso desse conhecimento, detectando e alterando o valor das variáveis.

Vejamos, então, inicialmente, como são armazenadas as variáveis numéricas. Sabemos que o nome da variável numérica tem necessariamente que começar com uma letra, e poderá, em seguida, ter ou não outros caracteres alfanuméricos. Se o nome da variável contiver uma única letra, a variável será armazenada da seguinte forma:

1.º byte	64 + código da letra
2.º byte	Representação binária do valor da variável em ponto flutuante.
3.º byte	
4.º byte	
5.º byte	
6.º byte	

Assim, se a primeira variável definida no programa em BASIC for:

10 LET M=8

teremos:

POSIÇÃO DA MEMÓRIA	VALOR	
PEEK 16400 + 256 × PEEK 16401	114	= 64 + 50, onde 50 é o código de M
PEEK 16400 + 256 × PEEK 16401 + 1	132	Representação binária do número 8 em ponto flutuante.
+ 2	0	
+ 3	0	
+ 4	0	
+ 5	0	

Se, agora, o nome da variável contiver mais de um caracter ocorre o seguinte: aos códigos do primeiro e último caracteres soma-se 128, e os caracteres intermediários ficam com valores iguais aos de seus códigos.

Assim, se a primeira variável definida é:

10 LET CASA=50

teremos:

POSIÇÃO DA MEMÓRIA	VALOR	
PEEK 16400 + 256 × PEEK 16401	168	= 128 + 40, onde 40 é o código de C
PEEK 16400 + 256 × PEEK 16401 + 1	38	código de A
+ 2	56	código de S
+ 3	166	= 128 + 38, onde 38 é o código de A
+ 4	134	} Representação binária de 50 em ponto flutuante
+ 5	72	
+ 6	0	
+ 7	0	
+ 8	0	

Por último, se a primeira variável definida no programa em BASIC for uma STRING (que só pode ser formada por uma letra e o símbolo \$), teremos o primeiro "byte" formado pelo código da letra somado a 32, seguido de dois "bytes" que indicam o número de caracteres da STRING, seguido, ainda, dos códigos dos caracteres do texto da STRING. Vejamos, então, um exemplo:

10 LET C\$="58P9A"

teremos:

POSIÇÃO DE MEMÓRIA	VALOR	
PEEK 16400 + 256 × PEEK 16401	72	= 32 + 40, onde 40 é o código de C
PEEK 16400 + 256 × PEEK 16401 + 1	5	} Comprimento da STRING
+ 2	0	
+ 3	33	código de 5
+ 4	36	código de 8

+ 5	53	código de P
+ 6	37	código de 9
+ 7	38	código de A

Bem, se o leitor tiver curiosidade de verificar esses exemplos dados, ou mesmo exercitar outros casos, poderá usar o programa abaixo, onde a linha 10 e a linha 20 podem mudar conforme o tipo de variável e o nome:

```

10 INPUT X           (OU INPUT X$)
20 PRINT X           (OU PRINT X$)
30 LET P=PEEK 16400+256*PEEK 1
6401
40 FOR J=0 TO 20
50 PRINT P+J,PEEK (P+J)
60 NEXT J

```

Se o seu micro tiver menos que 3,5K de RAM, esse programa não vai funcionar. Por que? Se a memória é menor que 3,5K, então o micro, ao ser ligado, não expande a memória na região da tela, mas sim deixa essa região totalmente comprimida, para economizar memória. Isso quer dizer que, para cada PRINT, haverá um deslocamento da área reservada às variáveis (pois essa vem depois da área da tela) e, então, o valor de P calculado na linha 30 não acompanhará a evolução de PEEK 16400 + 256 × PEEK 16401. Restam, então, duas alternativas: uma é recalcular P imediatamente antes de cada PRINT, e a outra é abrir a memória da tela, tomando o cuidado para não definir uma variável antes da que nos interessa.

Para recalcular P antes de cada PRINT, poderíamos fazer:

```

10 INPUT X           (OU INPUT X$)
20 PRINT X           (OU PRINT X$)
30 LET P$="PEEK 16400+256*PEEK
16401"
40 FOR J=0 TO 20
50 PRINT VAL P$+J,PEEK (VAL P$
+J)
60 NEXT J

```

• PIL FIM (16412/16413)

Na parte alta da memória existe uma região reservada para cálculos (abordaremos em detalhes esse assunto mais adiante) cuja finalidade é...

estão guardados nas variáveis PIL FUN (16410/16411) e PIL FIM (16412/16413). Nessa área, cada número é representado por 5 "bytes", pois estão em notação científica, isto é, ponto flutuante. A pilha é organizada pelo sistema FIFO (o primeiro número a entrar é o primeiro a sair). Por ora, esta informação é suficiente. Vamos deixar para estudar a parte de cálculo separadamente.

• DF — SZ (16418)

Essa variável, de um único "byte", indica o número de linhas na parte inferior da tela, reservadas para a edição de mensagens. Normalmente, essa variável vale 2. Isso quer dizer que temos 22 linhas para fazer PRINT e mais 2 linhas para edição. Se fizermos um POKE em 16418, podemos alterar a área de edição e, até mesmo, anulá-la. Mas cuidado: se você a anular, tem que tomar o cuidado para restabelecer esse espaço para edição (ao menos duas linhas), antes da primeira instrução INPUT, se não o computador perde o controle.

• FRAMES (16436/16437)

Esses dois "bytes" formam o contador de tempo da máquina que funciona da seguinte forma: quando o micro é ligado, o programa monitor coloca "1" s em todos os 16 "bits" e sincronizadamente com a geração de quadros da TV, esse contador é decrementado 60 unidades por segundo, até que chegue em 1000000000000000, (32768), o que demora 9 minutos e 6 segundos quando, então, volta à condição inicial, isto é, ao valor 65535. A instrução PAUSE usa esse contador colocando inicialmente no bit 15 (o mais significativo) o valor 0.

Ora, esse contador é um ótimo relógio para ser usado em nossos programas em linguagem de máquina, pois podemos, a qualquer momento, colocar um valor nesse contador e tomar uma decisão quando ele atingir um certo valor especificado.

Isso nos permite, então, fazer um contador até 9 minutos e 6 segundos (não devemos colocar o valor 0 no "bit" número 15).

Vamos ver, então, na prática, como fazemos para utilizar uma base de tempo em nossos programas.

O problema se resume sempre em criar um retardo de valor desejado. Podemos, então, formalizar mais o problema, dividindo-o em três casos:

- Retardo maior que 4 segundos
- Retardo entre 1/60 segundos e 4 segundos
- Retardo menor que 1/60 segundos.

No primeiro caso, que, na verdade, está limitado a, no máximo, 9 minutos e 6 segundos, podemos usar o contador FRAMES, carregando-o inicialmente com FFFF (hexadecimal) e medindo-o constantemente, até que atinja um valor tal que a variação demore o tempo de retardo desejado. Vamos supor que desejamos um retardo de 12 segundos. Como cada segundo representa uma variação (diminuição) de 60 unidades na variável FRAMES, vamos deixá-la variar de $60 \times 12 = 720$ unidades. Como o valor inicial é 16535, então temos que ler a variável FRAMES até que ela chegue ao valor $16535 - 720 = 15815$.

Vejamos, então, um programa, que é para simples demonstração de uso do contador de tempo. Usando-se o BASIC, carrega-se no par de "bytes" 16507/16508 o valor final a que deve chegar a variável FRAMES. O programa em linguagem de máquina fica lendo a variável FRAMES até que essa atinja o valor desejado, quando então o programa volta ao BASIC e é feito o PRINT da linha 100 para indicar o final do tempo. Note que o programa pára na linha 80 para que, apertando-se NEW LINE, comande-se o início da contagem de tempo. Em linguagem de máquina, temos:

16514	LD HL, 65535	21FFFF	} Carrega FRAMES com 65535.
	LD (FRAMES), HL	223440	
	AND A	A7	} Zera o "carry" para subtração.
	LD DE, (16507)	ED5B7B40	} Carrega DE com o valor final.
	(A) LD HL, (FRAMES)	2A3440	} Lê FRAMES.
	SBC HL, DE	ED52	
	JRNZ (A)	20F9	} Verifica se chegou ao fim.
	RET	C9	} Retorna ao BASIC.

e, para rodar, use o programa em BASIC:

```

10 PRINT "T=";
20 INPUT T
30 PRINT T: " SEGUNDOS"
40 LET T=65535-T*60
50 LET I=INT (T/256)
60 POKE 16508,I
70 POKE 16507,T-256*I
80 INPUT X$
90 RAND USR 16514
100 PRINT "FIM"

```

Para o segundo caso, isto é, retardo entre 1/60 segundos e 4 segundos, podemos usar o "byte" menos significativo de FRAMES (16436) e trabalhar com ele livremente, uma vez que ele não apresenta restrições quanto a valores permitidos. Assim, seguindo a mesma linha de raciocínio do caso anterior, temos:

16514	LD A, (16507)	3A7B40	} Carrega 16436 com o tempo de retardo.
	LD (16436), A	323440	
16520	(A) LD A, (16436)	3A3440	} Lê o tempo.
	AND A	A7	
	JRNZ (A)	20FA	} Verifica se chegou a zero.
	RET	C9	
			} Volta para a linha 16530 se não terminou.
			} Volta ao BASIC.

E, em BASIC:

```

10 PRINT "T=";
20 INPUT T
30 PRINT T;" SEGUNDOS"
40 POKE 16507,T*60
50 INPUT X$
60 RAND USR 16514
70 PRINT "FIM"

```

Reparem que aqui, fazemos diretamente POKE do valor do tempo que queremos e verificamos quando a variável FRAMES chega a zero. A instrução AND A é uma forma econômica de se verificar se o acumulador está em zero ou não (por que?).

Finalmente, se necessitarmos de um retardo inferior a 1/60 segundos, temos que sair da variável FRAMES e usar o tempo de execução das instruções em linguagem de máquina. Você encontrará esses tempos em qualquer manual do Z-80. Não se esqueça de corrigir os tempos dependendo do "clock" do seu micro. Na prática, o que se faz é criar uma variável e fazer um "loop" onde ela é decrementada até chegar a zero. Para exemplificar, vamos mostrar um pequeno programa que pode ser usado como gerador de retardo e calcular o tempo de execução do mesmo. O programa funciona da seguinte forma:

- em um lugar qualquer da memória (no caso usamos os "bytes" 16507 e 16508), armazena-se um número (N) entre 1 e 65535;

- o programa começa, então, lendo esse valor e, através de dois "loops" concatenados, faz-se percorrer um trecho do programa N^2 vezes.

Como o tempo de execução está amarrado à frequência do "clock" da máquina, esse gerador de retardo é absolutamente preciso. Esse programa permite gerar retardos de $27\mu s$ (para $N = 1$) até 9 horas 32 minutos e 38 segundos (para $N = 65535$) estando no modo FAST, ou esse tempo multiplicado por 6,32, se a máquina estiver no modo SLOW.

O programa é o seguinte:

Linha n.º			
1	LD DE, (16507)	ED5B7B40	Carrega DE com N.
2	ⓑ LD HL, (16507)	2A7B40	Carrega HL com N.
3	ⓐ DEC HL	2B	} Primeiro "loop".
4	LD A, H	7C	
5	OR A, L	B5	
6	JRNZ ⓐ	20FB	} Segundo "loop".
7	DEC DE	1B	
8	LD A, D	7A	
9	OR A, E	B3	
10	JRNZ ⓑ	20F3	

Esse programa de 10 linhas pode ser, para efeito de cálculo de tempo de execução, dividido nos quatro grupos de instruções como indicado na Fig. 4.1 (os números à esquerda nos blocos indicam as linhas do programa que pertencem a esse bloco, e os números à direita indicam o tempo de execução da instrução em μs).

1	6,15	Bloco 1
2	4,92	Bloco 2
3	1,85	Bloco 3
4	1,23	
5	1,23	
6	3,69	
<hr style="width: 50px; margin-left: auto;"/>		
8,00		
7	1,85	Bloco 4
8	1,23	
9	1,23	
10	3,69	
<hr style="width: 50px; margin-left: auto;"/>		
8,00		

Ora, o bloco 1 é percorrido uma única vez, o bloco 2 é percorrido N vezes, assim como o bloco 4, e o bloco 3 é percorrido N^2 vezes.

Então o tempo total de execução do programa será dado por:

$$T = T_1 + N (T_2 + T_4) + N^2 \cdot T_3$$

$$T = 6,15 + 12,92 \cdot N + 8 \cdot N^2 \text{ (}\mu\text{s)}$$

Com boa aproximação, podemos considerar $T \cong 8 \cdot N^2 \mu\text{s}$.

USANDO A ROM 5

O programa monitor e o interpretador são formados por uma enorme série de sub-rotinas em linguagem de máquina, guardadas na ROM. Assim, quando é dado um comando à máquina, ou quando ela está interpretando um programa, aquelas rotinas pertinentes são percorridas.

Essas sub-rotinas estão à nossa disposição e devemos conhecê-las, ao menos as mais importantes, para sabermos se são úteis ou não ao programa que estamos elaborando.

Com isso queremos dizer que muitas vezes necessitamos de uma determinada rotina de programa que está implementada na ROM mas que apresenta um dos seguintes inconvenientes:

- é muito extensa, porque vai além da necessidade específica que temos;
- é lenta. (Ao final do livro, por exemplo, vamos mostrar uma rotina de PLOT/UNPLOT bem mais rápida que a do ROM);
- requer que os dados estejam disponíveis de uma forma inconveniente para o programador, ou, o que é mais comum, ela destrói o conteúdo dos registradores com isso atrapalhando a vida do programador.

Vamos então ver as sub-rotinas mais importantes.

As principais são:

• RESET

Essa sub-rotina começa no endereço 0000 da ROM e faz exatamente as funções desempenhadas pela máquina ao ser ligada, isto é; verifica o tamanho da RAM, colocando o valor correto na RAMTOP e, depois, pula para a rotina NEW.

Pode-se chamar essa sub-rotina fazendo um JUMP para o endereço 0000, ou usando a instrução RST 0, isto é, o código C7.

• PRTCHA

Essa sub-rotina faz o PRINT do caracter cujo código é o conteúdo do acumula-

PRINT num lugar específico, diferente da posição seguinte de PRINT, temos que preparar antes a posição do PRINT. Essa é uma das mais úteis rotinas e tem a vantagem de não alterar o conteúdo dos registradores. Para chamá-la, usa-se RST 10 (código D7).

Já vimos, no Capítulo 2, um programa que enche a tela de preto e usa essa sub-rotina.

Para exercitar um pouco nossos conhecimentos, façamos um programa que simula uma máquina de escrever. A parte em BASIC é a seguinte:

```

10 IF INKEY$ <> "" THEN GOTO 10
20 IF INKEY$ = "" THEN GOTO 20
30 LET A$ = INKEY$
40 RAND USR 16514
50 GOTO 10

```

Nas linhas 10 e 20 fazemos a leitura do teclado (um único caracter de cada vez). Na linha 30 definimos a primeira variável do programa como sendo o último caracter pressionado; em seguida, vamos à rotina em linguagem de máquina, e, em 50, voltamos para aguardar o próximo caracter.

Como a variável A\$ contém o código da última tecla pressionada, e é a primeira variável definida, sabemos como lê-la, pois se trata de uma STRING de um único caracter cuja posição inicial é dada pela variável VARS. Em linguagem de máquina temos, então:

16514	LD HL, (VARS)	2A1040	Carrega VARS em HL
	INC HL	23	} Vai até aonde está o código da STRING.
	INC HL	23	
	INC HL	23	
	LD A, (HL)	7E	Carrega o código no acumulador.
	RST 10	D7	Chama a sub-rotina PRTCHA da ROM.
	RET	C9	Retorna ao BASIC.

• KSCAN

Essa sub-rotina faz a varredura do teclado e retorna com a informação da tecla acionada. Aconselhamos ao leitor consultar um dos livros da bibliografia para entender o método utilizado pela máquina para codificar a tecla que foi pressionada. Por ora, só nos interessa saber que, ao ser chamada essa sub-rotina, ela retorna

com a informação no par de registradores HL (existem códigos mesmo para as teclas acionadas com "shift"). No apêndice C, daremos uma tabela que mostra o valor dos registradores HL para cada tecla, com e sem "shift". Quando não existe tecla pressionada, o valor de H e de L é FF.

Essa sub-rotina começa no endereço 02BB. Para chamá-la, usa-se uma instrução CALL (ex CDBB02).

Voltemos ao exemplo, da máquina de escrever, e vamos alterar o programa, transferindo do BASIC para linguagem de máquina as duas primeiras linhas do programa. A linha 10 faz com que o programa fique nesta linha se houver uma tecla pressionada nesse instante. A instrução da linha 20 faz o inverso, isto é, se nenhuma tecla está pressionada, o programa fica parado nessa instrução. Isso faz com que, entre duas teclas acionadas, tenha que haver um período de repouso.

O programa em BASIC ficaria, então:

```

10 RAND USR 16514
20 LET A$=INKEY$
30 RAND USR 16527
40 GOTO 10

```

e, em linguagem de máquina, teríamos:

16514	(A)	CALL KSCAN INC L JRNZ (A)	CDBB02 2C 20FA
16520	(B)	CALL KSCAN INC L JRZ (B)	CDBB02 2C 28FA
16527		RET LD HL, (VARS) INC HL INC HL INC HL LD A, (HL) RST 10 RET	C9 2A1040 23 23 23 7E D7 C9

A implementação de IF INKEY\$ <> " " THEN GOTO 10 é feita inicialmente, pois, se existir alguma tecla pressionada, então o registrador L não conterà FF e, portanto, se incrementado, não terá o valor 00 e, então, o salto condicional será feito e assim ficará até que não exista tecla acionada. Aí caímos na implementação de IF INKEY\$ = " " THEN, que funciona de forma inversa.

• ACHR

Essa sub-rotina complementa a sub-rotina anterior (KSCAN), pois, a partir da informação dessa, ela fornece um endereço onde está o código da tecla pressionada.

Seu endereço é 07BD, portanto, para chamá-la, usa-se, por exemplo, CDBD07. Então, se quisermos saber o código de uma tecla pressionada, devemos fazer o seguinte:

- chamar a sub-rotina KSCAN, que retorna com a informação da tecla pressionada, no par de registradores HL;
- transferir o conteúdo de HL para BC (por exemplo, fazendo PUSH HL e, em seguida, POP BC);
- chamar a sub-rotina ACHR que, ao retornar, terá no par de registradores HL o endereço de onde está o código da tecla pressionada.

Apliquemos, então, esse recurso adicional para escrever o programinha da máquina de escrever, totalmente em linguagem de máquina.

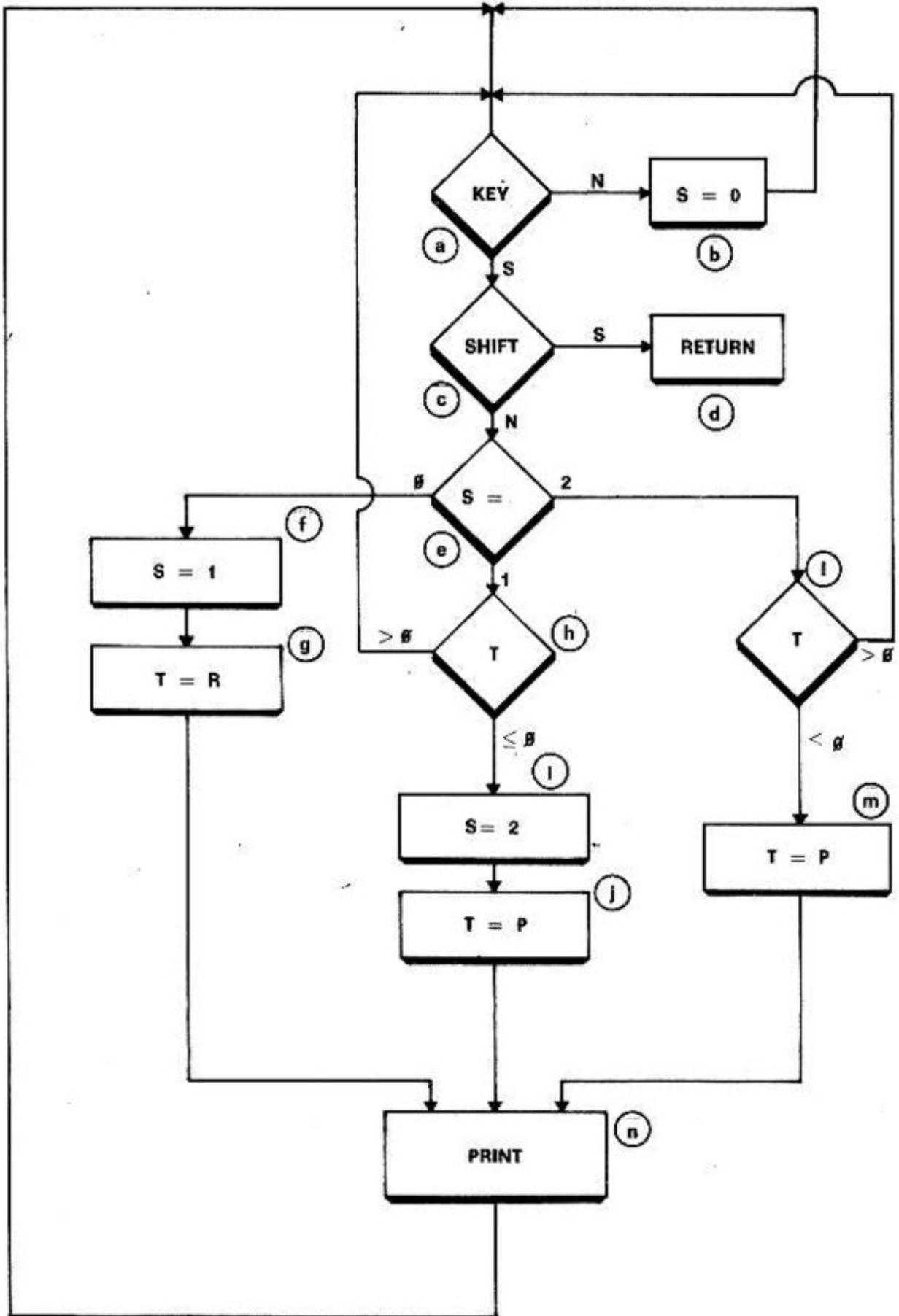
16514	(A)	CALL KSCAN	CDBB02	
		INC L	2C	IF INKEYS <> " " THEN
		JRNZ (A)	20FA	
16520	(B)	CALL KSCAN	CDBB02	
		INC L	2C	IF INKEYS = " " THEN
		JRZ (B)	28FA	
16526		DEC L	2D	Recupera o valor original de L
		PUSH HL	E5	Transfere o conteúdo de
		POP BC	C1	HL para BC
		CALL ACHR	CDBD07	Acha o endereço do código da
				tecla
		LD A, (HL)	7E	Carrega o acumulador com o
				código da tecla.
		RST 10	D7	Chama PRTCHA
		JR (A)	18EA	Retorna ao início

Para chamar o programa, faça RAND USR 16514.

Como a informação da tecla que foi pressionada é obtida ao se chamar KSCAN na linha 16520, temos que usar o conteúdo de HL aí obtido para chamar a sub-rotina ACHR. Daí a instrução DEC L.

Antes de passarmos para outra sub-rotina da ROM, vamos insistir uma vez mais na nossa máquina de escrever, dotando-a de dois recursos extras:

- retorno ao BASIC a qualquer momento. Note que, no último programa apresentado, só conseguimos voltar ao BASIC quando o PRINT chega ao



fim da tela, pois aí a sub-rotina PRTCHA faz o retorno automaticamente, por não existir mais posição para o PRINT;

- função REPEAT, isto é, se uma tecla ficar pressionada um determinado tempo, então o caracter passa a ser "escrito" seguidamente, até que a tecla seja solta.

Vamos primeiramente definir claramente o funcionamento da nossa máquina de escrever:

- 1) A máquina começa a escrever a partir do canto superior esquerdo.
- 2) Se nenhuma tecla é pressionada, ela fica parada, aguardando o pressionamento de uma tecla.
- 3) Uma tecla, ao ser pressionada, escreve o caracter imediatamente.
- 4) Se uma tecla fica pressionada mais do que um determinado tempo, que chamaremos de atraso (R), então o caracter em questão começa a ser escrito seguidamente, em intervalos de tempo que chamaremos P (período), até que a tecla seja solta.

Como a máquina pode estar em três estados possíveis, isto é, aguardando o pressionamento de uma tecla, com uma tecla pressionada mas antes do início da repetição, e repetindo o caracter, vamos definir uma variável de estado, que chamaremos de S, e que poderá assumir os valores 0, 1 e 2, respectivamente. Essa variável será armazenada no endereço 16417.

O "atraso" e o "período" serão definidos pelo usuário (no BASIC) e armazenados nas posições 16507 e 16508, respectivamente. Para a contagem do tempo de atraso e período, usaremos o relógio da máquina, utilizando um único "byte", como já vimos.

Analisemos, então, o diagrama de blocos da figura 5.1.

Para facilidade de análise, os blocos estão marcados com as letras "a" a "n". Os blocos "a" e "b" garantem que a variável de estado S fique com o valor 0 quando o teclado não está sendo pressionado. Ao ser pressionado, vamos para o bloco "c", que analisa o conteúdo do registrador H. Se a tecla "shift" tiver sido pressionada, volta-se ao BASIC — bloco "d" —. No bloco "e" o programa é dividido, dependendo do valor de S. Se for 0, fazemos $S = 1$ (pois entramos nesse estado) e colocamos o "byte" menos significativo do contador de tempo da máquina (FRAMES) igual ao tempo de atraso que queremos (R). Esses passos estão implementados nos blocos "f" e "g", respectivamente. Como a tecla acaba de ser pressionada, temos que fazer imediatamente um PRINT. Por isso, pulamos para o bloco "n" e daí para o início do programa. Se, no entanto, no bloco "e" o valor de S for 1, pulamos para o bloco "h", que verifica se já se passou o tempo de retardo R ($T \leq 0$) ou se ainda não ($T > 0$). Se não se passou o tempo de retardo, o programa fica girando nos blocos "a", "c", "e" e "h", até que $T \leq 0$ e o programa vai para "i". Nesse ponto, entraremos no estado 2 (bloco "i") e temos que usar a outra base de tempo P, que

determina o tempo do PRINT entre as repetições. Isso é feito no bloco "j". Se em "e" temos $S = 2$, analogamente ao que é feito em "h", temos que verificar se o tempo de repetição P já foi atingido ou não (bloco "l"). Se foi, precisamos colocar novamente a variável de tempo (FRAMES) no valor de P (bloco "m") para repetir o ciclo de repetição.

Esse programa ocupa 90 "bytes" de memória. Então, para carregá-lo com o programa MESTRE é preciso, primeiramente, aumentar a instrução REM para, no mínimo, 90 caracteres.

É o seguinte o programa:

16514 (4082) (B)	CALL KSCAN	CDBB02	} bloco "a"
	LD (16444),HL	223040	
	INC L	2C	
	JRNZ (A)	2007	
	LD A,0	3E00	} bloco "b"
	LD (16417),A	322140	
	JR (B)	18F0	} bloco "c"
(A)	BIT 0,H	CB44	
	RET Z	08	} bloco "d"
	LD A,(16417)	3A2140	} bloco "e"
	AND A	A7	
	JRNZ (C)	2000	
	INC A	3C	
	LD (16417),A	322140	} bloco "f"
	LD A,(16507)	3A7B40	
	LD (FRAMES),A	323440	} bloco "g"
	JP (D)	C3D040	
(C)	CP A,2	FE02	} bloco "h"
	JRZ (E)	2015	
	LD A,(FRAMES)	3A3440	
	CP A,0	FE00	} bloco "i"
	LD (FRAMES),A	3A3440	

	LD R, 2	3E02	}	bloco "i"
	LD (16417), A	322140		
	LD R, (16508)	3A7C40	}	bloco "j"
	LD (FRAMES), A	323440		
	JP (D)	C3D040	}	bloco "l"
(E)	LD A, (FRAMES)	3A3440		
	CP R, 0	FE00		
	JP P (B)	F28240		
	LD R, (16508)	3A7C40		
	LD (FRAMES), A	323440	}	bloco "m"
16592 (40D0)	(D) LD BC, (16444)	ED4B3C40		
	CALL ACHR	CDBD07	}	bloco "n"
	LD R, (HL)	7E		
	CALL PATCHA	D7		
	JP (B)	C38240		

Lembre-se que:

- o atraso R está em 16507 (407B);
- o atraso P está em 16508 (407C);
- o estado S está em 16417 (4021);
- o "byte" de menor ordem, de FRAMES, está em 16436 (4034).

Note que o valor do par de registradores HL, obtido em "a", vai ser usado no fim do programa em "n", para chamar a sub-rotina ACHR. Por isso, na segunda instrução guardamos esse valor na posição de memória 16444 (403C).

Para rodar o programa, é preciso, inicialmente, carregar (POKE) as posições 16507 e 16508 com o valor dos retardos R e P. Os valores permitidos para essas duas variáveis estão entre os limites 1 e 127 (isto é, até 2 segundos, aproximadamente). Valores maiores que 127 não são permitidos.

• PRINT AT

Esta sub-rotina, como o próprio nome diz, prepara a posição para o próximo PRINT. Seu endereço é 08F5 e, ao chamá-la, coloque no registrador B o número da linha e, no registrador C, o número da coluna para o próximo PRINT, obedecendo ao sistema de contagem usado pela função PRINT AT do BASIC.

Para demonstrar o uso dessa sub-rotina, vamos analisar o programa seguinte, que faz um PRINT na coluna 15, do conteúdo de uma variável STRING.

A primeira providência é entrar com a STRING desejada como primeira variável definida no BASIC.

O programa em linguagem de máquina é o seguinte:

16514	LD H, (VARS)	2A1040	Carrega VARS em HL
	INC HL	23	} Carrega A com o comprimento da STRING
	LD A, (HL)	7E	
	AND A	A7	} Se a STRING tiver comprimento nulo ou maior que 21, retorna ao BASIC
	RET Z	C8	
	CP A, 22	FE16	
	RET P	F0	
	LD D, 0	1600	} Carrega DE com o comprimento da STRING
	LD E, A	5F	
	LD BC, 150F	010F15	Prepara BC para o PRINT AT com: linha = 21, coluna = 15
16530	(A) PUSH BC	C5	} Salva os registradores BC e DE e faz o PRINT AT
	PUSH DE	D5	
	CALL PRINT AT	CDF508	
	POP DE	D1	
	POP BC	C1	
	LD HL, (VARS)	2A1040	} Faz HL apontar inicialmente para o último caracter da STRING
	INC HL	23	
	INC HL	23	
	ADD HL, DE	19	} Carrega A com o código do caracter
	LD A, (HL)	7E	
	RST 10 (PRTCHA)	D7	Faz o PRINT
	DEC B	05	} Prepara B e E para o próximo PRINT
	DEC E	1D	
	RET Z	C8	
	JR (A)	18EC	Volta para 16530

• PSTR

Esta sub-rotina faz o PRINT de uma STRING de forma direta. Ela está loca-

Ao chamá-la, deve-se ter o par de registradores DE apontando para o endereço da memória onde começa a STRING, e o par BC contendo o número de caracteres da STRING.

No exemplo que segue, o espaço da memória de 16514 a 16523 é reservado para receber uma STRING de até dez caracteres. Essa STRING é carregada pelo BASIC. O programa em BASIC faz ainda um POKE em 16534 para colocar no registrador B o número de caracteres da STRING. Vamos ao exemplo:

16514 a 16523		reservado para a STRING	
16524	LD BC,0B05	01050B	} Faz o início do PRINT em L=11 e C=5
	CALL PRINT AT	CDF50B	
	LD DE,16514	118240	Carrega DE com 16514
	LD BC,10	010A00	Carrega BC com o comprimento da STRING
	CALL PSTR	CD6B0B	
	RET	C9	

e, em BASIC, temos:

```

10 PRINT "ENTRE COM O TEXTO"
20 INPUT A$
30 IF LEN A$ > 10 THEN GOTO 20
40 FOR J=1 TO LEN A$
50 POKE 16513+J, CODE A$(J)
60 NEXT J
70 POKE 16534, LEN A$
80 CLS
90 RAND USR 16524

```

As duas primeiras instruções em linguagem de máquina foram incluídas unicamente para melhorar a estética do PRINT.

• CLS

Esta sub-rotina, localizada em 0A2A, perfaz exatamente a função de comando CLS do BASIC.

• FAST

A posição inicial desta sub-rotina é 0F23. Faz exatamente a função do FAST em BASIC.

• SLOW

A posição inicial desta sub-rotina é 0F2B. Faz exatamente a função do SLOW em BASIC.

• PLOT/UNPLOT

Com início na posição 0BB2, esta sub-rotina faz o PLOT/UNPLOT nas coordenadas indicadas pelo par de registradores BC. Para se fazer PLOT, tem-se inicialmente que carregar a posição de memória 16432 (4030) com 155 (9B) e, para fazer UNPLOT, tem-se que carregar essa posição com 160 (A0).

A coordenada X será dada pelo conteúdo do registrador C e a coordenada Y pelo conteúdo do registrador B.

Uma aplicação prática dessa sub-rotina e útil para uso em jogos é a da bola sendo rebatida nas bordas da tela. O leitor encontrará no capítulo de aplicações um programa chamado PLOT/UNPLOT RÁPIDO, que implementa essa função de forma mais rápida e é mais prática para uso do que essa sub-rotina da ROM.

Para tornar o programa um pouco mais atraente, faremos com que as teclas 5, 6, 7 e 8 permitam determinar a direção do movimento da bola. A posição da bola será dada pelas coordenadas X e Y, e o deslocamento da bola pelas variáveis DX e DY. Assim, a próxima posição da bola será dada por

$$X = X + DX$$

$$Y = Y + DY$$

Na figura 5.2 temos o diagrama de blocos do programa. Inicialmente, é feita a leitura do teclado (a). Se nenhuma tecla está pressionada, o programa pula para (g). Se a tecla pressionada é NEW LINE, então o programa retorna para o BASIC (f). Se a tecla pressionada for 5 (b), então a variável deslocamento DX será feita igual a -1, isto é, a próxima posição da bola terá a abscissa X diminuída de uma unidade (a menos que já esteja na borda esquerda — veja bloco (j) —, ou seja, a bola estará deslocando-se para a esquerda).

Analogamente se a tecla pressionada for a 6, 7 ou 8. Note que, se nenhuma tecla estiver pressionada, então os deslocamentos DX e DY anteriormente determinados serão mantidos. Logo, a bola estará andando em linha reta.

No bloco (g), é apagada a posição atual da bola. O bloco (h), calcula a próxima coordenada X da bola e analisa se é uma coordenada possível ou não. Por exemplo, se o valor de $X = 1$, $DX = 255$ (isto é, -1 em número relativo),

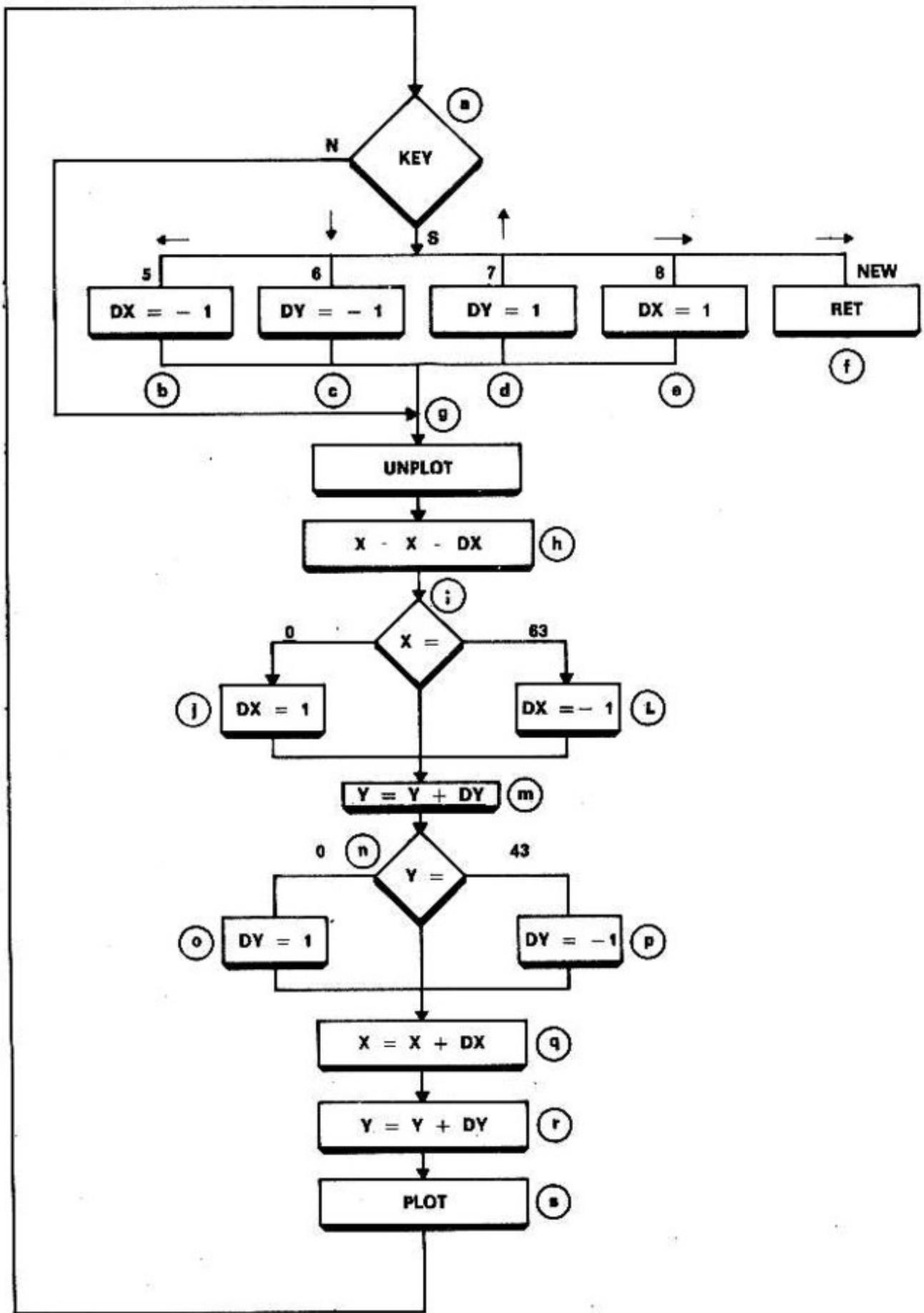


Fig. 5.2

então, a próxima posição de X passa a ser $1 + 1 = 2$. O programa foi feito assim para permitir ao leitor desenhar uma borda nas posições extremas da tela, pois a bola nunca atingirá essas coordenadas. Os blocos (m), (n), (o) e (p) fazem a análise da ordenada Y. Em (q) e (r), atualiza-se o par de coordenadas X e Y e, em (s), faz-se o PLOT.

As variáveis do sistema estão armazenadas nas seguintes posições:

X 16507 (407B)
 Y 16508 (407C)
 DX 16444 (403C)
 DY 16445 (403D)

O programa em BASIC gera a posição inicial da bola, aleatoriamente, bem como os deslocamentos DX e DY.

Em BASIC, temos:

```

10 POKE 16507,1+62*RND
20 POKE 16508,1+42*RND
30 POKE 16444,1
40 IF 2*RND>1 THEN POKE 16444,
255
50 POKE 16445,1
60 IF 2*RND>1 THEN POKE 16445,
255
70 LET L=USR 16514
  
```

Em linguagem de máquina, o programa ocupa 130 "bytes". Portanto, antes de entrar com ele no programa MESTRE, ajuste o tamanho da instrução REM.

16514	(P)	CALL KSCRN	CDBB02
		LD DE,01FF	11FF01
		BIT 3,L	CB5D
		JRZ (A)	2810
		BIT 5,H	CB6C
		JRZ (B)	2815
		BIT 4,H	CB64
		JRZ (C)	2814
		BIT 3,H	CB5C
		JRZ (D)	2807

bloco (a)

INC L	2C	} bloco (f)
RET NZ	C0	
JR (E)	1810	

(A) LD A,D	7A	} blocos (b) e (e)
JR (F)	1801	
(D) LD A,E	7B	
(F) LD (16444),A	323C40	
JR (E)	1807	

(B) LD A,D	7A	} blocos (c) e (d)
JR (G)	1801	
(C) LD A,E	7B	
(G) LD (16445),A	323D40	

(E) LD BC, (16507)	ED4B7B40	} UNPLOT
LD A, 150	3EA0	
LD (16432),A	323040	
PUSH DE	D5	
CALL UNPLOT	CDBB0B	
POP DE	D1	

LD A, (16507)	3A7B40	} bloco (h)
LD B,A	47	
LD A, (16444)	3A3C40	
ADD A,B	80	

JRZ	(H)	2806	} blocos (i) , (j) e (l)
CP	A, 63	FE3F	
JRZ	(I)	2805	
JR	(J)	1807	
(H)	LD A, E	7B	
JR	(K)	1801	
(I)	LD A, D	7A	
(K)	LD (16444) , A	323C40	

(J)	LD A, (16508)	3A7C40	} bloco (m)
	LD B, A	47	
	LD A, (16445)	3A3D40	
	ADD A, B	60	

JRZ	(L)	2806	} blocos (n) , (o) e (p)
CP	A, 43	FE2B	
JRZ	(M)	2805	
JR	(N)	1807	
(L)	LD A, E	7B	
JR	(O)	1801	
(M)	LD A, D	7A	
(O)	LD (16445) , A	323D40	



Ⓝ	LD HL, (16507)	2A7B40	} blocos Ⓚ e Ⓡ
	LD DE, (16444)	ED5B3C40	
	LD R, L	7D	
	ADD R, E	83	
	LD C, R	4F	
	LD A, H	7C	
	ADD A, D	82	
	LD B, A	47	
	LD (16507), BC	ED437B40	
	LD A, 155	3E9B	} bloco Ⓢ
	LD (16432), A	323040	
	CALL PLOT	CDE20B	
	JP Ⓟ	C38240	

Infelizmente, a qualidade desse programa é prejudicada pelo desempenho da sub-rotina PLOT/UNPLOT. O leitor verá que a bola não faz movimentos com velocidade constante. Como já dissemos, mostraremos uma rotina de PLOT/UNPLOT que permite um bom desempenho para esse programa. Também por essa razão, e para facilitar o transplante desse programa para outra posição qualquer de memória, usam-se exclusivamente saltos relativos, à exceção do último (volta para a primeira linha), que não pode ser definido como relativo pois é maior que 128 (por ironia, é 129).

Vimos neste capítulo as rotinas de ROM mais práticas para serem usadas pelo programador, exceto as rotinas de cálculo, para as quais — dada a sua extensão — reservamos os próximos dois capítulos.

CALCULANDO EM LINGUAGEM DE MÁQUINA

6

Do ponto de vista de cálculo, as instruções do Z-80 permitem que se façam somas e subtrações em números até 65535. É verdade que, com as instruções de "shift", fica fácil multiplicar ou dividir um número por potência de 2. De qualquer forma, para trabalharmos com ponto flutuante e para fazermos cálculos científicos, temos que usar as sub-rotinas da ROM.

Neste capítulo abordaremos unicamente esse tópico. Vamos ver que temos acesso a todas as rotinas de cálculo. Em outras palavras, vamos poder, dentro de programa em linguagem de máquina, fazer qualquer cálculo dentre o repertório oferecido pelas funções do BASIC.

A máquina, toda vez que cria uma variável numérica (LET), ou toda vez que se entra com uma variável numérica (INPUT), o faz armazenando essa variável como uma seqüência de 5 "bytes", que indicam: o sinal, o expoente e a mantissa.

Então, quando o programa em BASIC pede um cálculo qualquer (mesmo que seja uma simples soma), esse cálculo é feito utilizando essa notação de 5 "bytes". Para simplificar a explicação, vamos dar nomes aos "bytes", como segue: A B C D E.

Temos, então, que converter um número decimal, inteiro ou não, positivo ou negativo, em um conjunto de 5 "bytes". A primeira fase consiste em verificar se o número é zero ou não. Se for zero, então os 5 "bytes" serão 0. Note-se que regras que apresentaremos em seguida não nos levam a essa conclusão quanto à representação de zero. É por conveniência que o zero é assim representado.

Como segunda fase — isto é, números diferentes de zero —, vamos converter o número de decimal em binário (esquecendo, por ora, o sinal), tanto a parte inteira quanto a parte fracionária. Bem, a conversão da parte inteira de decimal em binário é fácil e estamos acostumados a fazê-la. Para converter a parte fracionária, devemos proceder da seguinte forma:

- escrever a parte decimal (parte à direita da vírgula) como sendo uma fração. Ex.: no número 18,125, a parte decimal é 0,125, que pode ser escrita como sendo

- achar uma fração equivalente à obtida anteriormente, onde o denominador seja uma potência de 2. Nem sempre vamos encontrar um denominador que faça com que o numerador seja um número inteiro. Nesse caso, quanto maior o denominador que pegarmos, maior será a precisão. Sempre que o numerador for um número decimal, tomamos unicamente sua parte inteira. No caso do exemplo anterior, isto é, o número 18,125, fazemos o seguinte:

$$\frac{125}{1000} = \frac{1}{8} = \frac{1}{2^3}$$

Seja agora o número 7,35. A parte decimal é $\frac{35}{100} = \frac{7}{20}$

Podemos ter, então,

$$\frac{7}{20} = \frac{11,20}{32} \cong \frac{11}{32}$$

Essa aproximação dá um erro de 1,8%.

Ou poderíamos fazer:

$$\frac{7}{20} = \frac{89,6}{256} \cong \frac{89}{256}$$

que dá um erro de 0,67%;

- escrever o numerador assim obtido em binário e dividir esse número por 2^n , onde 2^n é o denominador adotado na obtenção do numerador. Assim, no caso do último exemplo, onde achamos a fração $89/256$, faríamos $89 = 1011001$, que dividiríamos por $256 = 2^8$, que resulta na fração binária $0,01011001$.

Note-se que dividir um número binário por 2^n é equivalente a correr a vírgula n casas para a esquerda.

Convertido então o número de decimal em binário, vamos achar o expoente. Nós vamos procurar um expoente tal que o número em binário tenha a sua parte inteira sempre igual a zero, e tenha sempre 1 imediatamente após a vírgula.

Ora, achar o expoente é simplesmente correr a vírgula para a direita ou para a esquerda até que cheguemos à condição $0,1xxx\dots$ (onde x significa 0s ou 1s).

Ao correr a vírgula uma casa para a direita, estamos diminuindo o expoente de uma unidade e, inversamente, correndo a vírgula uma casa para a esquerda, estamos aumentando o expoente de uma unidade.

Vejam os caso do número 7,35. Já sabemos que sua representação binária é $111,01011001$, que podemos escrever como sendo $0,11101011001$ com o expoente $+3$.

Agora já podemos achar o primeiro "byte" dos cinco que irão representar o número: o "byte" A, que nos dá o expoente. Como podemos ter expoentes

positivos e negativos, foi adotada a convenção dos números relativos para o expoente, o que equivale a dizer que devemos somar 80 (em hexadecimal) ao valor do expoente obtido. Assim, no exemplo anterior, o expoente seria $80H + 3 = 83H$. É assim que se obtém o "byte" A.

Vimos que a mantissa, isto é, o número que se obtém após o ajuste do expoente, é um número de forma $0,1xxx\dots$, portanto, como sempre, teremos um 1 após a vírgula. Esse 1 é redundante e, conseqüentemente, dispensável. Nada melhor, então, do que usar esse "byte" para indicar se o número é positivo ou negativo. Assim, se o número for negativo, mantém-se o 1 após a vírgula, e, se é positivo, troca-se o 1 por 0. Feito isso, preenche-se a mantissa à direita com tantos zeros quantos são necessários para totalizar quatro "bytes", isto é, 32 bits. Esses "bytes" são os "bytes" BCDE.

Para fixar a idéia, vamos converter mais dois números de decimal para a notação do computador:

$$X = -13,25$$

$$Y = 0,42$$

- Parte inteira de $X = 1101$
Parte inteira de $Y = 0$
- Parte fracionária

$$\text{de } X: 0,25 = \frac{25}{100} = \frac{1}{4} = \frac{1}{2^2}$$

Portanto: 0,01

$$\text{de } Y: 0,42 = \frac{42}{100} = \frac{21}{50} = \frac{53,76}{128} \cong \frac{53}{128} = \frac{53}{2^7}$$

Portanto: 0,0110101

- Conversão para binário:

$$X = 1101,01$$

$$Y = 0,0110101$$

- Obtenção dos expoentes:

$$X = 0,110101 \quad \text{exp} = +4 \quad \text{Portanto: "byte" A} = 84H$$

$$Y = 0,110101 \quad \text{exp} = -1 \quad \text{Portanto: "byte" A} = 7FH$$

- Colocação do sinal:

$$X \text{ (negativo)} \rightarrow \text{mantissa} = 0,110101$$

$$Y \text{ (positivo)} \rightarrow \text{mantissa} = 0,0110101$$

- Geração da mantissa completa (4 "bytes")

$$\text{de } X \quad 11010100 \quad 00000000 \quad 00000000 \quad 00000000$$

$$\text{de } Y \quad 01010100 \quad 00000000 \quad 00000000 \quad 00000000$$

• Representação em hexadecimal:

X = 84 D4 00 00 00
 Y = 7F 54 00 00 00

Aconselhamos o leitor a exercitar um pouco essa conversão de decimal para binário em ponto flutuante utilizando, para verificação do resultado, o programa a seguir onde o número N decimal entra no campo de variáveis do micro e de lá é lido e convertido para hexadecimal:

```

10 DIM A(5)
20 PRINT "N= ";
30 INPUT N
40 PRINT N,
50 IF N=0 THEN GOTO 100
60 LET V=PEEK 16400+256*PEEK 1
6401
70 FOR J=1 TO 5
80 LET A(J)=PEEK (V+J)
90 NEXT J
100 FOR J=1 TO 5
110 LET I=INT (A(J)/16)
120 PRINT CHR$( 28+I);CHR$( 28+
A(J)-16*I);" ";
130 NEXT J
140 PRINT
150 GOTO 10
  
```

Se o leitor for verificar os exemplos aqui citados de conversão vai encontrar algumas divergências. Por exemplo: na conversão do número $Y = 0,42$ vamos encontrar 7F 57 0A 3D 70, que é, no entanto, uma aproximação melhor de 0,42, uma vez que, em nosso cálculo, utilizamos, na obtenção da parte fracionária, o numerador 128. Se, ao invés de 128, utilizarmos números maiores, vamos obter precisão melhor.

Qual será, então, o maior e o menor número que o nosso computador pode guardar?

Bem, o maior número será dado pelo maior expoente, que é FF, e por uma mantissa cheia de "1", exceto no "byte" B, onde o "bit" mais significativo será 0, pois se trata de um número positivo. Então, o maior número é

FF 7F FF FF FF

O expoente FF (isto é, 255) representa uma potência de $(255 - 128) = 127$. Portanto se trata do número binário:

$$0,111\dots 1 \cdot 2^{127} = \underbrace{111 \dots 1000 \dots 0}_{32 \text{ 1s} \quad 95 \text{ 0s}}$$

que é aproximadamente igual a $1,7014118 \cdot 10^{38}$.

Para o menor número (diferente de 0) temos

01 10 00 00 00 que vale $2,9387359 \cdot 10^{-39}$

onde o expoente é -127 , e não -128 , como seria de se esperar. Acontece que a máquina reserva o valor 0 para o "byte" A exclusivamente para o número 0. Isso é feito para acelerar um pouco as rotinas de cálculo, pois basta saber o valor do "byte" A para se saber se o número é zero ou não.

Bem, agora que dominamos a representação binária dos números em ponto flutuante, podemos esquecer-la, uma vez que a máquina encarrega-se dessas terríveis conversões.

As rotinas de cálculo constituem possivelmente o capítulo mais interessante do uso de linguagem de máquina, pois não se trata de rotina de cálculo apenas, mas algo ainda mais poderoso e extremamente flexível quanto ao uso.

Olhemos atentamente a Figura 6.1. Vemos inicialmente que as rotinas de cálculo (RC) comunicam-se com o programa em linguagem de máquina, com toda a memória da máquina (RAM & ROM), com uma região chamada "pilha de cálculo" (a qual também é acessada pelas rotinas normais de linguagem de máquina) e, finalmente, com uma área chamada "memória para cálculo". Os números EF e 34 serão explicados logo a seguir.

A "pilha de cálculo" é a interligação da rotina em linguagem de máquina com as RC para transferência de variáveis, sejam elas numéricas ou "strings". Essa pilha é semelhante à "stack" (a pilha de PUSHs e POPs), salvo quanto a:

- a pilha de cálculo não é invertida, isto é, à medida que vamos colocando dados nessa pilha, os mesmos vão entrando em endereços cada vez maiores;
- ao contrário da "stack", que guarda unicamente dois "bytes" de cada vez, pois armazena sempre pares de registradores, a pilha de cálculo, quando trabalha com variáveis numéricas, utiliza a notação de cinco "bytes" (ponto flutuante em binário), mas podendo também armazenar "strings" de tamanho variado;
- a pilha de cálculo fica numa região da memória indicada pelas variáveis PILFUN (posição 16410 e 16411) e PILFIM (posição 16412 e 16413), isto é, a primeira variável contém o endereço do início da pilha e a segunda o endereço de fim da pilha;
- a pilha de cálculo é um pouco volátil, ou seja, devemos tomar cuidado para não deixar dados armazenados nessa pilha quando estamos pulando de linguagem de máquina para BASIC. Veremos isso, mais adiante, de forma mais clara.

O leitor a esta altura deve estar perguntando como se vai passar um dado do programa em linguagem de máquina para a pilha de cálculo, se essa passagem

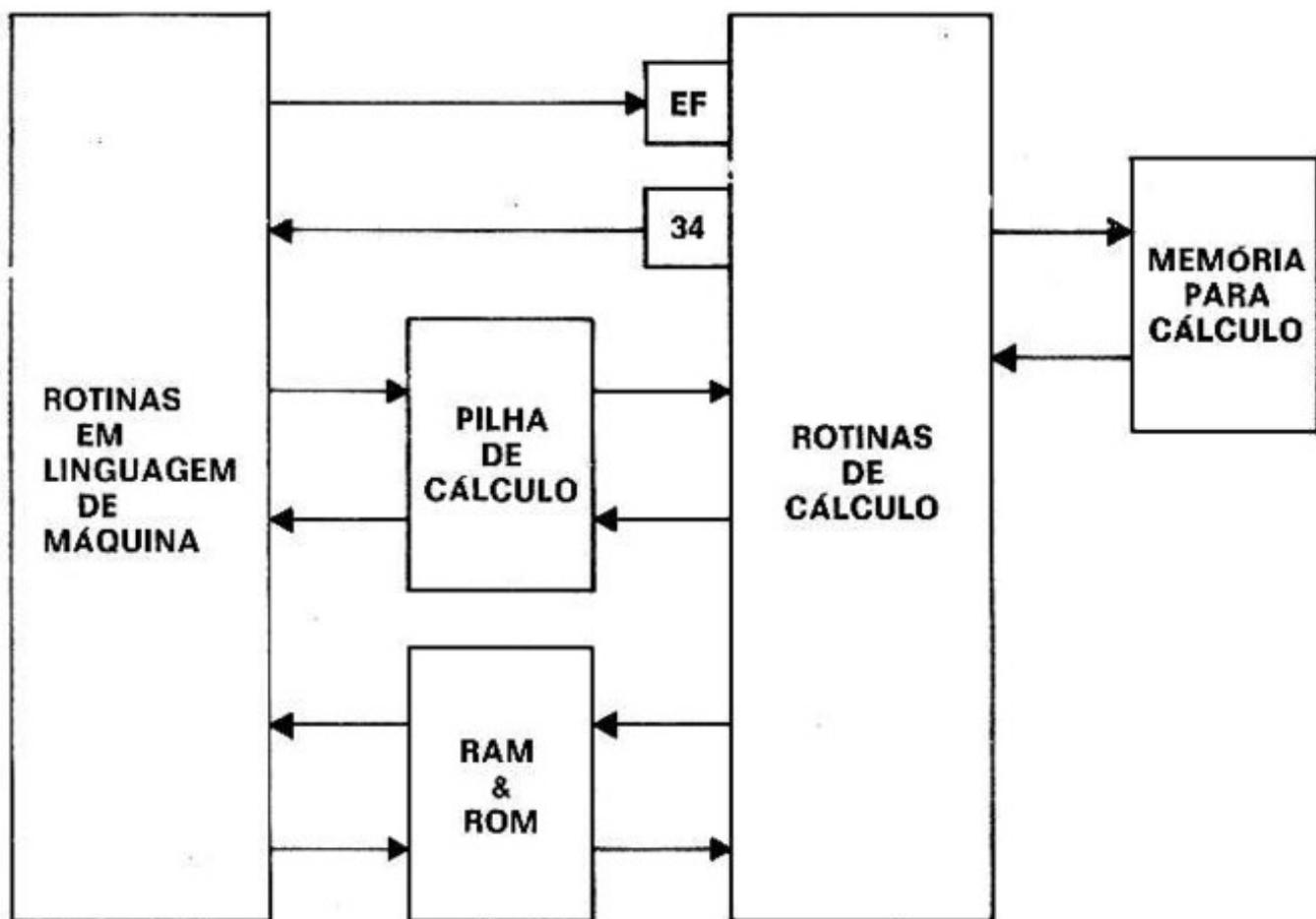


Fig. 6.1 Rotinas de cálculo e sua interligação com a máquina

tem que ser feita via pilha de cálculo, isto é, usando cinco "bytes" para variáveis numéricas, se em linguagem de máquina não trabalhamos com essa notação. A dúvida procede e o que deve ser feito é o seguinte:

- se for uma variável numérica armazenada pelo BASIC, é fácil, pois ela já está armazenada nessa notação. Trata-se, pois, de deslocar cinco "bytes" do campo das variáveis para a pilha de cálculo;
- se for uma variável armazenada nos registradores do Z-80, isto é, números de 0 a 65535, temos que converter esse número na notação de cinco "bytes" e aí colocar na pilha. Não deve haver preocupação, pois a ROM é generosa até nesse caso, apresentando uma sub-rotina que se encarrega disso. E, por que não, quando tivermos que colocar de volta um dado numérico da pilha de cálculo num par de registradores também vamos encontrar uma sub-rotina da ROM que faz isso. Para colocar da pilha no campo de variáveis é fácil, pois se trata de um simples deslocamento de cinco "bytes".

Vejam, agora, a memória para cálculo. Essa é uma memória de uso exclusivo das rotinas de cálculo, que só pode guardar variáveis na notação de cinco "bytes" e tem capacidade para seis variáveis, isto é, ocupa trinta "bytes". Essa memória está localizada no fim da porção de RAM destinada a variáveis do sistema, ocupando do "byte" 16477 ao "byte" 16506, inclusive. Vamos ver o que significam os dois bloquinhos junto às RC indicados por EF e 34. Estando num programa em linguagem de máquina, para irmos para a RC chamamos da ROM a sub-rotina RST 28, cujo código é EF.

Feito isso, entramos no mundo do cálculo, onde não vamos nos preocupar com acumulador, registradores nem mesmo com as instruções do Z-80. Temos aí uma terminologia própria. Ora como a Fig. 6.1 mostra, que para sairmos da RC temos que usar a senha de saída, que é 34. O que vem no nosso programa em linguagem de máquina depois desse 34 são as instruções normais de linguagem de máquina.

Assim, o programa que segue é um simples passeio pelas RC, sem nada fazer lá.

```
16514    EF    entra em RC
          34    saída das RC
          C9    volta ao BASIC
```

Chamaremos de literal aos códigos usados dentro da RC, uma vez que não se tratará de instruções propriamente ditas, mas sim chamadas de sub-rotina da ROM. Assim, 34 é um literal.

Antes de explorarmos os recursos da RC, descrevendo as funções dos literais, vamos nos ocupar com a comunicação com a pilha de cálculo e a memória de cálculo. Mas, para que seja possível fazer algo em termos de cálculo, vamos apresentar as literais das operações fundamentais:

FUNÇÃO	LITERAL
Soma	0F
Subtração	03
Multiplicação	04
Divisão	05

Para realizarmos qualquer uma dessas operações, precisamos de dois números. Esses números são o último e o penúltimo números colocados na pilha de cálculo. A subtração será feita subtraindo o último número do penúltimo, e a divisão, dividindo o penúltimo pelo último. O leitor que conhece a RPN ("reverse polish notation"), usada em calculadoras como a HP, não terá dificuldade para trabalhar com essa pilha, por ser idêntica àquela quanto ao uso. Recomendamos ao leitor que não estiver seguro quanto ao deslocamento de dados nessa pilha, quando são adicionados novos dados ou retirados, que revise o STACK de 7 es-

Ao efetuarmos uma operação entre dois números (sempre o penúltimo e o último da pilha), o resultado será colocado no fim da pilha, os dois números, objetos da operação, desaparecerão, e a pilha se deslocará de uma posição. Na Figura 6.2 mostramos o que ocorre:

POSIÇÃO DE DADOS NA PILHA DE CÁLCULO

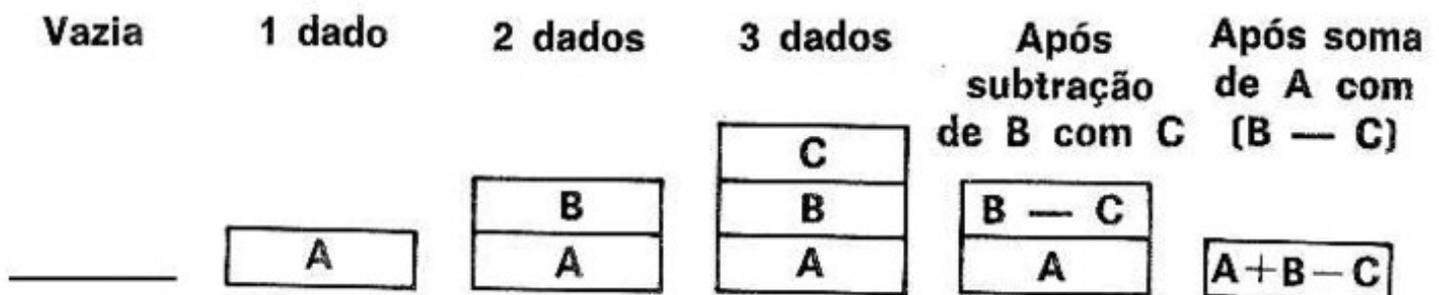


Figura 6.2 Colocando um dado dos registradores na pilha de cálculo

Temos duas sub-rotinas que colocam dados do Z-80 para a pilha de cálculo. A primeira, conhecida como "STACK A", coloca o conteúdo do acumulador na pilha. Seu endereço é 151D. Para chamá-la, usa-se CD1D15. A outra sub-rotina fica em 1520, e coloca o conteúdo do par de registradores BC na pilha de cálculo. Para chamar essa sub-rotina, usa-se CD2015. Essa sub-rotina é conhecida como "STACK BC".

Para retirar um dado da pilha e colocar nos registradores temos em 0EA7 uma sub-rotina conhecida como "UNSTACK BC" (para chamar, usa-se CDA70E). No entanto muito cuidado, porque essa volta da RC para os registradores só pode ser feita se o número armazenado na pilha for positivo e se seu inteiro for menor do que 65535. Se não, vai dar erro e a máquina voltará ao sistema de comando. Temos que ter em mente, também, que essa sub-rotina "UNSTACK BC" não pega simplesmente a parte inteira do número no topo da pilha, mas faz o arredondamento para o inteiro mais próximo. Assim, temos:

PILHA	BC
63,44	63
1248,51	1249
- 3,8	erro
141,833	erro

Vamos usar o que já aprendemos sobre cálculo, no exemplo que segue. Façamos um programa, em BASIC, que pede dois números, X e Y, e também a operação desejada entre eles (+, -, /, x).

Esses dados podemos guardar assim:

X em 16444 e 16445
Y em 16446 e 16447
operação em 16417

Em linguagem de máquina, fazemos a leitura do número, vamos à RC, retornamos à linguagem de máquina e trazemos a resposta para o BASIC, no par de registradores BC.

A operação escolhida será inserida na RC através de um POKE feito em BASIC na posição do programa onde fica o literal.

Em BASIC, temos:

```
10 PRINT "X=" ;
20 INPUT X
30 PRINT X
40 PRINT "Y=" ;
50 INPUT Y
60 PRINT Y
70 PRINT "OPERAÇÃO E";
80 INPUT A$
90 PRINT A$
100 POKE 16445, INT (X/256)
110 POKE 16444, X — 256 * INT (X/256)
120 POKE 16447, INT (Y/256)
130 POKE 16446, Y — 256 x INT (Y/256)
140 IF A$ = "+" THEN POKE 16529,15
150 IF A$ = "-" THEN POKE 16529,3
160 IF A$ = "/" THEN POKE 16529,5
170 IF A$ = "x" THEN POKE 16529,4
180 PRINT X; A$; Y; "="; USR 16514
```

E, em linguagem de máquina

16514	LD BC, (16444)	ED4B3C40	Lê X
	STACK BC	CD2015	Coloca X na pilha
	LD BC, (16446)	ED4B3E40	Lê Y
	STACK BC	CD2015	Coloca Y na pilha
	RST 28	EF	Entra na RC
16529	(4091) OPERACAO	00	Executa a operação

FIM CALC	34	Sai da RC
UNSTACK BC	CDA70E	Carrega BC com o que está no topo da pilha
RET	C9	Retorna ao BASIC

Curioso, faça a conta:

$$X = -15$$

$$Y = 40 \quad \text{Operação} = +$$

e a máquina indicará erro na RC.

Agora faça:

$$X = -65.000$$

$$Y = 6 \quad \text{Operação} = +$$

e a tela mostrará a resposta 542. Por quê?

Essas aparentes anomalias ocorrem porque, quando entramos com um número negativo e fazemos o POKE desse número, a máquina transforma o número no seu negativo (isto é, complemento binário + 1).

Assim, no primeiro caso, $X = -15$ entrará na posição 16444/16445 como 65521 ($15 = 000F$, o complemento é $FFF0$, somado a 1 dá $FFF1$), que, somado a $Y (=40)$, deveria dar 65561, que faz estourar a capacidade dos registradores BC.

No segundo caso $X = -65.000$ entrará como 536 ($65.000 = FDE8$, o complemento é 0217 , somado a 1 dá 0218), que, somado a 6, dará 542.

NOTA: leve sempre em consideração que as sub-rotinas de STACK e UNSTACK destróem o conteúdo dos registradores, portanto, se for preciso preservá-los faça uso dos PUSHs e POPs necessários.

Já vimos, então, que constitui um grande problema ter que descarregar o resultado de um cálculo feito pela RC em um par de registradores que está limitado a números positivos e menores que 65535,5 (por quê?).

Para contornar esse inconveniente, podemos jogar o número que está no topo da pilha de cálculo para dentro de uma variável definida no BASIC, portanto armazenada a partir do endereço fornecido por VARS. Não existe na ROM uma sub-rotina que faz essa transferência. A rotina abaixo faz isso:

16514 (4082)	LD HL, (VARS)	2A1040	} Carrega DE com o endereço do quinto "byte" ("byte" E) da primeira variável armazenada pela BASIC
	LD BC, 5	010500	
	ADD HL, BC	09	
	PUSH HL	ES	
	POP DE	D1	
	LD HL, (PILFIM)	2A1C40	} Carrega HL com o endereço do quinto "byte" do número no topo da pilha de cálculo
	DEC HL	2B	

LDOR	EDB8)	Transfere os cinco "bytes" da pilha para a variável
INC HL	23)	Corrige o valor de PILFIM
LD (PILFIM), HL	221C40		
RET	C9)	Retorna ao BASIC

Note que, como o valor de PILFIM foi refeito, isto é, apontando agora para uma posição de cinco "bytes" abaixo, a variável transferida foi também eliminada da pilha de cálculo. De forma inversa, a sub-rotina que segue transfere um número armazenado na área das variáveis para o topo da pilha de cálculo. Para realizar essa transferência, precisamos, em primeiro lugar, abrir espaço para a entrada dessa variável. Isso é feito utilizando, na RC, um literal que não vimos ainda: A0, que coloca o número 0 (cinco "bytes") no topo da pilha de cálculo.

A sub-rotina é a seguinte:

16514	RST 28	EF) Abre espaço no topo da pilha de cálculo
	CARREGAR 0	A0	
	FIM CALC	34	
	LD DE, (PILFIM)	ED5B1C40) Carrega DE com o endereço do último "byte" na variável no topo da pilha
	DEC DE	1B	
	LD HL, (VARS)	2A1040) Carrega HL com o endereço do último "byte" da primeira variável armazenada em BASIC
	LD BC, 5	010500	
	ADD HL, BC	09	
	LDOR	EDB8) Transfere os cinco "bytes"
	RET	C9) Retorna ao BASIC

Como exercício, façamos o seguinte: vamos entrar com um número em BASIC (X), transferi-lo para a pilha de cálculo, transferi-lo mais uma vez para a pilha de cálculo, multiplicar ambos (ou seja, elevar o número ao quadrado e transferir o resultado de volta para o lugar da mesma variável (X):

```

10 PRINT "X=";
20 INPUT X
30 PRINT X
40 LET L=USR 16514
50 PRINT "X**2="; X

```

Em linguagem de máquina, temos:

16514	LD A,2	3E02	} Cria contador em A
	(B) PUSH AF	F5	} Salva o acumulador
	RST 28	EF	} Sub-rotina que transfere o valor da variável X para a pilha de cálculo
	CARREGA 0	A0	
	FIM CALC	34	
	LD DE, (PILFIM)	ED5B1C40	
	DEC DE	1B	
	LD HL, (VARS)	2A1040	
	LD BC,5	010500	
	ADD HL,BC	09	} Repete uma vez a rotina anterior
	LDDR	EDB8	
	POP AF	F1	
	DEC A	3D	
	JRZ (A)	2802	
	JR (B)	18E8	
	(A) RST 28	EF	} Eleva X ao quadrado
	MULT	04	
	FIM CALC	34	
	LD HL, (VARS)	2A1040	} Transfere o resultado da pilha para X
	LD BC,5	010500	
	ADD HL,BC	09	
	PUSH HL	E5	
	POP DE	D1	
	LD HL, (PILFIM)	2A1C40	
	DEC HL	2B	
	LDDR	EDB8	} Retorna ao BASIC
	INC HL	23	
	LD (PILFIM),HL	221C40	
	RET	C9	

Quando falamos que a pilha de cálculo era um pouco volátil, queríamos dizer o seguinte: os dados armazenados na pilha de cálculo lá permanecem enquanto o programa estiver rodando em linguagem de máquina ou, então, na Rotina de Cálculo. Ao voltar para o BASIC, a pilha é destruída.

Vamos ver agora como funciona a memória para cálculo. O diagrama da Fig. 6.1 mostra que se tem acesso a essa memória via RC. Essa memória não é afetada pela volta ao BASIC, nem mesmo pela instrução NEW. Para armazenar um dado nessa memória para cálculo o dado precisa estar no topo da pilha de cálculo e o armazenamento não afeta o dado na pilha de cálculo. Quando lemos um dado da memória para cálculo, o dado é colocado no topo da pilha de cálculo e, assim, desloca todos os outros dados dessa pilha para baixo em cinco "bytes". O dado nesse caso é lido e não retirado, pois continua na memória para cálculo.

Os literais para guardar um dado na memória para cálculo e ler um dado da memória para cálculo são:

C0	armazena	um	dado	na	memória	N.º 0
C1	"	"	"	"	"	N.º 1
C2	"	"	"	"	"	N.º 2
C3	"	"	"	"	"	N.º 3
C4	"	"	"	"	"	N.º 4
C5	"	"	"	"	"	N.º 5

E0	lê	o	dado	armazenado	na	memória	N.º 0
E1	"	"	"	"	"	"	N.º 1
E2	"	"	"	"	"	"	N.º 2
E3	"	"	"	"	"	"	N.º 3
E4	"	"	"	"	"	"	N.º 4
E5	"	"	"	"	"	"	N.º 5

Se, de um lado, a memória para cálculo é um lugar indelével para guardar dados, de outro, temos que ficar sempre atentos às Rotinas de Cálculo, pois muitas delas utilizam para a execução de seus cálculos uma ou mais dessas memórias, alterando, assim, valores que tenham sido guardados anteriormente.

Voltando, ainda, à Fig. 6.1, vemos que existe uma comunicação direta da RC com as memórias RAM e ROM. O real significado dessa comunicação será visto ao abordarmos o literal 28.

Agora que já conhecemos a estrutura de cálculo da máquina, podemos abordar, uma a uma, as sub-rotinas disponíveis para uso. No apêndice F, mostraremos uma tabela que contém o literal, o nome da rotina e o endereço da mesma.

AS SUB-ROTINAS DE CÁLCULO 7

Para simplificar a explicação que segue, vamos chamar de X o número que se encontra no topo da pilha de cálculo, de Y o número imediatamente seguinte, e de Z o próximo (se existir).

Os literais estão em hexadecimal. Para que o leitor possa se familiarizar com essa sub-rotina que vamos apresentar, recomendamos que as analise uma a uma, fazendo uso do programa que segue. Esse programa pede, inicialmente, um valor para X e outro para Y. Em seguida, aparece na tela LITERAL =, e entramos com o literal em hexadecimal.

Como resposta, aparece primeiramente a indicação PILHA = XX BYTES, para mostrar o tamanho resultante da pilha de cálculo após a aplicação da sub-rotina chamada pelo literal dado sobre os dois números que estavam na pilha de cálculo X e Y. Note-se que cada cinco "bytes" significa um número, isto é, antes da sub-rotina em questão a pilha tem sempre dez "bytes". Finalmente, aparece o resultado da operação na forma RESULTADO (EM X) =, uma vez que o resultado estará sempre na posição X.

O programa para análise das sub-rotinas de cálculo é o seguinte:

- em BASIC:

```
10 PRINT
20 PRINT
30 PRINT "X=";
40 INPUT X
50 PRINT X
60 POKE 16532,0
70 LET L=USR 16514
80 PRINT "Y=";
90 INPUT X
100 PRINT X
110 POKE 16532,5
120 PRINT "LITERAL=";
130 INPUT A$
140 PRINT A$
150 POKE 16540,16*CODE A$(1)+CO
DE A$(2)-476
```

```

160 PRINT "PILHA=";
170 LET L=USR 16514
180 PRINT PEEK 16417;" BYTES"
190 PRINT "RESULTADO (EM X)";X
200 PRINT
210 GOTO 30

```

• e, em linguagem de máquina:

16514	RST 28	EF	} Coloca a primeira variável numérica no topo da pilha
	CARREGA 0	A0	
	FIM CALC	34	
	LD DE, (PILFIM)	ED5B1C40	
	DEC DE	1B	
	LD HL, (VARS)	2A1040	
	LD BC,5	010500	
	ADD HL,BC	09	} Decisão
	LDDR	ED68	
16531	RST 28	EF	} Rotina para armazenar X na memória número 5
	ARMAZENA MEM	C5	
	TROCA	02	} Realiza a operação desejada
	FIM CALC	34	
	RET	C9	
16538	RST 28	EF	} Calcula o tamanho da pilha em número de "bytes"
	LE MEM	E5	
16540	OPERACAO	00	} Rotina que transfere o número no topo da pilha (X) para a primeira variável definida em BASIC
	FIM CALC	34	
		2A1C40	
		ED5B1A40	
		A7	
		ED52	
		7D	
		322140	
	LD HL, (VARS)	2A1040	
	LD BC,5	010500	
	ADD HL,BC	09	
	PUSH HL	E5	
	POP DE	D1	
	LD HL, (PILFIM)	2A1C40	
	DEC HL	2B	
	LDDR	ED68	
	INC HL	23	
	LD (PILFIM),HL	221C40	
	RET	C9	

Nesse programa, vale a pena observar como é feito o carregamento dos valores X e Y para a pilha. Inicialmente, entra-se com o valor X (primeira variável definida em BASIC) e, na linha 60, fazemos o deslocamento da instrução JUMP RELATIVE (em 16531) igual a 0. Dessa forma, quando for chamada a sub-rotina em linguagem de máquina pela instrução 70, o valor de X será armazenado na posição 5 da memória para cálculo.

Em seguida, é pedido o valor Y, mas o que realmente se faz é entrar com um novo valor para X (linha 90) e agora (em 110) fazemos o deslocamento da instrução "JUMP RELATIVE" igual a 5. Assim, ao chamarmos pela segunda vez a

rotina em linguagem de máquina (linha 170), o programa, ao chegar na posição 16531, saltará para 16538, realizando a operação desejada sobre X e Y. Em seguida, é calculado o tamanho da pilha e o resultado é armazenado em 16417. Finalmente, transfere-se o resultado da operação para a variável X. Note-se, portanto, que a variável X é usada para a entrada dos valores X e Y e para a saída do resultado.

Existem três categorias de operação possíveis na sub-rotina de cálculo:

- operação sobre uma única variável. Ex.: SIN X, LN X, INT X, etc. Em nosso programa de demonstração essa classe de operação será realizada sobre o conteúdo da memória X.
- operação sobre duas variáveis. Ex.: soma, multiplicação, potenciação, etc. Em nosso programa de demonstração serão realizados sobre X e Y, ficando sempre o resultado em X;
- operação de deslocamento de dados. Ex.: armazenar e ler da memória para cálculo, que não podem ser demonstradas em nosso programa, porém que já foram vistas.

Depois de tantos “considerandos”, vamos finalmente às sub-rotinas.

• 01 TROCA

Esta sub-rotina troca de lugar X e Y. A posição dos demais números, se existirem, fica inalterada.

Trata-se de uma sub-rotina útil antes de uma subtração ou divisão, quando se quer inverter os termos.

• 02 APAGA

Esta sub-rotina apaga X e desloca toda a pilha de cálculo de uma posição (cinco “bytes”) para cima. Assim, Y passa a ser X, Z passa a ser Y e assim por diante.

• 03 SUBTRAÇÃO

Esta sub-rotina realiza a operação $Y - X$, colocando o resultado em X. Os valores originais de X e Y desaparecem e o restante da pilha se desloca, uma posição para cima, i. é, Z passa a ocupar o lugar de Y, e assim por diante.

• 04 MULTIPLICAÇÃO

Esta sub-rotina realiza a operação $X * Y$, colocando o resultado em X. Z passa a ocupar o lugar de Y, e assim por diante.

● 05 DIVISÃO

Esta sub-rotina realiza a operação Y/X , colocando o resultado em X. Z passa a ocupar o lugar de Y e assim por diante.

● 06 POTENCIAÇÃO

Esta sub-rotina realiza a operação Y^X , colocando o resultado em X. Z passa a ocupar o lugar de Y e assim por diante.

Se $Y = 0$ e $X = 0$, o resultado será 1;

Se $Y = 0$ e $X > 0$, o resultado será 0;

Se $Y = 0$ e $X < 0$, dá "overflow".

● 07 OU

Esta sub-rotina faz o seguinte:

para $X \neq 0$, retorna com 1;

para $X = 0$, retorna com Y.

O retorno é feito na posição X e a pilha se desloca de uma posição para cima.

● 08 E

Esta sub-rotina faz o seguinte:

para $X \neq 0$, retorna com Y;

para $X = 0$, retorna com 0;

O retorno é feito na posição X e a pilha se desloca de uma posição para cima.

● 0F SOMA

Esta sub-rotina realiza a operação $X + Y$, colocando o resultado em X. Os números originais são perdidos e a pilha sobe de uma posição.

● 18 MUDA SINAL

Esta sub-rotina troca o sinal do número X. Os demais números da pilha permanecem inalterados e na posição original.

● 1C SEÑO

Esta sub-rotina realiza a operação $\text{sen } X$, onde X é interpretado em radiano. O resultado, isto é, $\text{sen } X$, fica na posição X. Os demais números não se alteram.

- **1D COSSENO**

Esta sub-rotina realiza a operação $\cos X$, onde X é interpretado em radiano. O resultado, isto é, $\cos X$, fica na posição X . Os demais números não se alteram.

- **1E TANGENTE**

Esta sub-rotina realiza a operação $\tan X$, onde X é interpretado em radiano. O resultado, isto é, $\tan X$, fica na posição X . Os demais números não se alteram.

- **1F ARCO-SENO**

Esta sub-rotina realiza a operação $\arcsin X$. O resultado, $\arcsin X$, é dado em radianos e fica na posição X . Os demais números não se alteram.

- **20 ARCO-COSSENO**

Esta sub-rotina realiza a operação $\arccos X$. O resultado, $\arccos X$, é dado em radianos e fica na posição X . Os demais números não se alteram.

- **21 ARCO-TANGENTE**

Esta sub-rotina realiza a operação $\arctan X$. O resultado, $\arctg X$, é dado em radianos e fica na posição X . Os demais números não se alteram.

- **22 LOGARITMO NEPERIANO**

Esta sub-rotina realiza a operação $\log_e X$, se $X > 0$. Caso contrário, a rotina pára, dando erro. O resultado é colocado na posição X . Os demais números não são alterados.

- **23 EXPOENTE**

Esta sub-rotina realiza a operação e^X . O resultado é colocado na posição X . Os demais números não são alterados.

- **24 INTEIRO**

Esta sub-rotina calcula a parte inteira de X , arredondando-a sempre para baixo. Ex.: $\text{INT } 8,7 = 8$; $\text{INT } 3,1 = 3$; e $\text{INT } -5,6 = -6$. O resultado é colocado na posição X . Os demais números não são alterados.

- **25 RAIZ QUADRADA**

Esta sub-rotina realiza a operação \sqrt{X} , achando, assim, o valor positivo da raiz quadrada, que é colocado na posição X . Se $X < 0$, a rotina pára e acusa o erro. Os demais números não são alterados.

• 26 SINAL

Esta sub-rotina detecta o sinal de X e dá como resultado em X o seguinte:

se $X > 0$, o resultado é 1;

se $X < 0$, " " " - 1;

se $X = 0$, " " " 0.

Os demais números não são alterados.

• 27 MÓDULO

Esta sub-rotina realiza a operação $|X|$. O resultado é colocado em X. Os demais números não são alterados.

• 28 PEEK

Esta sub-rotina realiza a operação PEEK X. Isto é, o valor de X é trocado pelo conteúdo da posição de memória X. Os demais números não são alterados.

Experimente, no programa de verificação dado, fazer $X = 16514$ e entrar com esse literal, i, é, 28. O resultado será 239, que é o valor de EF, primeira instrução do nosso programa em linguagem de máquina.

• 29 USR

Esta sub-rotina realiza a operação USR X. Em outras palavras, ela substitui o valor no topo da pilha (X) pelo resultado da sub-rotina que tem início em X. O resultado, como sabemos, é o valor armazenado no par de registradores BC. Para testar essa sub-rotina, façamos o seguinte: criemos no nosso programa de verificação a linha 2 REM 0000 e entremos com o seguinte programa:

```
16584      LD BC, 1984      01C007  
          RET              C9
```

Agora façamos, no programa de verificação, $X = 16584$, $Y =$ qualquer, e literal = 29. O resultado será 1984.

• 2C NÃO

Esta sub-rotina realiza a seguinte operação:

se $X = 0$, o resultado será 1;

se $X \neq 0$, o resultado será 0.

Os demais números não são alterados.

• 2D DUPLICAÇÃO

Esta sub-rotina duplica o valor que está no topo da pilha, deslocando Y para a posição Z.

Assim, temos:

antes de duplicar

$$X = A$$

$$Y = B$$

depois de duplicar

$$X = A$$

$$Y = A$$

$$Z = B$$

• 2E Y MÓDULO X

Esta sub-rotina transforma os números em X e Y, da seguinte forma:

$$X \rightarrow \text{INT} (Y/X)$$

$$Y \rightarrow Y - \text{INT} (Y/X)$$

É necessário que X seja diferente de zero, senão dará erro. Os demais números não são alterados.

• 32 MENOR QUE ZERO

Esta sub-rotina detecta o sinal do número colocado em X e o substitui por 1 se o valor de X for negativo, e, por zero, se o valor for maior ou igual a zero. Os demais números não são alterados.

• 33 MAIOR QUE ZERO

Esta sub-rotina detecta o sinal do número colocado em X e o substitui por 1 se esse número for maior que zero, e, por zero, se o número for menor ou igual a zero. Os demais números não são alterados.

• 34 FIM CÁLCULO

Este literal, como já vimos, indica à máquina que acabou a rotina de cálculo.

• 35 ARGUMENTO REDUZIDO

Esta sub-rotina considera que o número em X é um ângulo em radianos e o transforma numa variável V, tal que $-1 \leq V \leq 1$ e satisfazendo ainda a condição $\text{sen } X = \text{sen} (V * \pi/2)$.

Esta sub-rotina é preparatória para as sub-rotinas 1C, 1D e 1E, uma vez que as funções trigonométricas requerem argumentos mínimos para que as séries que as calculam sejam convergentes.

- **36 ARREDONDAMENTO PARA ZERO**

Esta sub-rotina substitui o valor de X pelo seu inteiro, arredondado sempre em direção ao zero. Assim, se $X = -8,7$, o resultado será -8 ; se $X = 5,2$, o resultado será 5 . Os demais números permanecem inalterados.

- **A0 COLOCA ZERO**

Esta sub-rotina coloca no topo da pilha o valor 0, deslocando, assim, todos os números existentes na pilha de uma casa.

Assim, X passa a ser Y, Y passa a ser Z, e assim por diante.

- **A1 COLOCA UM**

Esta sub-rotina coloca no topo da pilha o valor 1, deslocando assim todos os números existentes na pilha de uma casa. Assim, X passa a ser Y, Y passa a ser Z, e assim por diante.

- **A2 COLOCA MEIO**

Esta sub-rotina coloca no topo da pilha o valor $1/2$, deslocando assim todos os números existentes na pilha de uma casa. Assim, X passa a ser Y, Y passa a ser Z, e assim por diante.

- **A3 COLOCA $\pi/2$**

Esta sub-rotina coloca no topo da pilha o valor $\pi/2$, deslocando assim todos os números existentes na pilha de uma casa. Assim, X passa a ser Y, Y passa a ser Z, e assim por diante.

- **A4 COLOCA DEZ**

Esta sub-rotina coloca no topo da pilha o valor 10, deslocando assim todos os números existentes na pilha de uma casa. Assim, X passa a ser Y, Y passa a ser Z, e assim por diante.

- **C0, C1, C2, C3, C4 e C5**

Estas sub-rotinas colocam o número que está no topo da pilha, X, nas posições de 0 a 5 da memória para cálculo. Todos os números da pilha ficam inalterados, inclusive X.

- **E0, E1, E2, E3, E4 e E5**

Estas sub-rotinas colocam os números armazenados nas posições de 0 a 5 da

memória para cálculo, no topo da pilha, deslocando, assim, todos os números lá existentes de uma casa. Portanto X passa a ser Y, Y passa a ser Z, e assim por diante.

Fora as sub-rotinas aqui apresentadas existem outras de uso muito restrito para o programador, razão pela qual não foram analisadas. Estas são, na maioria, sub-rotinas usadas pelo interpretador da máquina para cálculo.

Entre as sub-rotinas não analisadas existem algumas que tratam de "STRINGS" e podem por isso despertar a curiosidade do leitor para seu uso. Não vamos nos aprofundar nesse assunto, mas apenas lembrar que a pilha pode armazenar STRINGS também.

O leitor poderá ter uma idéia do que fazem algumas dessas sub-rotinas entrando com o seguinte programa:

```
10 PRINT "ENTRE COM DADO";
20 INPUT A
30 PRINT A
40 POKE 16508,INT (A/256)
50 POKE 16507,A-256*INT (A/256)
)
60 PRINT "LITERAL=";
70 INPUT A$
80 PRINT A$
90 POKE 16523,16*CODE A$(1)*CO
DE A$(2)-476
100 PRINT "RESULTADO=";USR 1651
4
110 PRINT
120 GOTO 10
```

E, em linguagem de máquina, com o programa:

16514	LD BC, (16507)	ED4B7B40
	STACK BC	CD2015
	RST 28	EF
	STRING	2A
	OPERACAO	00
	FIM CALC	34
	UNSTACK BC	CDA70E
	RET	C9

Esse programa pede a entrada de um número (entre 0 e 65535) e um literal. Em linguagem de máquina, o número em X é transformado em STRING (literal 2A) para aí receber a operação desejada, já como STRING. Assim, se entrarmos com o literal 1A = VAL, obteremos como resultado o próprio número que entramos. Se entrarmos com 19 = CODE, vamos obter como resultado o código do primeiro caracter da STRING. Assim, se o número entrado for 584, obteríamos 33, que é código de "5".

As sub-rotinas de cálculo, como o leitor deve ter verificado, constituem um poderoso instrumento de trabalho para o programador em linguagem de máquina. Devemos ressaltar, no entanto, que seu uso deve ficar restrito aos casos realmente de cálculo que não podem facilmente ser resolvidos ao nível de instruções diretas no Z-80. Isso porque essas rotinas são extremamente longas e complexas, demorando muitas delas um tempo que se torna proibitivo para os programas em linguagem de máquina.

Explicando melhor, se o programador deseja fazer um programa cuja essência é cálculo, dificilmente será vantajoso ir para linguagem de máquina, pois, de um lado, ela não oferece vantagem quanto a recursos de cálculo e, de outro, o ganho em tempo de execução do programa será mínimo e, ainda mais, é bem mais difícil programar em linguagem de máquina do que em BASIC.

Muito embora o tempo gasto para executar uma boa parte das sub-rotinas de cálculo dependa do valor do argumento, isto é, o tempo para calcular $\sin 7 \text{ rad}$ é diferente do tempo gasto para calcular $\sin 0$, vamos assim mesmo mostrar alguns valores típicos dos tempos gastos pelas principais sub-rotinas:

LITERAL	DESCRIÇÃO	TEMPO (10^{-3}s)
0F	soma	3,3
03	subtração	10,0
04	multiplicação	10,0
05	divisão	15,0
26	sinal	2,0
27	módulo	2,0
2D	duplicação	2,0
02	apaga	2,0
A0	coloca zero	2,0
24	inteiro	5,5
28	peek	5,5
A1	coloca 1	5,5
A2	coloca 1/2	5,5
A3	coloca $\pi/2$	5,5
A4	coloca 10	5,5
1C	seno	260
1D	coosseno	270

1E	tangente	540
1F	arco-seno	1.050
20	arco-cosseno	1.050
21	arco-tangente	360
25	raiz quadrada	680
22	logaritmo neperiano	420

Fica claro, portanto, que as sub-rotinas de cálculo são várias ordens de grandeza mais lentas do que as instruções em linguagem de máquina.

Vejamos um exemplo de aplicação das sub-rotinas de cálculo. Seja o caso de se medir o tempo entre dois eventos, utilizando-se a variável FRAMES. O que vamos fazer é um cronômetro que indica o tempo em minutos, segundos e décimos de segundos. Para disparar o cronômetro, basta pressionar uma tecla qualquer. O final do tempo será determinado no instante em que se pressionar uma tecla novamente.

Como já vimos, a variável FRAMES é decrescente e diminui de 60 unidades por segundo. Por isso, a primeira providência que tomamos no programa é carregá-la com 65535 (FFFF). Como o problema exige que se tenha décimos de segundo, optaremos por criar uma variável (A) no início do programa em BASIC a qual, no final da rotina em linguagem de máquina, será carregada com o valor dos segundos e décimos. Já para os minutos, que é número inteiro e menor que 256 [isso porque a variável FRAMES pode, no máximo, contar 18 minutos, isto é, $65535/[60 \times 60]$], vamos usar o "byte" 16417.

A explicação da seqüência dos eventos na parte do programa que usa as sub-rotinas de cálculo será dada na Fig. 7.1, onde se mostra, passo a passo, o que ocorre com a pilha de cálculo.

O programa em BASIC é o seguinte:

```

10 LET A=0
20 RAND USR 16514
30 PRINT AT 11,9; PEEK 16417; "
MIN ";A;" SEG"

```

e em linguagem de máquina:

16514	(A) KSCAN INC L	CDBB02 2C	}	Espera pressionar uma tecla
	JRZ (A) LD HL, 65535 LD (FRAMES), HL	28FA 21FFFF 223440		
	(B) KSCAN INC L	CDBB02 2C	}	Espera soltar a tecla

JRNZ (B)	20FA
(C) KSCAN	CD6B02
INC L	2C
JRZ (C)	28FA
LD BC, (FRAMES)	ED4B3440
STACK BC	CD2015
LD BC, 65535	01FFFF
STACK BC	CD2015
RST 28	EF
TROCA	01
SUBTRAI	03
COLOCA 10	A4
DUPLICA	2D
MULTIPLICA	04
COLOCA 1/2	A2
MULTIPLICA	04
COLOCA 10	A4
SOMA	0F
ARMAZENA	C5
DIVIDE	05
DUPLICA	2D
LE MEM 5	E5
DIVIDE	05
INTEIRO	24
ARMAZENA	C4
LE MEM 5	E5
MULTIPLICA	04
SUBTRAI	03
COLOCA 10	A4
MULTIPLICA	04
INTEIRO	24
COLOCA 10	A4
DIVIDE	05
LE MEM 4	E4
FIM CALC	34
UNSTACK BC	CDA70E
LD A, C	79
LD (16417), A	322140
LD HL, (VARS)	2A1040
LD BC, 5	010500
ADD HL, BC	09
PUSH HL	E5
POP DE	D1
LD HL, (PILFIM)	2A1C40
DEC HL	2B
LDDA	EDB8
INC HL	23
LD (PILFIM), HL	221C40
RET	C9

Espera pressionar uma tecla

Coloca "FRAMES" na pilha

Coloca 65535 na pilha

Coloca minutos no "byte"
16417

Coloca o valor do topo da
pilha para a primeira variável
definida em BASIC

Fig. 7.1

PILHA	LITERAL	COMENTÁRIOS			
<table border="1"> <tr><td>65535</td></tr> <tr><td>FRAMES</td></tr> </table>	65535	FRAMES		Posição inicial da pilha	
65535					
FRAMES					
<table border="1"> <tr><td>FRAMES</td></tr> <tr><td>65535</td></tr> </table>	FRAMES	65535	01 (troca)		
FRAMES					
65535					
<table border="1"> <tr><td>T'</td></tr> </table>	T'	03 (subtrai)	Inverte a posição para a subtração		
T'					
<table border="1"> <tr><td>10</td></tr> <tr><td>T'</td></tr> </table>	10	T'	A4 (coloca 10)	$T' = 65535 - (\text{FRAMES})$	
10					
T'					
<table border="1"> <tr><td>10</td></tr> <tr><td>10</td></tr> <tr><td>T'</td></tr> </table>	10	10	T'	2D (duplica)	Início da criação do número 60)
10					
10					
T'					
<table border="1"> <tr><td>10</td></tr> <tr><td>10</td></tr> <tr><td>T'</td></tr> </table>	10	10	T'	04 (multiplica)	
10					
10					
T'					
<table border="1"> <tr><td>100</td></tr> <tr><td>T'</td></tr> </table>	100	T'	A2 (coloca 1/2)		
100					
T'					
<table border="1"> <tr><td>1/2</td></tr> <tr><td>100</td></tr> <tr><td>T'</td></tr> </table>	1/2	100	T'	04 (multiplica)	
1/2					
100					
T'					
<table border="1"> <tr><td>50</td></tr> <tr><td>T'</td></tr> </table>	50	T'	A4 (coloca 10)		
50					
T'					

10
50
T'

0F (soma)

60
T'

Fim da criação do número 60

C5 (armazena)

60
T'

Armazena 60 na memória número 5

05 (divide)

T

2D (duplica)

Obtém $T = T'/60$, i. é, tempo em segundos

T

E5 (lê mem 5)

60
T
T

05 (divide)

M'
T

24 (inteiro)

Obtém $M' = T/60$ que é o tempo em minutos

M
T

C4 (armazena)

Obtém $M = \text{INTEIRO EM } M'$

M
T

E5 (lê mem 5)

Armazena M na memória número 5

60
M
T

04 (multiplica)

60 x M
T

03 (subtrai)

X

A4 (coloca 10)

Obtém $X = T - 60 \times M$, que é o tempo em segundos descontados os minutos (inteiros)

10
X

04 (multiplica)

10 x X

24 (inteiro)

INT (10 x X)

A4 (coloca 10)

10
INT (10 x X)

0 (divide)

S

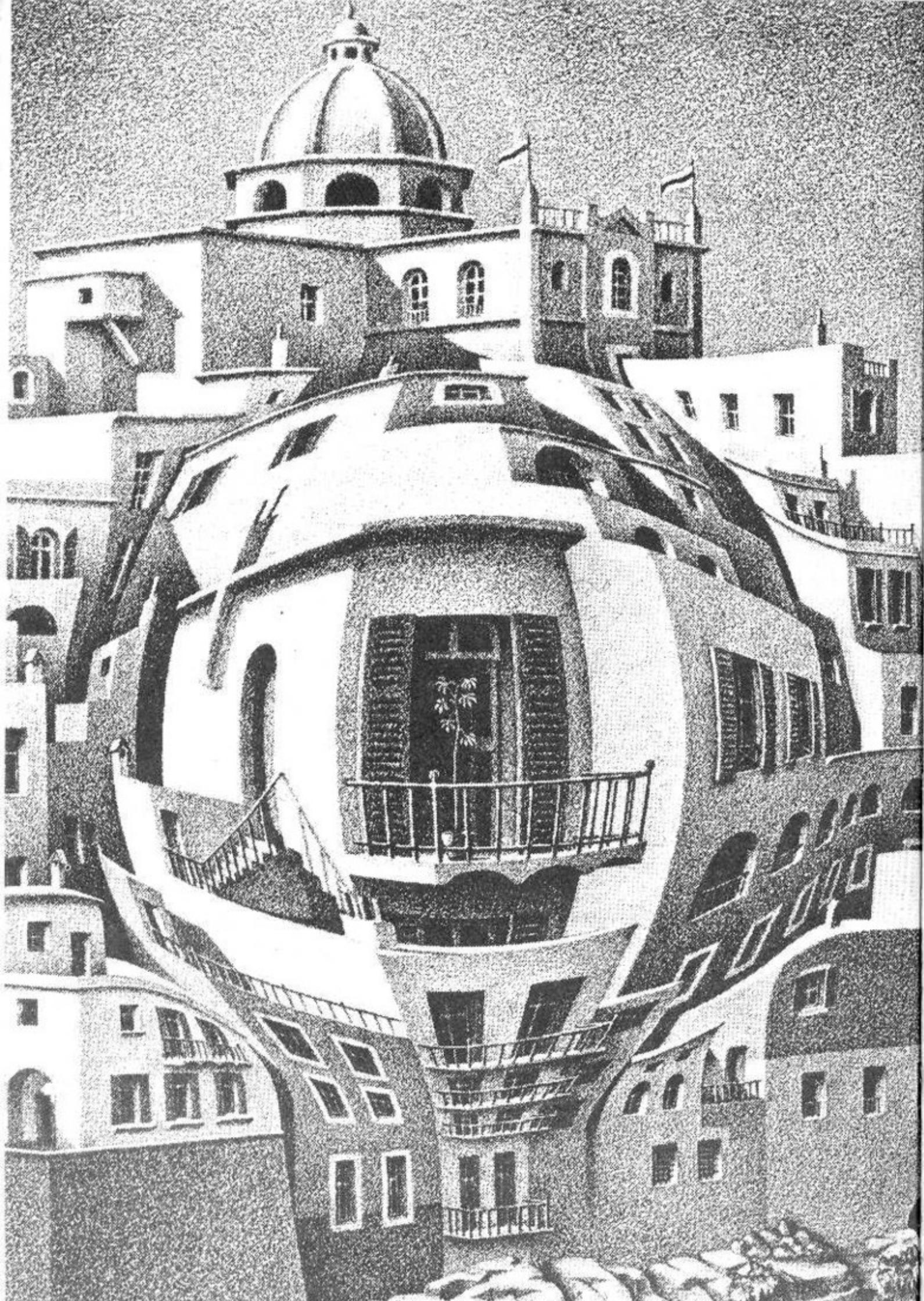
Obtém $S = \text{INT}(10 \times X)/10$ que é a fórmula para se obter décimos de segundos

S
M

E4 (lê mem 4)

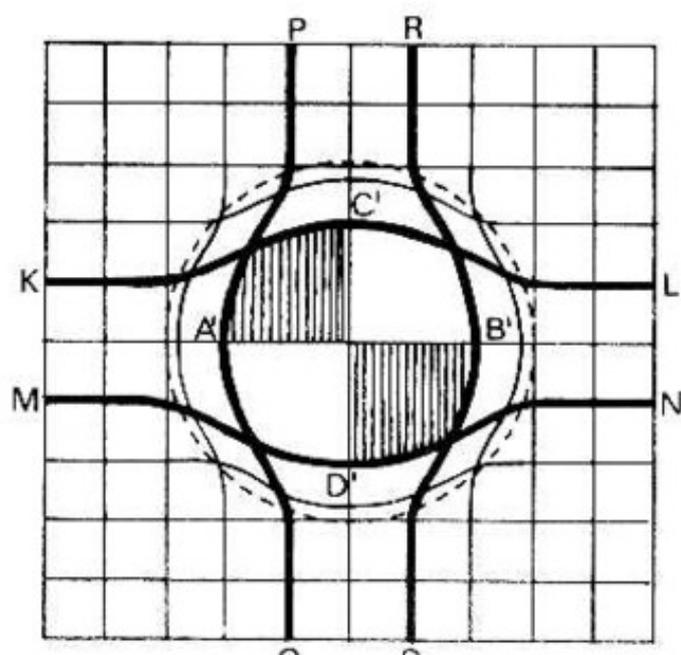
34 (fim calc)

Deixa o resultado na pilha para ser retirado



2ª PARTE

PROGRAMAS: APLICAÇÕES & JOGOS



MESTRE-2

8

Na primeira parte do livro vimos um pequeno programa, escrito em BASIC, que permitia carregar programas escritos em linguagem de máquina numa instrução REM. Mencionamos, também, que, na bibliografia, existe uma boa variedade de sugestões de programas desse tipo. Bem, o que vamos mostrar em seguida é, na verdade, mais um programa para aumentar a lista de “carregadores de linguagem de máquina”. Esse programa apresenta, no entanto, algumas características que o distinguem dos demais, e, a nosso ver, justificam sua existência. Vejamos porquê.

Um programa carregador de linguagem de máquina deve satisfazer às seguintes condições:

- ser de uso simples;
- ocupar pouca memória;
- ser rápido;
- não perturbar o programa que está sendo gerado quer em BASIC quer em linguagem de máquina;
- poder ser gravado;
- ser de fácil apagamento.

Se o programa carregador de linguagem de máquina — que chamaremos de MESTRE — for escrito em BASIC, ele apresentará quatro desvantagens imediatas, que são: lentidão, estar na área de trabalho do programa que está sendo gerado, ser de difícil apagamento (que tem que ser feito linha por linha) e, finalmente, ocupa muita memória.

Por essas razões, o programa MESTRE-2 que vamos mostrar está escrito integralmente em linguagem de máquina. Esse programa deve ser escrito em uma instrução REM e gravado. Para usá-lo, lê-se o programa da fita, roda-se o programa utilizando o comando direto RAND USR 16514, quando, então, aparece o cardápio. Pressionando-se a tecla T, o programa é transferido para depois de RAMTOP. Pode-se, aí, apagar a instrução REM e teremos o programa MESTRE-2 transparente ao BASIC, ou, mesmo, ao programa em linguagem de máquina que vamos escrever.

O MESTRE-2 apresenta os seguintes recursos:

- transfere-se para além da RAMTOP;
- entra com programas escritos em hexadecimal;
- lê programas a partir de qualquer endereço e por um número escolhido de bytes. A leitura é mostrada em hexadecimal;
- insere programas em linguagem de máquina numa rotina já existente;
- transfere-se de volta para qualquer endereço, especificado pelo programador;
- na operação de leitura (chamada de listagem, no programa), entrada de dados e inserção de dados, o programa calcula a soma sintática e coloca no par de registradores BC. Assim, se desejamos essa soma, devemos chamar o programa dando o comando direto PRINT USR X, ou LET L=USR X, onde X é o endereço onde começa o MESTRE-2, que pode ser na posição RAMTOP ou em qualquer endereço para onde tenha sido chamado.

O MESTRE-2 ocupa 376 bytes. Assim, para transferi-lo para além da RAMTOP, temos que alterar essa variável, reduzindo-a de 376 unidades. Então, dependendo do tamanho da memória, temos:

MEMÓRIA RAM	RAMTOP (Original)	RAMTOP (Alterado)
1 K	17408	17032
2 K	18432	18056
16 K	32768	32392
64 K	65536	65160

Vejamos, agora, etapa por etapa, como criar e utilizar o MESTRE-2. Para carregar:

- utilizando o programa em BASIC dado no início do livro, carrega-se o MESTRE-2 numa instrução REM, previamente preparada com 376 caracteres (376 caracteres é equivalente a 11 linhas de caracteres na tela e mais 24 caracteres. Assim, o programa vai começar no endereço 16514.
Grava-se o programa.

Para usar:

- muda-se a RAMTOP de acordo com os valores indicados na tabela. Em seguida, faz-se NEW e NEW LINE.
- Faz-se o LOAD do programa da fita.

- Com o programa já carregado na instrução REM, faz-se o comando direto RAND USR 16514.
- Aparecerá o cardápio das funções, da seguinte forma:
ENT INSER LIST REM RAMTOP BASIC .
 mostrando as seis operações disponíveis. Note-se que, para entrar com uma das opções, pressiona-se a tecla da letra em forma de vídeo invertido.
- Pressiona-se a tecla T para transferir o programa para depois do RAMTOP. Não é preciso usar NEW LINE, uma vez que a leitura do teclado é feita por uma rotina do tipo INKEY\$.
- Apaga-se a instrução REM onde estava o programa fazendo NEW e NEW LINE.

Descrição das funções:

• ENT

Para entrar com um programa é preciso, antes de se pressionar a tecla E, indicar o endereço a partir do qual o programa deve ser escrito. Se vamos colocar numa instrução REM, no início do programa, então o endereço inicial é 16514. Coloca-se, então, o endereço inicial nas posições de memória 16507 e 16508. Assim, se o endereço é 16514, faz-se

POKE 16507, 130

POKE 16508, 64

Faz-se a instrução REM suficientemente grande para conter o programa. Agora, pressiona-se a tecla E, e o programa entra numa sub-rotina de leitura de teclado. Entra-se com o programa em hexadecimal, direto, isto é, sem apertar NEW LINE. Quando se pressionar NEW LINE, o programa pára e volta ao BASIC, trazendo no par de registradores BC a soma-sintática. Se, por qualquer razão, se pressionar NEW LINE após ter-se entrado com um número ímpar de caracteres (como cada byte requer dois caracteres, ao final do programa temos que ter entrado sempre com um número par de caracteres), o programa simplesmente ignorará esse NEW LINE e continuará aguardando entrada de dados.

• INSER

Teclando I, o programa entra na rotina de inserção que necessita de duas informações para rodar: o endereço onde deve começar a inserção e o número de bytes que vão ser inseridos. O endereço deve ser colocado nas posições 16507/16508, na forma já descrita. O número de bytes tem que ser colocado na posição 16417. Isso quer dizer que podemos inserir, de uma vez, no máximo 255 bytes. O programa, antes de receber os dados que serão inseridos, desloca todo o programa

já existente da posição indicada pelo endereço especificado para inserção até o final da instrução REM, de tantas casas quantos são os bytes a serem inseridos. Aqui vale a pena fazer um comentário. Para o programa detectar o final da instrução REM, ele procura um caracter de código 118 (76H), que é o indicativo de fim de instrução, e verifica se o byte seguinte é zero. Isso porque, se o endereço, em BASIC, da instrução seguinte for menor que 256, então o primeiro byte que indica o endereço dessa instrução é zero. Dessa forma, o programa irá detectar como final da linha o primeiro byte 118 (76H), seguido de um byte zero. Portanto, quando se for usar a função inserir, tem-se que ter no programa em BASIC uma instrução, seguindo a REM onde está o programa, cujo número de linha seja inferior a 256. Uma vez feito o deslocamento, o programa entra no modo de entrada de dados, exatamente como já foi descrito.

• LIST

Nesse modo, o programa irá listar, em hexadecimal, o número de bytes especificado a partir do endereço dado. Assim, antes de se rodar o programa, tem-se que colocar na posição 16507/16508 o endereço inicial, e, em 16417, o número de bytes.

Feito isso, ao se pressionar a tecla L, o programa apresenta a listagem na tela. A soma sintática é também calculada e pode ser obtida lendo-se o conteúdo do par de registradores BC na volta ao BASIC. Note-se que não há restrição quanto ao endereço, isto é, de 0 a 65535. Assim, podemos listar a ROM, se assim desejarmos.

• REM

Nesse modo, o programa MESTRE-2 é copiado da posição em que está (que necessariamente é a que começa no RAMTOP) para a posição que tem início no endereço especificado. Dessa forma, antes de rodar o programa, tem-se que carregar as posições 16507/16508 com o endereço desejado.

• RAMTOP

Essa função faz com que o programa MESTRE-2 seja copiado da posição em que se encontra para a posição de memória que se inicia no RAMTOP. Para realizar essa função, é necessário que se tenha o RAMTOP posicionado previamente na posição correta, como, aliás, já foi visto.

• BASIC

Essa função permite que se volte ao BASIC sem realizar qualquer outra função. É comum chamarmos o programa e ao ler o cardápio lembrarmos-nos que nos

esquecemos de colocar o endereço ou o número de bytes. Nesse caso, basta pressionar B que voltamos ao BASIC.

USO DO MESTRE-2

Agora que já vimos como gravar o programa MESTRE-2, bem como suas funções e características vejamos como utilizá-lo:

- muda-se o RAMTOP de acordo com os valores dados na tabela;
- lê-se (LOAD) o programa MESTRE-2 da fita;
- transfere-se o programa para depois o RAMTOP e faz-se NEW;
- cria-se uma instrução REM com tantos lugares quantos são os bytes do programa que se vai entrar. Uma boa regra é fazer a instrução REM 10% maior que o tamanho que achamos necessário. Futuramente, se for preciso inserir dados, já teremos lugar;
- se, por qualquer razão, não terminarmos de entrar com o programa, devemos gravá-lo, e quando formos continuar com a entrada do mesmo, devemos adotar o seguinte procedimento:
 - mudar o RAMTOP de acordo com a tabela, fazer o "load" do MESTRE-2, transferi-lo para depois do RAMTOP e fazer um NEW;
 - fazer o "load" do programa que está sendo escrito e daí para frente continuar normalmente com a entrada do programa.

DESCRIÇÃO DO PROGRAMA

Como o programa MESTRE-2 pode ocupar qualquer posição na memória, as instruções de salto têm que ser todas relativas e, ainda mais, não podemos ter sub-rotinas, uma vez que não existe instrução CALL relativa. A figura 8.1 mostra o diagrama de blocos do programa.

As variáveis envolvidas no programa são:

VARIÁVEL	POSIÇÃO DE MEMÓRIA
K = leitura do teclado	16444
I = endereço	16507/16508
N = número de bytes	16417
E = endereço de onde começa o programa MESTRE-2	16445/16446
S = soma sintática	16447/16448
P = indicador de caracter alto ou baixo	16449
L = área de manobra para cálculo do byte	16450

DIAGRAMA DE BLOCOS

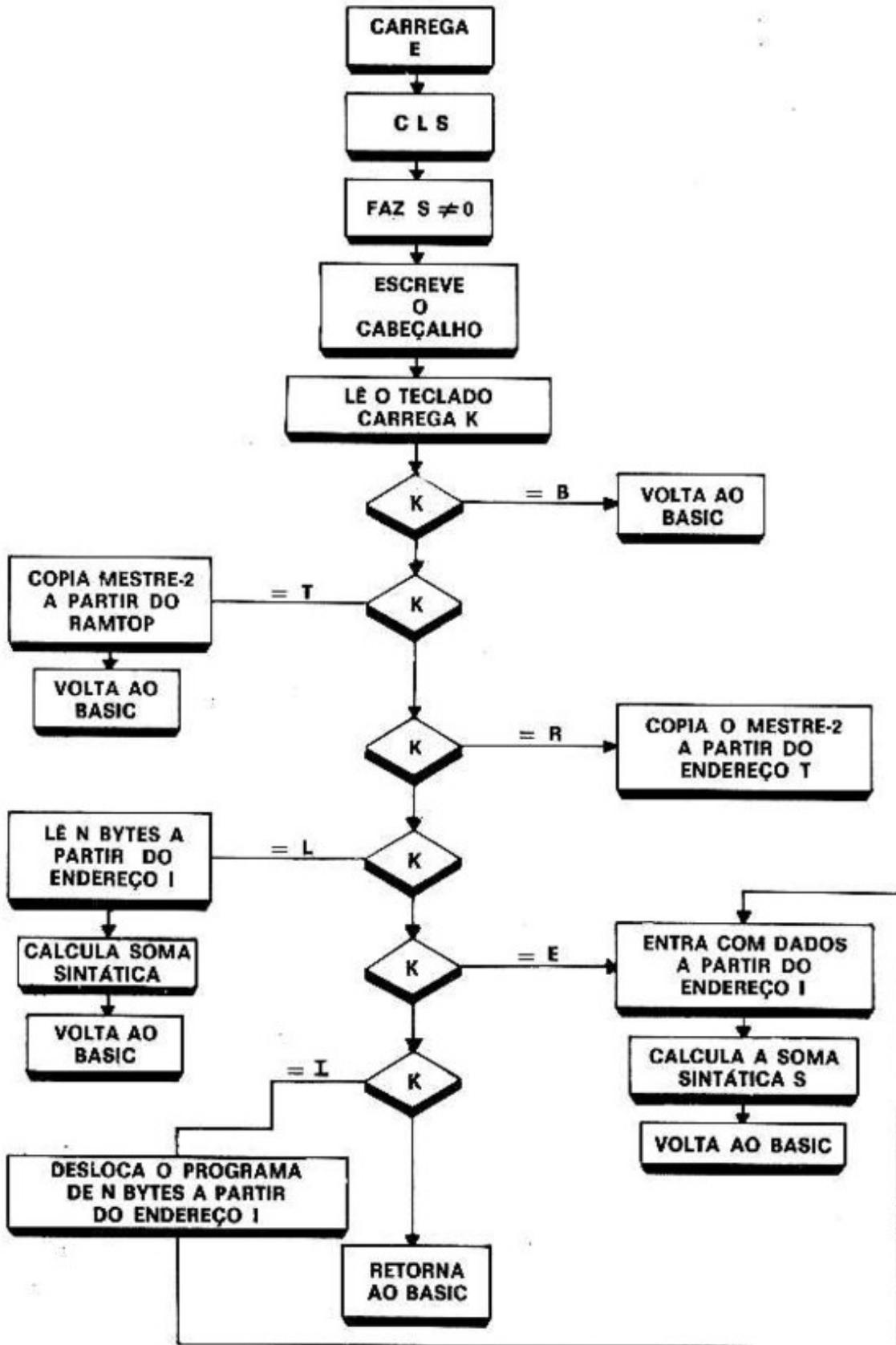


Fig. 8.1

LD (16445) ,BC ED433D40
 CALL CLS CD2A0A
 LD HL,0 210000
 LD (16447) .HL 223F40

Carrega E

Zera S

JR (A)

CABEÇALHO

1840
 AA333900AE33382A
 3700B12E383900B7
 2A3200372632B934
 3500A726382E2000
 2A33292A372A2034
 001D22211C231824
 000000331B273E39
 2A38001D22201D23

(A) LD HL, (16445)
 LD DE, 15
 ADD HL, DE
 LD B, 64

(B) LD A, (HL)
 PRTCHA
 INC HL

DJNZ (B) 10FB

(C) CALL KSCAN
 INC L CDBB02
 2C

JRNZ (C) 20FA

(D) CALL KSCAN
 INC L CDBB02
 2C

JRZ (D) 28FA
 DEC L 2D
 PUSH HL E5
 POP BC C1
 CALL ACHR CDBD07
 LD A, (HL) 7E
 LD (16444) ,A 323C40
 CP A, 39 FE27
 RET Z C8
 CP A, 57 FE39

JRNZ (E) 200D

LD DE, (RAMTOP) ED5B0440
 LD HL, (16507) 2A3D40
 LD BC, 376 017801
 LDIR EDB0
 RET C9

(E) CP A, 55 FE37

JRNZ (F) 200D
 LD HL, (RAMTOP) 2A0440
 LD DE, (16507) ED5B7B40
 LD BC, 376 017801
 LDIR EDB0
 RET C9

(F) CP A, 49 FE31

ESCREVE O CABEÇALHO

LÊ O TECLADO

CARREGA K

VOLTA AO BASIC SE
 FOI TECLADO B

SALTA SE K NÃO É T

COPIA MESTRE-2
 A PARTIR DO RAMTOP

SALTA SE K NÃO É R

COPIA MESTRE-2
 A PARTIR DO ENDEREÇO I

SALTA SE K NÃO É L



JRNZ (G)	203C
LD A, (16417)	3A2140
AND A	A7
JRZ (H)	2601
DEC A	3D
(H) LD D, 0	1600
LD E, A	5F
LD HL, (16507)	2A7B40
PUSH HL	E5
ADD HL, DE	19
PUSH HL	E5
POP DE	D1
POP HL	E1
(J) LD A, (HL)	7E
LD B, 0	0600
LD C, A	4F
PUSH HL	E5
LD HL, (16447)	2A3F40
ADD HL, BC	09
LD (16447), HL	223F40
POP HL	E1
AND A, F0	E6F0
LD B, 4	0604
(I) SRL A	CB3F
DJNZ (I)	10FC
ADD A, 28	C61C
PRCHA	D7
LD A, 15	3E0F
AND A, C	A1
ADD A, 28	C61C
PRCHA	D7
PUSH HL	E5
SBC HL, DE	ED52
POP HL	E1
INC HL	23
JRNZ (J)	200B
LD BC, (16447)	ED4B3F40
RET	C9
(G) CP A, 42	FE2A
JRNZ (K)	2061
LD DE, (16507)	ED5B7B40
(O) PUSH DE	D5
(L) CALL KSCAN	CDBB02
INC L	2C
JRNZ (L)	20FA
(M) CALL KSCAN	CDBB02
INC L	2C
JRZ (M)	26FA
DEC L	2D
PUSH HL	E5
POP BC	C1
CALL ACHR	CD8D07
LD A, (HL)	7E
POP DE	D1
CP A, 118	FE76
JRNZ (N)	200C
LD A, (16449)	3A4140
CP A, 0	FE00
JRNZ (O)	20E0
LD BC, (16447)	ED4B3F40
RET	C9

LÊ OS BYTES A PARTIR
DO ENDEREÇO I

CALCULA A SOMA
SINTÁTICA

TRANSFORMA O BYTE LIDO
EM DOIS CARACTERES
HEXADECIMAIS E COLOCA
NA TELA

SALTA SE K NÃO
É E

DETERMINA O
ENDEREÇO INICIAL

LÊ O TECLADO

VERIFICA SE FOI
TECLADO NEW LINE E PARA
A ENTRADA SE O NÚMERO
DE CARACTERES JÁ
ENTRADOS É PAR
CARREGA BC COM A SOMA
SINTÁTICA

(N) PRTCHA	D7
ADD A, E4	C6E4
LD C, A	4F
LD A, (16449)	3A4140
AND A	A7
JRNZ (P)	2014
LD B, 4	0604
(Q) SLA, C	CB21
DJNZ (Q)	10FC
LD A, C	79
LD (16450), A	324240
LD A, (16449)	3A4140
ADD A, 128	C680
LD (16449), A	324140
JR (O)	18BD
(P) LD A, (16450)	3A4240
ADD A, C	81
LD (DE), A	12
LD B, 0	0600
LD C, A	4F
LD HL, (16447)	2A3F40
ADD HL, BC	09
LD (16447), HL	223F40
LD A, (16449)	3A4140
ADD A, 128	C680
LD (16449), A	324140
INC DE	13
JR (O)	18A3
(K) CP A, 46	FE2E
RET NZ	C0
LD A, 118	3E76
LD HL, (16507)	2A7B40
LD BC, 0	010000
(R) CPIR	EDB1
LD A, (HL)	7E
CP A, 0	FE00
JRNZ (R)	20F9
DEC HL	25
DEC HL	2B
PUSH HL	E5
INC BC	03
LD A, (16417)	3A2140
LD L, A	6F
LD H, 0	2600
ADD HL, BC	09
PUSH HL	E5
POP BC	C1
LD HL, 0	210000
AND A	A7
SBC HL, BC	ED42
PUSH HL	E5
POP BC	C1
LD A, (16417)	3A2140
LD E, A	5F
LD D, 0	1600
POP HL	E1
PUSH HL	E5
AND A	A7
SBC HL, DE	ED52
POP DE	D1
LDDR	EDB8
LD DE, (16507)	ED5B7B40

ESCREVE NA TELA O CARACTER TECLADO
VERIFICA SE O CARACTER É O MAIS OU O MENOS SIGNIFICATIVA NO BYTE
SALTA SE FOR O MENOS SIGNIFICATIVO

CALCULA A CONTRIBUIÇÃO DO CARACTER MAIS SIGNIFICATIVO MULTIPLICANDO-O POR 16.

SOMA O VALOR DO CARACTER MAIS SIGNIFICATIVO COM MENOS SIGNIFICATIVO.

ESCREVE NA MEMÓRIA O BYTE.

ATUALIZA A SOMA SINTÁTICA

ATUALIZA VARIÁVEL P E VOLTA PARA LER OUTRO CARACTER

RETORNA AO BASIC SE O K NÃO É I.

BUSCA O PRIMEIRO BYTE 118

VERIFICA SE O PRÓXIMO BYTE É ZERO

DESLOCA TODOS OS BYTES A DIREITA DO ENDEREÇO I (INCLUSIVE) DE N POSIÇÕES.

JOGO DOS PALITOS



A solução aqui apresentada para o jogo dos palitos é um bom exemplo de como se pode conciliar um programa escrito em BASIC e em linguagem de máquina. Na implementação desse programa utilizou-se o BASIC unicamente para fazer a entrada de dados e apresentar os resultados (PRINT), uma vez que essas funções são bem mais fáceis de serem feitas em BASIC. A implementação do algoritmo solução do problema foi toda feita em linguagem de máquina. A comunicação entre as duas linguagens é feita através de PEEKs e POKEs.

Foi mencionado no início do livro, quando se avaliava a conveniência ou não de se usar linguagem de máquina, que certos tipos de problema — que requeiram o uso de álgebra de Boole — podiam ser facilmente resolvidos usando-se linguagem de máquina. No presente caso, é muito interessante a forma como se implementa a solução utilizando-se a instrução "EXCLUSIVE OR".

Veamos, inicialmente, de que consiste o jogo. Trata-se de um jogo entre duas pessoas (no caso, o computador é uma delas). Inicialmente, o jogador que começa o jogo coloca palitos numa mesa, formando filas, em princípio quantos palitos quiser por fila e quantas filas desejar. Em seguida, o outro jogador retira quantos palitos quiser de uma fila, mas somente de uma fila, e necessariamente um palito, no mínimo. Em seguida, é a vez do outro jogador, que deve retirar palitos e assim por diante. Perde o jogo o jogador que retirar o último palito.

Vamos analisar agora a forma como o computador deve proceder para analisar a sua jogada e daí tirar um algoritmo. Para facilitar a análise, vamos definir algumas variáveis do problema que serão depois utilizadas no programa:

W = número de filas com, no mínimo, um palito

$N(j)$ = número de palitos na fila j

P = número da fila com maior número de palitos (se existirem várias filas nessa condição, pega-se uma qualquer)

S = número da fila com a segunda maior contagem de palitos (aqui também podem existir várias)

Se $W = 1$ e $N(P) > 1$, a jogada é simplesmente fazer $N(P) = 1$ e o jogo está ganho. Obviamente, se recebemos (o computador) $W = 1$ e $N(P) = 1$, estamos perdidos.

Se, agora, $W > 1$ e $N(S) = 1$, isso quer dizer que temos mais de uma fila com palitos e todas têm 1 palito, exceto uma, a fila P, que pode ter mais que um palito.

Se $N(P) = 1$ também, então não há nada a pensar, pois só nos resta tirar um palito de uma fila qualquer e, se W for par, vamos ganhar o jogo, se ímpar, perder.

Se $N(P) > 1$, tiraremos todos os palitos da fila P se W for par, pois aí deixaremos para o adversário um número ímpar de filas com um palito. De outro lado, se $N(P) > 1$ e W ímpar, deixaremos um palito na fila P e caímos no caso anterior.

Esses casos vistos são fáceis de serem analisados (pelo ser humano) e constituem a parte final do jogo. É interessante ressaltar que, se num determinado momento do jogo um jogador estiver numa posição ganhadora (e veremos adiante como saber isso) e não cometer deslize algum, ele irá ganhar o jogo, isto é, jogando corretamente não se perde uma posição ganhadora. Do outro lado, se se estiver numa posição perdedora e o adversário cometer um único erro, é sempre possível passar-se à posição ganhadora e aí ganhar o jogo.

Bem, quando o número de filas e de palitos aumenta, o poder de análise do homem diminui assustadoramente, ao ponto de, com quatro filas e, no mínimo, três palitos por fila, já fica praticamente impossível ao homem analisar o jogo. Não vamos aqui provar que o algoritmo que segue é a solução do problema, mas vamos tentar induzir o leitor a aceitá-lo.

Fora os casos já analisados, a unidade mínima de configuração dos palitos que o algoritmo resolve é para $W = 2$, $N(P) > 2$ e $N(S) = 2$. Se, diante dessa configuração, deixarmos duas filas com dois palitos cada, teremos o jogo ganho, como o leitor poderá facilmente deduzir.

É fácil ver que, para o caso $W = 2$, o algoritmo é simplesmente deixar as duas filas com o mesmo número de palitos, fazendo sempre $N(P) = N(S)$, até que $N(S) = 0$ ou 1, quando caímos nos casos vistos inicialmente e para os quais sabemos como jogar para ganhar.

Ora, se é esse o algoritmo, para $W = 2$, podemos entender porque, estando-se numa posição ganhadora, não se perde mais essa condição, pois, se deixarmos sempre as duas filas com o mesmo número de palitos, é impossível ao adversário nos deixar duas filas com o mesmo número de palitos. Se tivermos $W > 2$, a posição ganhadora é mantida deixando-se sempre quantidades pares de mesmo número de palitos nas filas. Isso é feito escrevendo-se o número de palito de cada fila em forma binária, somando-se os "bits" de mesma ordem de todas as filas. Se a nossa jogada permite deixar pares as somas em todas as ordens, estaremos em posição ganhadora. E, ainda, se recebermos uma configuração onde a soma não é par em alguma ou algumas das ordens, é sempre possível deixá-la par. Essa é a forma de sabermos se estamos ou não numa posição ganhadora.

Vejamos um exemplo com pequeno número de palito e somente três filas — mas já razoavelmente complexo para análise (do homem) —. Imaginemos que é a nossa vez de jogar e temos na mesa:

IIII
 IIIII
 III

A representação binária do número de palitos é:

1 0 0
 1 0 1
 0 1 1

PIP

A soma das colunas de ordem 0 e 2 (ordem 0 é a do dígito menos significativo do número binário, isto é, o que fica mais à direita) é par e a de ordem 1 é ímpar. Isso quer dizer que estamos numa posição ganhadora e temos que fazer uma jogada que transforme a soma em par na coluna de ordem 1. Para tal, basta mudar o "bit" de ordem 1 da terceira fila (a de baixo), isto é, passá-lo para 0 e ficaremos então com:

IIII
 IIIII
 I

A forma como se implemente esse algoritmo no programa é criando um "byte" "soma binária", que tem em cada "bit" o resultado da soma de todos os "bits" de mesma ordem, sendo que, se ele é 0, quer dizer que a soma é par; se é 1, trata-se de uma soma ímpar. Para podermos fazer essa soma "bit" a "bit", sem esbarrar com o problema de "carry" de uma ordem para outra, fazemos inicialmente SB (soma binária) = 0. Aí fazemos um OU EXCLUSIVO com cada "byte" representativo do número de palitos de cada fila. O valor final de SB indica (pelos 1s) em que ordem temos que fazer alteração ou alterações.

No caso do exemplo dado, teríamos:

Fila n.º	Palitos	"Byte"
1	IIII	100
2	IIIII	101
3	III	011

Faz-se, inicialmente, SB = 0 e aí se aplica o "OU EXCLUSIVO", com os três "bytes", como segue:

	Valor inicial	1.º "byte"	2.º "byte"	3.º "byte"
SB	000	000	100	001
"byte"		100	101	011
OU EX		100	001	010

Portanto o valor final de SB = 010.

Em seguida, utiliza-se novamente o "OU EXCLUSIVO" para determinar a fila que terá que ser alterada, pois, fazendo-se o "OU EXCLUSIVO" com o "byte" de uma fila, faz-se mudar no byte" da fila de 0 para 1, ou de 1 para 0 os "bits" das ordens onde a SB for 1. Verifica-se depois se o número resultante deu menor que o anterior ou maior. Se for maior, passa-se para outra fila. Continuando no nosso exemplo, teríamos:

	Inicial	1.º "byte"	2.º "byte"	3.º "byte"
SB	010	010	010	010
"byte"	—	100	101	011
OU EX		110	111	001

Vemos que a solução — nesse caso, única — é tirar um palito da terceira fila.

Se a soma binária na nossa vez de jogar for 0, quer dizer que estamos numa posição perdedora. A melhor jogada nesse caso é tirar um único palito (por exemplo, da fila P, onde sabemos existir mais de um palito, procurando-se esticar ao máximo o jogo, na esperança de que o nosso adversário tenha, assim, mais oportunidades de cometer um erro e nos dar a posição ganhadora para sempre.

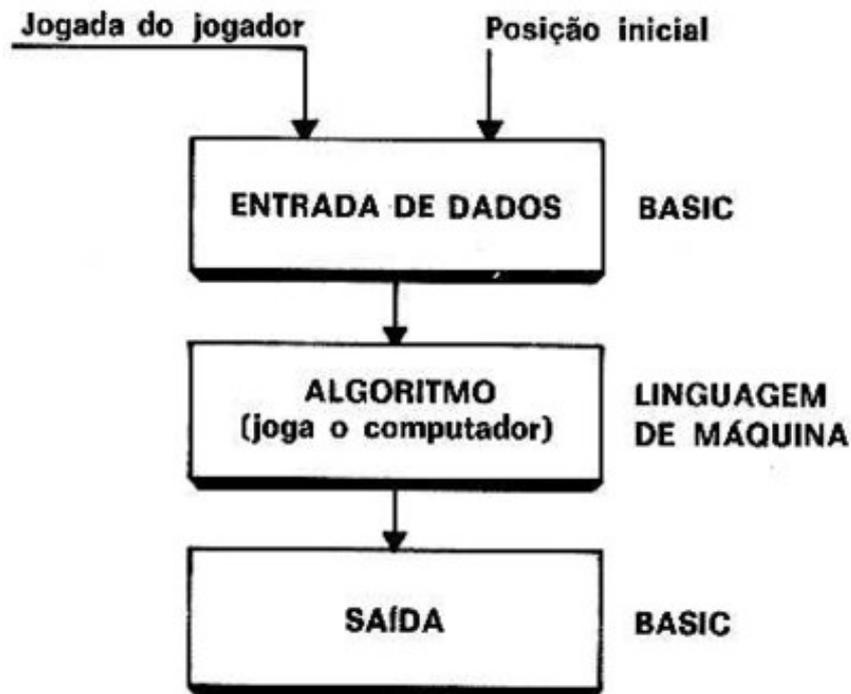
- O programa.

Na Fig. 9.1 temos a representação simbólica do programa, mostrando as partes escritas em BASIC e em linguagem de máquina. A Fig. 9.2 já mostra como o algoritmo foi implementado em linguagem de máquina.

As variáveis utilizadas pelo programa são:

NOME	LOCAL DE ARMAZENAMENTO		DESCRIÇÃO
	Dec	Hexa	
W	16444	403C	Número de filas com no mínimo um palito.
EP	16445	403D	Endereço ("byte" menor) da fila P.
ES	16446	403E	Endereço ("byte" menor) da fila S.
	16449	4041	Número de palitos nas quinze filas possíveis.
	16463	404F	
J	16417	4021	Endereço ("byte" menor) da fila onde foi feita a jogada.

Fig. 9.1



Quanto a essas variáveis, temos o seguinte a comentar:

- por questões de limitação de tela, o programa aceita no máximo quinze filas e vinte e cinco palitos por fila;
- utilizamos para guardar as variáveis a área da RAM reservada ao "buffer" da impressora, que comporta no máximo 32 "bytes". Por essa razão, os três endereços aqui definidos começam sempre com 40XX (em hexa) e, portanto, só precisamos guardar um "byte" por endereço, que é o "byte" menos significativo;
- a variável J contém o "byte" menos significativo do endereço da fila onde o computador faz a jogada. A idéia é a de indicar na tela a linha em que a máquina fez a jogada. Essa variável não foi usada no programa em BASIC por problema de memória na configuração 2K. Se o leitor desejar, poderá melhorar o programa utilizando essa variável.

Na Fig. 9.2 indicamos por (BC) o conteúdo do par de registradores BC. Isso é feito pois o programa em linguagem de máquina é chamado por uma instrução GOTO USR X. Assim, dependendo da jogada, já vamos para o programa em BASIC referente à jogada. Assim, na linha 300 do BASIC, temos a mensagem "PERDEU". Na linha 400, temos a mensagem "GANHEI" e, na linha 1000, altera-se o número de palitos de acordo com a jogada.

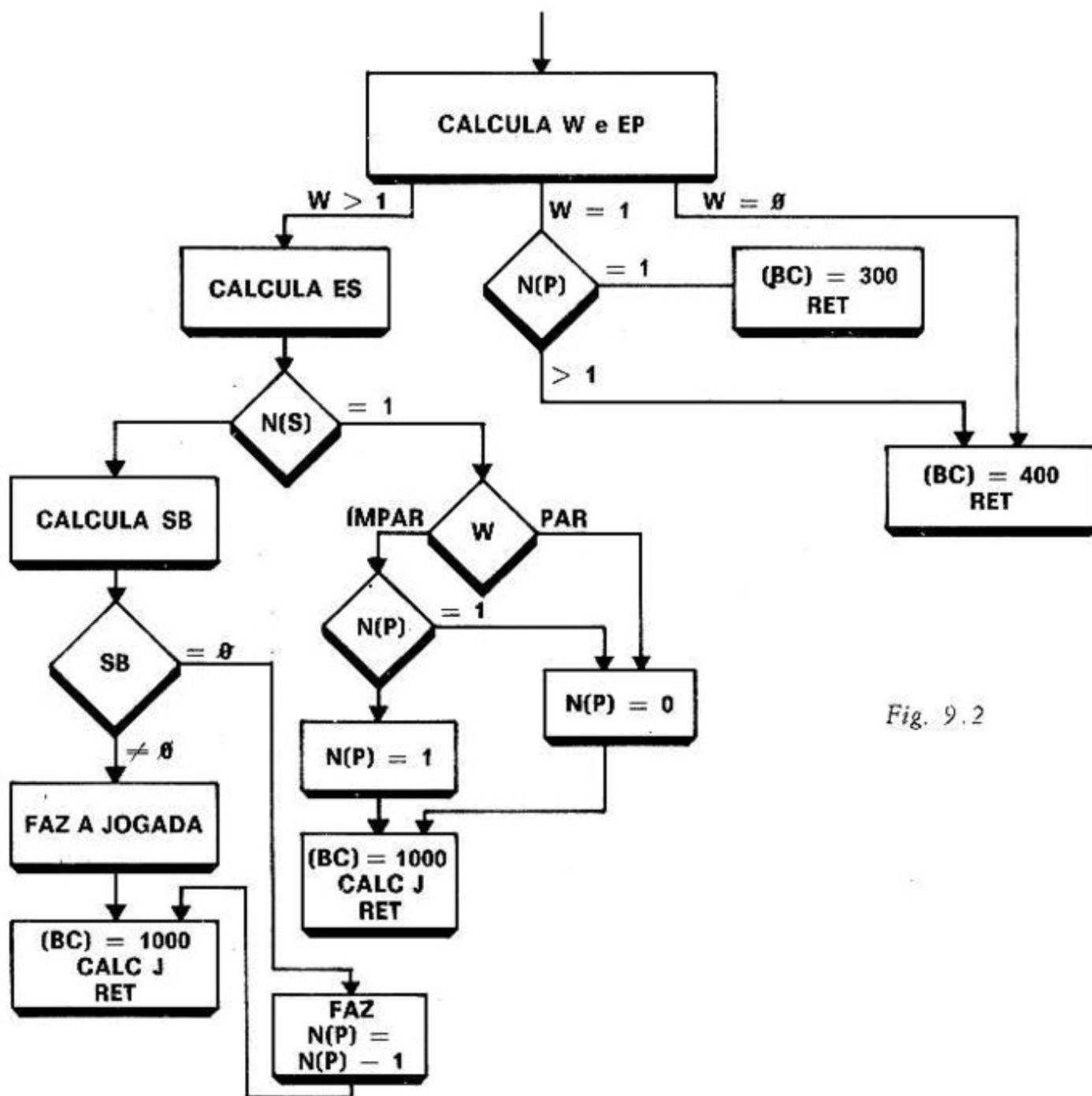


Fig. 9.2

Para jogar, coloca-se o computador no modo SLOW e faz-se RUN. Aparece na tela "LINHAS =", quando, então, devemos entrar com o número de linhas de palitos que queremos jogar. Em seguida, a tela mostra "N.º PALITOS" e, na linha de baixo, o número 1, pedindo para entrarmos com o número de palitos da linha 1. Feito isso, aparece 2, e entramos com o número de palitos da linha 2, e assim até a última linha. A tela então se apaga e mostra a configuração dos palitos. Nesse ponto, a máquina entende que é ela que deve jogar, uma vez que nós escolhemos o número de linhas e de palitos. Para fazer a máquina jogar, pressiona-se "NEW LINE". Após a jogada da máquina, aparece na parte de baixo da tela a mensagem "LINHAS, N.º

PALITOS", quando devemos fazer a nossa jogada, entrando primeiro com o número de linhas e, em seguida, com o número de palitos que devem ficar nessa linha.

O programa em BASIC:

```
10 LET Z$=""
20 RAND USR 16514
30 PRINT "LINHAS=";
40 INPUT L
50 IF L > 15 OR L = 0 THEN GOTO 40
60 PRINT L
70 PRINT "Nº PALITOS"
80 FOR J=1 TO L
90 PRINT J,
100 INPUT X
110 IF X > 25 THEN GOTO 100
120 POKE (16448+J), X
130 PRINT X
140 NEXT J
150 CLS
160 PRINT AT 1,8;"JOGO DOS PALI
TOS"
170 FOR J=1 TO L
180 PRINT AT 4+J,1;J;TAB 5;
190 GOSUB 2000
200 NEXT J
210 GOSUB 3000
220 GOTO USR 16526
300 PRINT "PERDI"
310 STOP
400 PRINT "GANHEI"
410 STOP
1000 LET J=PEEK 16417-64
1010 PRINT AT 4+J,5;Z$
1020 PRINT AT 4+J,5;
1030 GOSUB 2000
1040 PRINT AT 21,3;"LINHA","Nº PA
LITOS"
1050 INPUT J
1060 INPUT X
1070 IF X >=PEEK (16448+J) THEN G
OTO 1060
1080 POKE (16448+J), X
1090 PRINT AT 4+J,5;Z$
1100 PRINT AT 4+J,5;
1110 GOSUB 2000
1120 GOTO 210
2000 IF PEEK (16448+J)=0 THEN GO
TO 2040
2010 FOR I=1 TO PEEK (16448+J)
2020 PRINT "#";
2030 NEXT I
```

```

2040 RETURN
2050 IF INKEY$ <> "" THEN GOTO 3000
3010 IF INKEY$ = "" THEN GOTO 3010
3020 RETURN
8990 STOP
9000 SAVE "PALITOS"
9010 SLOW
9020 RUN

```

A rotina em linguagem de máquina, que segue, ocupa 210 "bytes".

16514	LD A,0	3E00	}	Zera as quinze filas de palitos	
	LD B,15	06FF			
	LD HL,16449	214140			
(A)	LD (HL),A	77			
	INC HL	23			
	DJNZ (A)	10FC			
	RET	C9			
	LD A,0	3E00			
	LD (16444),A	323C40			} W = 0
	LD C,A	4F			
	LD HL,16449	214140	}	Conta o número de filas com, no mínimo, um palito, i. é, calcula W e calcula EP	
	LD B,15	060F			
(C)	LD A,(HL)	7E			
	AND A	A7			
	JRZ (B)	2812			
	PUSH AF	F5			
	LD A,(16444)	3A3C40			
	INC A	3C			
	LD (16444),A	323C40			
	POP AF	F1			
	CP A,C	B9			
	JP M (B)	F8AF40			
	LD C,A	4F			
	LD A,L	7D			
	LD (16445),A	323D40			
(B)	INC HL	23			
	DJNZ (C)	10E7			

LD A, (16444)	3A3C40	} Lê W
AND A	A7	
JRNZ (D)	2004	} Se W = 0 faz (BC) = 400
LD BC, 400	019001	
RET	C9	
(D) CP A, 1	FE01	} Compara W com 1
JRNZ (E)	2013	
LD H, 64	2640	
LD A, (16445)	3A3D40	
LD L, A	6F	
LD A, (HL)	7E	
CP A, 1	FE01	} Se W = 1 e P = 1, então faz (BC) = 300 Se W = 1 e P > 1, então faz (BC) = 400
JRNZ (F)	2004	
LD BC, 300	012C01	
RET	C9	
(F) LD BC, 400	019001	
RET	C9	
(E) LD H, 64	2640	
LD A, (16445)	3A3D40	
LD L, A	6F	
LD E, (HL)	5E	
PUSH HL	E5	
LD A, 0	3E00	
LD (HL), A	77	
LD B, 15	060F	
LD HL, 16449	214140	
LD C, 0	0E00	} Calcula ES e coloca no registrador C a variável N(P)
(H) LD A, (HL)	7E	
CP A, C	B9	
JP M (G)	FAEF40	
LD C, A	4F	
LD A, L	7D	
LD (16446), A	323E40	
(G) INC HL	23	
DJNZ (H)	10F3	



LD A,C	79	} Verifica se N(S) = 1
CP A,1	FE01	
JRNZ (I)	2029	
LD A,(16444)	3A3C40	
BIT 0,A	CB47	
JRZ (J)	2814	
LD A,(16445)	3A3D40	
CP A,1	FE01	
JRZ	2800	
LD H,64	2640	
LD L,A	6F	} Se N(S) = 1, W é ímpar e então faz N(P) = 1 se N(P) > 1 faz N(P) = 0 se N(P) = 1
LD(16417),A	322140	
LD A,1	3E01	
LD(HL),A	77	
(K) LD BC,10000	01E803	
RET	C9	
(J) LD A,(16445)	3A3D40	
LD(16417),A	322140	
LD L,A	6F	
LD H,64	2640	
LD A,0	3E00	} Calcula SB, i. é, a soma binária
LD(HL),A	77	
JR (K)	18EE	
(I) LD B,15	060F	
LD HL,16449	214140	
LD C,0	0E00	
(L) LD A,(HL)	7E	
XOR C	A9	
LD C,A	4F	
INC HL	23	
DJNZ (L)	10FA	} Verifica se SB = 0 ou não
LD A,C	79	
AND A	A7	
JRNZ (M)	200A	

	LD A, (16445)	3A3D40	} Se SB = 0, carrega J e decrementa N(P)
	LD (16417), A	322140	
	LD L, A	6F	
	DEC (HL)	35	
	JA (N)	1013	} Se SB ≠ 0, procura a fila a ser alterada e a altera
(M)	LD L, 65	2E41	
	LD B, 15	060F	
(P)	LD A, (HL)	7E	
	LD D, A	57	
	XOR C	A9	
	CP A, D	BA	
	JP M (O)	FA4B41	
	INC HL	23	
	DJNZ (P)	10F6	
(O)	LD (HL), A	77	} Carrega J
	LD A, L	7D	
	LD (16417), A	322140	
(N)	LD BC, 10000	01E803	} Faz (BC) = 10000
16723	RET	C9	

UM PLOT RÁPIDO 10

I. INTRODUÇÃO

As funções PLOT e UNPLOT são extremamente importantes para a geração de gráficos e em jogos animados, pois não só utilizam o "pixel" — que é o elemento de maior definição gráfica —, como são comandadas pelo par de coordenadas cartesianas X e Y, o que simplifica e facilita sua utilização.

Entretanto a utilização do PLOT/UNPLOT em BASIC é dificultada pela lentidão com que o "pixel" é escrito ou apagado, não permitindo, assim, movimentos rápidos. Isso torna impraticável o uso dessa função quando se trata de jogos ou simulação de movimento em tempo real.

Uma maneira de se resolver essa dificuldade é utilizar a sub-rotina de PLOT/UNPLOT da ROM, que está no endereço 0BB2, chamando-a, evidentemente, por um programa em linguagem de máquina. Infelizmente essa sub-rotina, bem mais rápida que a função em BASIC, é ainda bastante lenta para uma razoável gama de aplicações.

A matéria que segue visa justamente contornar esse inconveniente, apresentando um programa em linguagem de máquina ainda mais rápido do que aquele da sub-rotina do ROM, sendo útil, portanto, para uso em jogos e muitas outras aplicações.

Antes de entrarmos em detalhes sobre sua utilização, mostraremos, através da Fig. 10.1, uma comparação entre os tempos gastos para encher a tela com o "pixel" (isto é, $64 \times 44 = 2.816$ elementos), utilizando-se o BASIC, a sub-rotina da ROM e o programa ora abordado — chamado PLOT RÁPIDO —, para os casos SLOW e FAST:

	SLOW	FAST
BASIC	238	37
ROM	67	11
PLOT RÁPIDO	3,6	0,6

Fig. 10.1: Tempos em segundos para gerar 2.816 elementos "pixel".

II. USO DO PROGRAMA PLOT RÁPIDO

O programa PLOT RÁPIDO, listado no capítulo ocupa 74 "bytes" e está apresentado como sendo uma sub-rotina, isto é, deve ser chamado através de uma instrução "CALL".

Antes do programa PLOT RÁPIDO, tem-se um pequeno programa, também em forma de sub-rotina, que faz abrir a tela (em 22 linhas). Esse programa é necessário para quem for utilizar o PLOT RÁPIDO em computadores com menos de 3,5K de RAM.

O usuário, então, ao escrever o seu programa, coloca a sub-rotina do PLOT RÁPIDO (ou ambas, se tiver menos de 3,5K de RAM) aonde achar mais apropriado e, ao chamar essa sub-rotina, deve ter o registrador B carregado com a ordenada Y, com valores obrigatoriamente entre 0 e 43; o registrador C, com a abcissa X, com valores entre 0 e 63; e o acumulador A, com a informação se é PLOT ou UNPLOT, com 64 (40H) para o primeiro caso e 0 para o segundo. Muito cuidado deve ser tomado para não se colocar nos registradores B e C coordenadas fora dos limites permitidos (acima indicado), assim como no acumulador só se podem colocar, ao se chamar essa sub-rotina, os valores 0 ou 64.

Não há no programa uma rotina para verificar a validade das coordenadas, ou do valor do acumulador, simplesmente para se reduzir a sub-rotina ao mínimo e, com isso, ganhar velocidade.

É importante notar que essa sub-rotina não altera os registradores (isso pode ser visto facilmente no programa, já que as primeiras instruções são PUSHs e as últimas, POPs), o que é muito bom para o programador. Vale a pena salientar mais uma vez que, para se utilizar essa sub-rotina, é necessário que a memória da TELA esteja expandida, pois a sub-rotina faz o PLOT/UNPLOT através de POKEs nos endereços da tela. Portanto, quando o computador não fizer essa expansão sozinho, tem-se que abrir a tela primeiro. Daí a sub-rotina de 13 "bytes" com endereço inicial em 16514, que o usuário aloca aonde quiser.

Deve-se frisar que esse programa foi desenvolvido exclusivamente para gerar PLOT e UNPLOT. Assim sendo, se efetuarmos um PLOT ou UNPLOT em uma região da tela onde já exista um caracter não consistente com o PLOT, poderá aparecer um caracter estranho. Para se entender a razão disso, é preciso acompanhar a descrição do programa, que vem a seguir.

III. DESCRIÇÃO DO PROGRAMA

A partir do endereço 16514 e ocupando 13 "bytes", tem-se uma pequena sub-rotina para abrir a tela, isto é, alocam-se 704 "bytes" (32 colunas \times 22 linhas) da memória na região destinada à função PRINT. Dessa forma, quando se quiser fazer um PRINT, pode-se fugir das rotinas da ROM — que verificam se as coordenadas do PRINT são válidas e se aquela posição específica do PRINT já tem uma

posição da memória RAM reservada para esse fim — e, assim, fazer diretamente um POKE de qualquer caracter, em qualquer posição da tela. Como se sabe, o programa-monitor é responsável pela administração das regiões da memória, deslocando-as, se necessário, à medida que se aumenta ou se diminui um programa em BASIC, ou se criam variáveis, ou se usa mais ou menos PRINT. Ao se alocarem todas as posições da memória correspondentes à tela, os PRINTs e os PLOTs feitos não interferirão mais com o programa-monitor.

Todas essas considerações só se aplicam quando o computador tiver menos que 3,5K de RAM.

A seguir descrevemos como foi feita essa sub-rotina:

16514	LD A,0	3E00	Carrega o acumulador com 0, que é espaço para a função PRINT.
16516	LD C,4	0F04	Usa-se o registrador C de contador, pois serão feitos 4 vezes 176 PRINTs para se abrirem os 704 espaços.
ⓑ 16518	LD B,176	06B0	Carrega-se B como contador de "loop" para a função DJNZ.
Ⓐ 16520	PRINT CHR\$	D7	Usa-se a sub-rotina do ROM de "PRINT CHARACTER".
16521	DJNZ	0FD	Fecha o "loop", voltando para 16520 até que o contador B chegue em zero.
16523	DEC C	0D	Decrementa o contador C
16524	JRNZ	20F8	Volta para 16518 C ≠ 0, ou seja, faz 4 vezes o "loop" de 176 PRINTs.
16526	RET	C9	Volta ao BASIC após 704 PRINTs.

Analisemos, então, a sub-rotina do PLOT, acompanhando o diagrama de blocos mostrado na Fig. 10.3 e o esquema apresentado na Fig. 10.2.

O programa baseia-se no fato de que o "pixel" nada mais é do que um quadrante de uma posição PRINT. Assim, na Fig. 10.2, se o quadrante 0 for preto e os demais em branco, teremos o caracter de código 1 (veja tabela de caracteres do computador). Se o quadrante que estiver em preto for o 2, teremos, então, o caracter de código 4. Se agora tivermos em preto os quadrantes 1 e 2, teremos o caracter de código. Isso tudo pode parecer confuso, mas realmente não o é, se considerarmos os números dados aos quadrantes na Fig. 10.2 como sendo as posições onde existem "bits" "1" no código do caracter. Assim podemos criar os oito primeiros códigos de caracteres, isto é, os caracteres de código 0 a 7.

0	1
2	3

Fig. 10.2

Os quadrantes do PLOT dentro de uma posição PRINT

Se, no entanto, o quadrante 3 estiver ocupado, a regra muda um pouco, uma vez que o código passa a ter um "bit" "1" na sétima posição do "byte" (128 em decimal) e seus quatro "bits" menos significativos passam a ser o complemento binário do código criado usando-se a regra dos quadrantes. Assim, se o quadrante 3 for preto e os demais brancos, temos um "bit" "1" na posição 7 e, nas posições menos significativas, 0111, que é o complemento de 1000. Isso quer dizer que, em decimal, o código seria $128 + 7 = 135$. Conhecendo-se, então, a regra de formação dos caracteres, é possível fazer-se um PLOT/UNPLOT numa posição de PRINT (definida por 32 colunas e 22 linhas), lendo-se o caracter que já está naquela posição do PRINT e alterando-o de acordo com o quadrante do PLOT/UNPLOT em questão, a fim de se acrescentar/eliminar o quadrante na figura já existente.

Passemos, então, a um exemplo concreto. Suponhamos que se queira fazer um PLOT nas coordenadas X e Y que, analisadas, mostram tratar-se de se fazer um PRINT na linha L e coluna C, sendo o quadrante o de número 3. Inicialmente, devemos ler o que já existe na posição PRINT (para tal, faz-se um PEEK no endereço da memória referente àquela posição). Suponhamos, ainda, que o PEEK mostra a presença de um caracter de código 5 (isto é, $1 + 4$, o que quer dizer que temos em preto as posições 0 e 2, ou seja, a metade esquerda da posição PRINT). Ora, fazer um PLOT no quadrante 3 significa colocar um "1" no "bit" de número 3 (isto é, no quarto, da direita para a esquerda). Dessa forma, o código de novo caracter seria 00001101, mas, como temos um "1" na posição número 3, devemos complementar os quatro "bits" menos significativos e colocar um "1" no "bit" de número 7, o que leva o código anterior para $10000010 = 128 + 2 = 130$.

Na Fig. 10.3 temos, na forma de diagrama de blocos, a descrição do programa. Inicialmente, salvam-se os registradores e se determina o quadrante do PLOT/UNPLOT simplesmente verificando o "bit" "0" dos registradores B e C, que guardam as coordenadas X e Y, respectivamente. Aqui foi utilizado um pequeno artifício, que consiste em se carregar inicialmente o acumulador com 87H e, dependendo se o "bit" "0" de B e C é "0" ou "1" (há quatro possibilidades), somam-se valores ao conteúdo de A de tal forma que o valor final do mesmo será o segundo "byte" da instrução que coloca ou retira um "bit" "1" (portanto SET ou RESET) no conteúdo do que já está naquela posição do PRINT. Depois, calcula-se o endereço da posição PRINT referente ao PLOT em questão. Isso é feito através do seguinte procedimento:

$$\text{Coluna} = \text{INT } X/2$$

$$\text{Linha} = 22 - \text{INT } Y/2$$

A função $\text{INT } (X \text{ ou } Y)/2$ é feita pela instrução SRL. Conhecida a posição do PRINT, lê-se seu conteúdo. Se for maior que 7, quer dizer que se trata de uma posição que tem o quadrante 3 preto, o que requer uma correção, que é feita simplesmente achando-se o complemento de todo o "byte" (instrução CPLA, no endereço 16581). Agora, faz-se a alteração do código do PRINT colocando-se um 1, ou

colocando-se um 0, na posição referente ao quadrante do PLOT/UNPLOT — o que é feito mediante um outro truque: as instruções de SET um “bit” num registrador têm o valor do segundo “byte” de código igual ao valor da correspondente instrução RESET, somado 64 (isto é, 40 em hexadecimal). Assim, o código de RES 3,A é CB9F, e o código de SET 3,A é CBDF. Ora, como já foi dito, para se fazer um PLOT tem-se que colocar 40H no acumulador. Ao se chamar a sub-rotina, esse valor é justamente somado ao segundo “byte” da instrução, na posição de memória 16582. É bom o leitor verificar cuidadosamente o que acabamos de expor, uma vez que se trata de alterar o código de uma instrução em função dos dados do problema, isto é, o valor do segundo “byte” da instrução em 16582 depende da posição do quadrante em que será feito o PLOT/UNPLOT e, em seguida, é alterado se se tratar de um PLOT. É justamente esse truque que permite que o programa seja tão curto e, portanto, tão rápido.

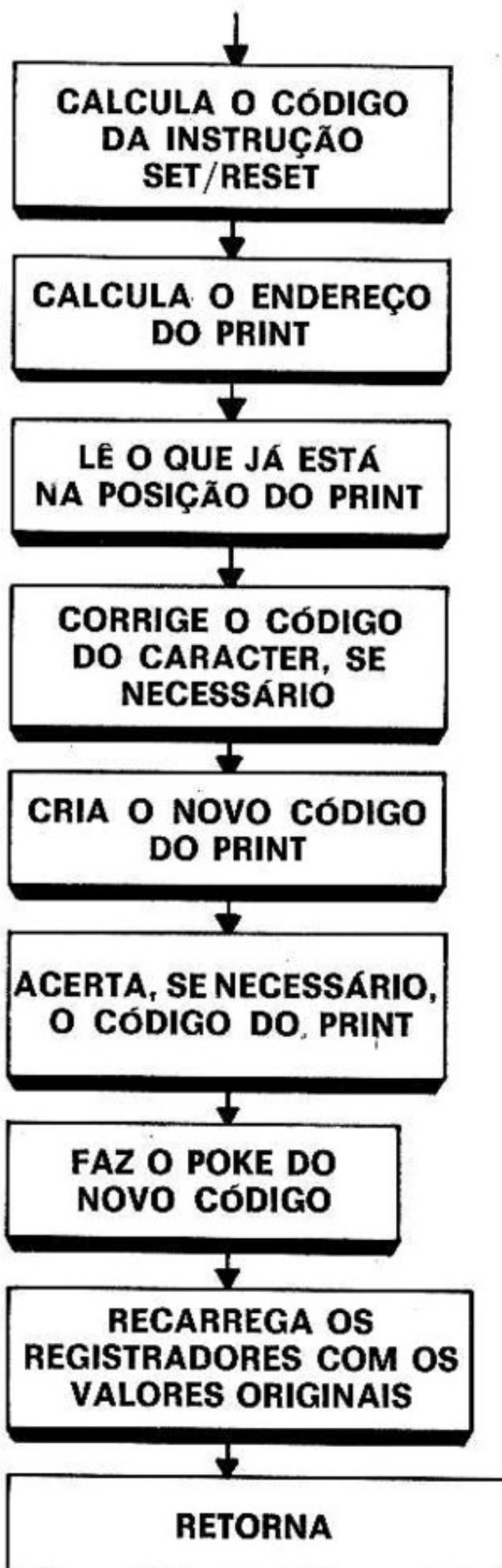
A seguir transcrevemos o programa, indicando a posição de memória, o mnemônico da instrução, o código hexadecimal e algumas explicações. O leitor não terá dificuldades em acompanhar o programa, já que os pontos mais difíceis já foram abordados.

16527	PUSH BC	C5	} Salva os registradores.
16528	PUSH HL	E5	
16529	PUSH DE	D5	
16530	PUSH AF	F5	
16531	LD A, 135	3E87	} Determina o quadrante do PLOT dentro do PRINT.
16533	BIT 0, B	C640	
16535	JR NZ	2002	
16537	ADD A, 16	C610	} Calcula o código da instrução SET/RESET.
16539	BIT 0, C	CB41	
16541	JR Z	2602	
16543	ADD A, 6	C608	} Calcula o endereço do PRINT.
16545	LD D, A	57	
16546	POP AF	F1	
16547	PUSH AF	F5	
16548	ADD A, D	82	
16549	LD (16583), A	32C740	
16552	SRL B	CB36	
16554	SRL C	CB39	
16556	LD HL, (16396)	2A0C40	
16559	INC HL	23	
16560	LD D, 0	1600	
16562	LD E, C	59	
16563	ADD HL, DE	19	
16564	LD A, 21	3E15	
16566	SUB A, B	90	
16567	LD B, A	47	
16568	JR Z	2606	
16570	LD DE, 33	11210E	
16573	ADD HL, DE	19	

16576	LD A, (HL)	7E	} Lê o que já está na posição de PRINT.
16577	CP A, 6	FE08	
16579	JR C	3801	} Corrige o código do caracter, se necessário
16581	CP L, A	2F	
16582	SET/RES X, A	C600	} Cria o novo código do PRINT.
16584	BIT 3, A	C65F	
16586	JR Z	2605	} Acerta, se necessário, o código do PRINT.
16588	CP L, A	2F	
16589	AND A, 130	E66F	
16591	JR	1602	
16593	AND A, 15	E60F	
16595	LD (HL), A	77	} Faz o POKE do novo código.
16596	POP AF	F1	} Recarrega os registradores com os valores originais.
16597	POP DE	D1	
16598	POP HL	E1	
16599	POP BC	C1	
16600	RET	C9	} Retorna.

Fig. 10.3





CALEIDOSCÓPIO

11

Este programa gera na tela figuras de um bonito efeito visual, que imitam razoavelmente um caleidoscópio. O programa consiste de uma pequena parte escrita em BASIC e de quatro sub-rotinas escritas em linguagem de máquina.

Em linhas gerais, é a seguinte a estrutura do programa: é reservado na memória um espaço de 176 "bytes", isto é, da posição 16640 até a posição 16815, onde são colocados, de forma aleatória, os valores 0 ou 128, que correspondem aos caracteres espaço e seu inverso. O preenchimento dessa região da memória é feito em BASIC. Em seguida, e a partir desses 176 "bytes", é criada a figura na tela com dupla simetria, ou seja, com relação a um eixo horizontal passando no meio da tela e com relação a um eixo vertical passando pelo meio da tela. A tela tem 704 elementos arrumados em 32 colunas e 22 linhas. Dessa forma, os 176 "bytes" anteriormente descritos vão representar os elementos do primeiro quadrante da figura a ser gerada, como mostra a Fig. 11.1.

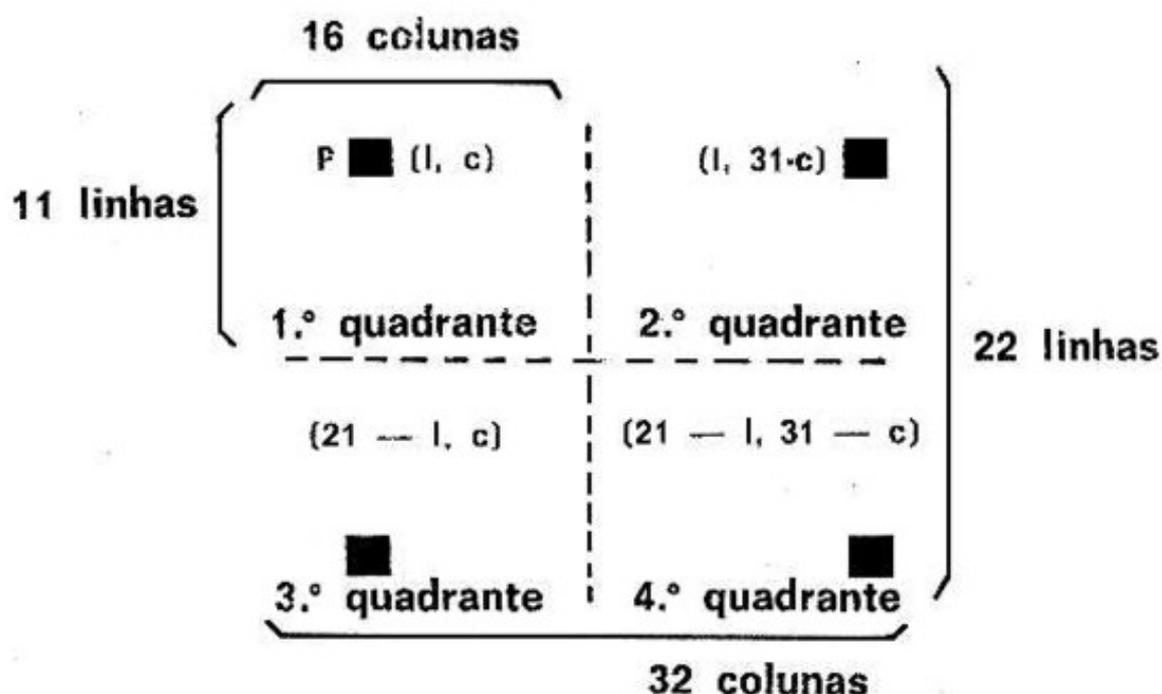


Fig. 11.1

Assim, o elemento genérico P, do primeiro quadrante, cujas coordenadas são (l, c), onde "l" corresponde à linha e "c" à coluna, é gerado inicialmente na área da memória de 176 "bytes" de onde é lido e colocado na região da RAM correspondente à tela, ocupando a posição (l, c). O elemento simétrico a P com relação ao eixo vertical é obtido alterando-se a coordenada c para $31 - c$. De outro lado, o elemento simétrico a P com relação ao eixo horizontal é obtido mudando-se a linha de l para $21 - l$. Finalmente, o elemento do quarto quadrante tem coordenada $(21 - l, 31 - c)$ e pode ser obtido a partir de qualquer um dos elementos já gerados no 2.º ou 3.º quadrantes. Dessa forma, o programa lê um por um os elementos da memória (de 176 "bytes"), faz corresponder ao elemento um par de coordenadas (l, c) e gera na tela os quatro elementos simétricos como mostra a Fig. 11.1.

Ao final dessa sub-rotina, o programa volta ao BASIC, aguardando a entrada de uma "STRING". Assim, se pressionarmos "NEW LINE", o programa prossegue e entra em outra sub-rotina, que apaga a tela. O leitor poderá estar pensando porque uma sub-rotina para apagar a tela e não, simplesmente, um comando CLS. Acontece que, se o computador tiver menos do que 3,5K de memória, o comando CLS irá apagar a região da RAM correspondente à tela e deixar somente o código 76H. Por isso usamos uma sub-rotina que deixa o espaço na RAM correspondente à tela com o valor 0 (ou seja, espaço) em todas as 704 posições.

Uma vez apagada a tela, o programa volta para outra sub-rotina, cuja função é mexer nos 176 elementos da memória e, em seguida, voltar novamente para a sub-rotina que gera a figura na tela.

Como tudo isso se processa muito rapidamente, tem-se a impressão de que os elementos, sempre respeitando as duas simetrias, estão se mexendo como em um caleidoscópio. Se, em vez de pressionar "NEW LINE", como foi anteriormente assumido, o usuário pressionar N (de novo), o programa irá para a última sub-rotina, que apaga a região de 176 "bytes" e, a partir da, tudo recomeça.

Na Fig. 11.2 temos, em blocos, a descrição do programa.

Antes de mostrarmos o programa, devemos comentar três pontos ainda. Primeiramente, para aqueles que têm uma RAM com menos que 3,5K, é necessário abrir-se espaço na RAM para a região da tela. Isso pode ser feito de diversas maneiras, algumas já vistas no livro, mas, nesse caso, usamos uma diferente, que consiste em "enganar" o computador alterando a RAMTOP (linha 10 do programa), de tal forma que quando o programa interpretador for verificar o tamanho da memória ele vai pensar que existe mais que 3,5K e reservará automaticamente espaço para a tela. O segundo comentário é quanto ao que chamamos de "mexer" com os 176 elementos da memória. Na verdade, o que fazemos é deslocá-los para cima em uma casa, sendo que o mais alto, isto é, na posição 16815, é colocado na primeira posição, ou seja, 16640. Finalmente, a região da memória de 176 "bytes" não foi colocada por acaso a partir do endereço 16640, que é em hexadecimal 4100, mas, sim, porque temos que associar a cada "byte" dessa região uma posição

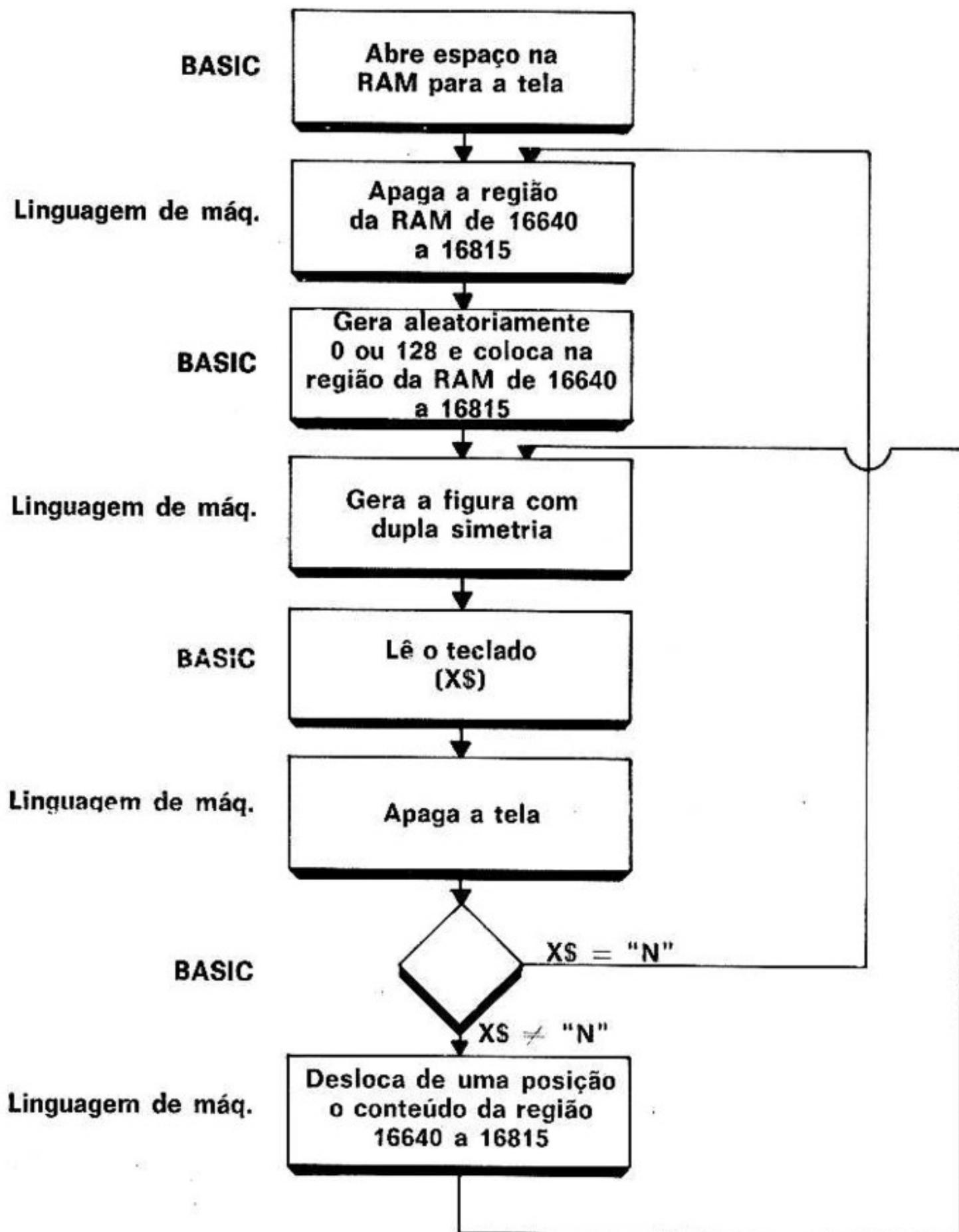


Fig. 11.2

na tela no primeiro quadrante. Ora, como temos dezesseis colunas num quadrante, então o "byte" menos significativo do endereço pode indicar a linha e coluna automaticamente.

Vejamos como.

O "byte" mais significativo será sempre 41. O menos significativo começa em 00 e termina em AF, isto é, podemos associar o "nibble" (meio "byte") menos significativo à coluna e o mais significativo à linha, que variam, respectivamente, de 0 a 15 e de 0 a 10, isto é, de 0 a F e de 0 a A. Assim, a posição de memória 413C corresponde a posição na tela: coluna 12 (vem do C)

linha 3 (vem do 3)

O programa em BASIC:

```

10 POKE 16389,85
20 RAND USR 16604
30 FOR J=16640 TO 16815
40 IF RND>.8 THEN POKE J,128
50 NEXT J
60 RAND USR 16514
70 INPUT X$
80 RAND USR 16816
90 IF X$="N" THEN GOTO 20
100 RAND USR 16616*
110 GOTO 60
8990 STOP
9000 SAVE "CALEIDOSCOPIO"
9010 RUN

```

As sub-rotinas em linguagem de máquina terão os seguintes endereços:

16514 a 16603: gera a figura com dupla simetria

16604 a 16615: zera os 176 "bytes"

16616 a 16633: desloca os 176 "bytes" uma casa para cima

16634 a 16639: não ocupado

16640 a 16815: região dos 176 "bytes"

16816 a 16834: limpa a tela

O programa todo ocupa 321 "bytes" em linguagem de máquina.

```

7 16514 LD HL,16640      210041
      INICIO LD A,(HL)   7E
      AND A             A7
      JR Z (A)          264C

```

Lê a região de 176 "bytes"

Salta se o "byte" for 0

PUSH HL	E5
LD A,15	3E0F
AND A,L	A5
LD E,A	5F
LD B,4	0604
ⓑ SRL L	CB3D
DJNZ ⓑ	10FC
LD D,L	55
LD HL,(15395)	2A0C40
INC HL	23
LD B,0	0600
LD C,E	4B
ADD HL,BC	09
LD A,D	7A
PUSH DE	D5
LD DE,33	112100
AND A	A7
JR Z Ⓒ	2604
LD B,A	47
ⓓ ADD HL,DE	19
DJNZ ⓓ	10FD
Ⓒ POP DE	D1
LD A,128	3E60
LD (HL),A	77
SLA E	CB23
LD A,31	3E1F
SUB A,E	93
LD (16417),A	322140
LD B,0	0600
LD C,A	4F
ADD HL,BC	09
LD A,128	3E60

Faz: (E) = coluna
(D) = Linha

Faz (HL) = (HL) + (E)

(HL) = (HL) + (E) + 33 × (D)

Coloca o caracter no
1.º quadrante

Faz (E) = 2(E)

Faz (A) = 31 - 2(E)
e guarda em 16417

Coloca caracter no
2.º quadrante



	LD (HL), A	77	}	Coloca caracter no 4.º quadrante
	LD A, 21	3E15		
	SLA D	CB22		
	SUB A, D	92		
	LD B, A	47		
	LD DE, 33	112100		
	Ⓔ ADD HL, DE	19		
	DJNZ Ⓔ	10FD		
	LD A, 128	3E80		
	LD (HL), A	77		
	LD A, (16417)	3A2140	}	Coloca caracter no 3.º quadrante
	LD C, A	4F		
	SBC HL, BC	ED42		
	LD A, 128	3E80		
	LD (HL), A	77		
	POP HL	E1		
	Ⓐ INC HL	23		
	LD A, L	7D		
	CP A, 176	FE80		
	JR NZ INICIO	20AA		
	RET	C9	}	Zera os 176 "bytes"
* 16604	LD A, 0	3E00		
	LD B, 176	0680		
	LD HL, 16640	210041		
	Ⓕ LD (HL), A	77		
	INC HL	23		
	DJNZ Ⓕ	10FC		
	RET	C9		
✕ 16616	LD A, (16815)	3AAF41		
	LD BC, 176	01AF00		
	LD HL, 16814	21AE41		
	LD DE, 16815	11AF41		
	LDDR	EDB8		
	LD (16640), A	320041		
	RET	C9		

	RET	C9
*16816	LD HL, (16396)	2A0C40
	INC HL	23
	LD C, 32	0E16
	LD A, 0	3E00
	(H) LD B, 32	0620
	(G) LD (HL), A	77
	INC HL	23
	DJNZ (G)	10FC
	INC HL	23
	DEC C	0D
	JRNZ (H)	20F6
16834	RET	C9

Apaga a tela

UM JOGO AUTO PROGRAMÁVEL 12

O programa que veremos agora é baseado num brinquedo chamado "GABRIELA", produzido pela FUNBEC — Fundação Brasileira para o Desenvolvimento do Ensino e da Ciência, e criado por Isaac Epstein.

Trata-se de um jogo para duas pessoas ou, nesta versão, para um jogador e o computador, e consiste de um tabuleiro com 21 quadrados, numerados de 1 a 21, onde cada jogador, partindo da posição 1 e jogando alternadamente com o adversário, avança de uma a três casas à frente da posição ocupada pelo adversário. Ganha aquele que chegar primeiro ao quadrado 21. Trata-se de um jogo simples, que poderia, inclusive, ter sido implementado inteiramente em BASIC.

O que caracteriza e justifica a apresentação deste jogo é que o programa é feito de tal forma que o computador vai "aprendendo" a jogar à medida que joga e perde. Em outras palavras, o programa faz com que o computador jogue de acordo com as regras e faça movimentos aleatoriamente. À medida que vamos jogando com o computador, ele registra numa região da memória as jogadas catastróficas, ou seja, aquelas que conduziram a máquina à derrota. De outro lado, as jogadas do computador que o conduzirem à vitória são sabiamente desprezadas, uma vez que ele não conhece a qualidade de seu adversário e, muito menos, as suas legítimas intenções.

E, dessa forma, por processo de erro e acerto, a máquina vai jogando aleatoriamente sem incorrer em erros já cometidos. Se continuarmos jogando contra o computador, chegará o momento em que ele vencerá sempre.

Aconselhamos ao leitor interessado em se aprofundar no assunto a consultar o jogo "GABRIELA", onde é apresentado um outro jogo, no mesmo estilo, chamado "mini-damas".

O programa para o jogo "21" foi escrito em BASIC para a parte de entrada de dados (movimentos do jogador) e impressão na tela, enquanto que a parte lógica do jogo foi feita em linguagem de máquina, por ser mais fácil e rápida na execução.

A mencionada memória onde o computador registra suas más jogadas ocupa a posição que vai de 16514 a 16533, isto é, vinte "bytes". Cada um desses "bytes" guarda os possíveis movimentos que o computador pode fazer se o jogador colocar sua peça naquela posição. No início do jogo, tanto o jogador quanto o computador

ocupam a casa número 1. O jogador faz o primeiro movimento indo para a casa 2, 3 ou 4. Se ele for, por exemplo, para a casa 3, o computador pode fazer seu lance indo para a casa 4, 5 ou 6. Em princípio, a máquina escolhe aleatoriamente uma dessas três possibilidades (para tal, ela lê o "byte" menos significativo da variável FRAMES); em seguida, ele lê o conteúdo do "byte" 16516 (o "byte" 16514 corresponde à casa 1, o "byte" 16515 corresponde à casa 2, e assim por diante, até o "byte" 16533, que corresponde à casa 20) e verifica se a jogada que escolheu está condenada ou não. Se não estiver, será esse o lance do computador; se estiver condenada, o computador volta a escolher outra jogada, até que ache uma possível.

As informações armazenadas no "byte" em questão indicam se o computador pode, a partir daquela casa (que é a posição ocupada pelo jogador) avançar uma, duas ou três posições. Para tal, adotou-se a seguinte convenção:

- um 1 no "bit" de ordem zero indica que avançar uma casa é possível;
- um 1 no "bit" de ordem um indica que o computador pode avançar duas casas;
- um 1 no "bit" de ordem dois indica que o computador pode avançar três casas.

Assim, se o conteúdo de um desses "bytes" for 00000101 quer dizer que, estando o adversário na casa correspondente àquele "byte", o computador pode avançar uma ou três casas e a escolha entre essas duas opções será feita aleatoriamente.

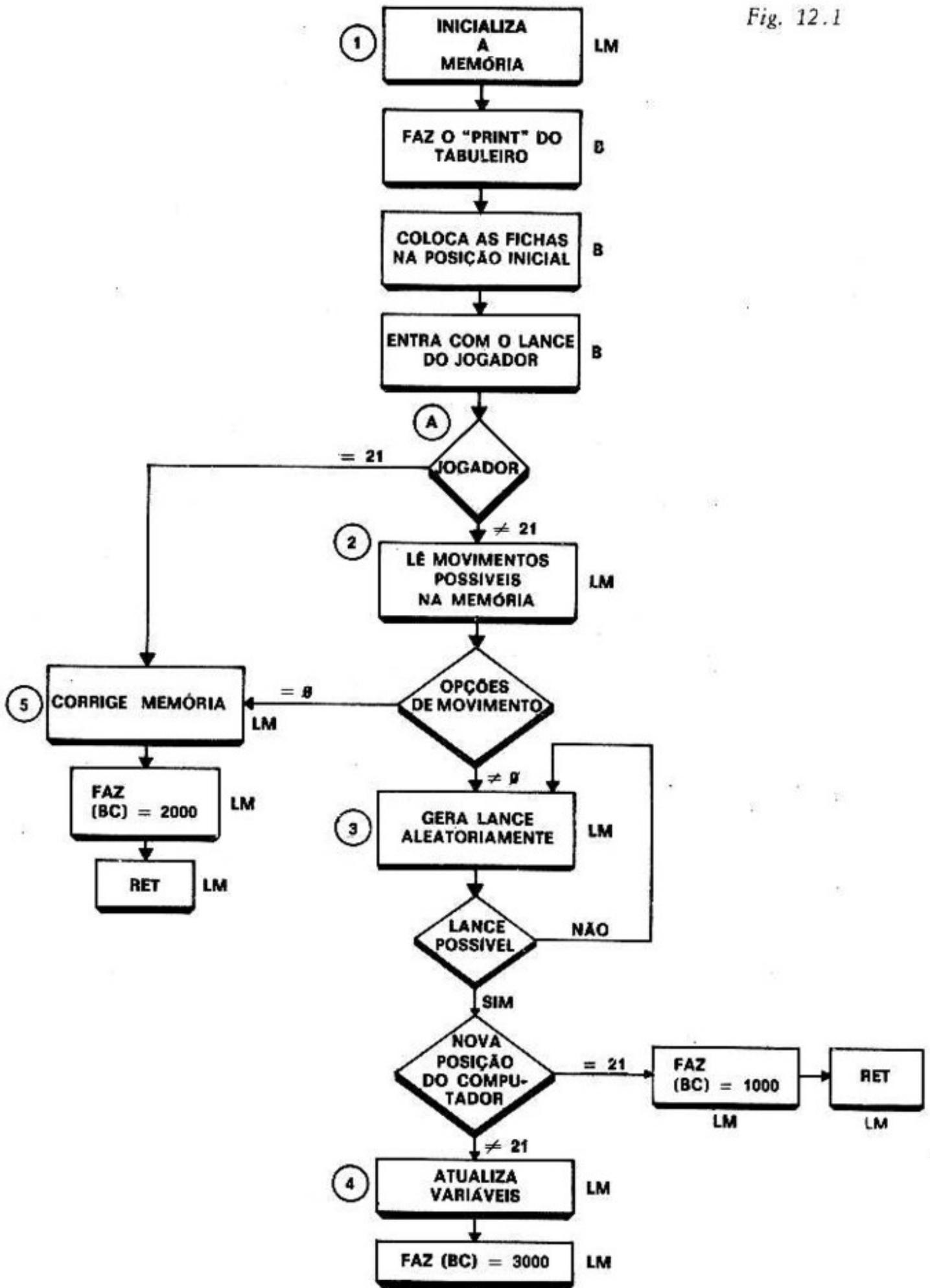
As variáveis utilizadas no programa são:

POSIÇÃO	DESCRIÇÃO	CONTEÚDO
16507	Última posição ocupada pelo computador	130 a 150 (decimal)*
16508	Última jogada do computador.	00000001 ou 00000010 ou 00000100
16444	Última posição do jogador.	1 a 20 (decimal)
16445	Posição do último movimento do computador.	130 a 150 (decimal)

* Note-se que somente o "byte" menos significativo do endereço é que é armazenado, uma vez que o "byte" mais significativo é sempre 64.

Na Fig. 12.1 temos o diagrama de blocos do programa. O bloco 1 carrega as vinte posições da memória que armazenam as jogadas possíveis. Da posição 1

Fig. 12.1



até a posição 18 é colocado o valor 7, ou seja, 00000111, pois inicialmente temos para todas essas posições as três opções possíveis, que são: anda uma, duas ou três casas. Na posição 19 é colocado o valor 3 e, na posição 20, o valor 1.

O ponto A representa o início do programa propriamente dito e é para onde retorna o "loop" sempre que for a vez do jogador jogar.

O bloco 2 lê o conteúdo da memória correspondente à posição onde ficou o jogador após o seu lance. Se o conteúdo dessa posição de memória for zero, quer dizer que não existe lance possível a ser feito ou, em outras palavras, o computador já perdeu. Se o conteúdo dessa posição for diferente de zero, então o computador escolhe aleatoriamente — bloco 3 — uma das posições possíveis.

Em seguida, o computador verifica se com a sua última jogada ele ganhou ou não o jogo. Se ganhou, carrega-se o par de registradores BC com 1000 e volta-se para o BASIC. O leitor vai ver no programa em BASIC que a chamada do programa em linguagem de máquina (a partir do bloco 2) é feita com a instrução GOTO USR X. Assim, dependendo da jogada, volta-se para a linha do programa 1000, 2000 ou 3000.

O bloco 4 atualiza as variáveis do jogo em função da última jogada do computador, carrega BC com 3000 e retorna para o BASIC.

No bloco 5 é feita a correção na memória, isto é, a opção de jogada utilizada pela máquina ao fazer seu último movimento é eliminada. Vejamos um exemplo, como indicado em seguida:

jogador	16	21
computador	19	

Nessa seqüência de movimentos, o computador optou por andar três casas (da casa 16 à casa 19) em seu último movimento, o que propiciou ao jogador ganhar a partida. Num caso como esse, o programa referente ao bloco 5 eliminaria da posição de memória referente à posição 16 o lance de três casas.

Após a correção da memória, o par de registradores BC é carregado em 2000 e é feito o retorno ao BASIC.

Ao lado de cada bloco está indicado se o mesmo foi escrito em BASIC (B) ou em linguagem de máquina (LM).

O programa em BASIC é o seguinte:

```
10 LET Z$=""
20 RAND USR 16534
30 CLS
40 FAST
50 PRINT TAB 10;"JOGO DO 21"
60 FOR J=1 TO 21
70 PRINT AT J,1;J
80 NEXT J
90 PRINT AT 1,5;"0";TAB 8;"*"
100 PRINT AT 7,20;"0 = JOGADOR"
110 PRINT AT 10,20;"*=COMPUTADOR"
R"
```

```

120 SLOW
130 LET A=1
140 LET B=1
150 PRINT AT 17,20;"JOGUE"
160 INPUT X
170 IF X>21 OR X-B>3 OR X<=B TH
EN GOTO 160
180 PRINT AT A,5;" "
190 PRINT AT X,5;"0"
200 PRINT AT 17,20;Z$
210 POKE 16444,X
220 LET A=X
230 IF X=21 THEN GOTO USR 16619
240 GOTO USR 16553
1000 PRINT AT 17,20;"GANHEI";TAB
20;"OUTRA (S/N)"
1010 INPUT X$
1020 IF X$="N" THEN STOP
1030 GOTO 30
2000 PRINT AT 17,20;"PERDI";TAB
20;"OUTRA (S/N)"
2010 GOTO 1010
3000 PRINT AT B,8;"#"
3010 LET B=PEEK 16507-129
3020 PRINT AT B,8;"#"
3030 GOTO 150
9900 SAVE "21"

```

E, em linguagem de máquina, temos o seguinte programa:

16514 a 16533 20 posições de memória

16534	LD HL,16514	218240
	LD A,7	3E07
	LD B,18	0612
Ⓐ	LD (HL),A	77
	INC HL	23
	DJNZ Ⓐ	10FC
	LD A,3	3E03
	LD (HL),A	77
	INC HL	23
	LD A,1	3E01
	LD (HL),A	77
	RET	C9

Inicializa a memória

16553	LD A, (16444)	3A3C40	} Lê posição do jogador
	LD HL, 16513	218140	
	ADD A, L	85	} Lê movimentos possíveis na memória
	LD L, A	6F	
	LD A, (HL)	7E	
	AND A	A7	
	JP Z (B)	CAEB40	
	LD D, A	57	
(C)	LD A, (FRAMES)	3A3440	} Gera lance aleatório e verifica se é uma opção possível
	AND A, 3	E603	
	JR Z (C)	28F9	
	LD B, A	47	
	LD A, 128	3E80	
(D)	RLC A	CB07	
	DJNZ (D)	10FC	
	LD E, A	5F	
	LD A, D	7A	
	AND E	A3	
	JR Z (C)	28ED	
	LD A, L	7D	
	LD (16445), A	323D40	
	LD A, E	7B	
	LD C, E	4B	
	CP A, 4	FE04	} Calcula nova posição do computador
	JR NZ (E)	2002	
	LD C, 3	0E03	
(E)	LD A, L	7D	
	ADD A, C	81	
	CP A, 150	FE96	

	JR NZ (F)	2004	
	LD BC, 1000	01E803	
	RET	C9	
16508 (F)	LD (16507), A	327B40	} Atualiza variáveis
	LD A, E	7B	
	LD (16508), A	327C40	
	LD BC, 3000	01B80B	
	RET	C9	
16519 (40EB) (B)	LD A, (16508)	3A7C40	} Corrige memória
	CP L	2F	
	LD B, A	47	
	LD A, (16445)	3A3D40	
	LD H, 64	2640	
	LD L, A	6F	
	LD A, (HL)	7E	
	AND B	A0	
	LD (HL), A	77	
	LD BC, 2000	01D007	
	RET	C9	

Esse programa em linguagem de máquina ocupa 123 "bytes".

BIBLIOGRAFIA

Como foi dito inicialmente, este livro foi escrito visando preencher uma lacuna, existente segundo o autor, entre o material literário sobre linguagem de máquina em si e aplicações mais sofisticadas da mesma nos microcomputadores compatíveis com o ZX-81.

Pretende, assim, o autor dar alguma contribuição ao material bibliográfico existente e convida os leitores a que tomem igualmente conhecimento destes livros que certamente ajudarão a dar uma visão mais ampla e mais perfeita do assunto.

LORD, Mike. THE EXPLORERS GUIDE TO THE ZX81. Essex/UK, Timedata Ltd, 1982.

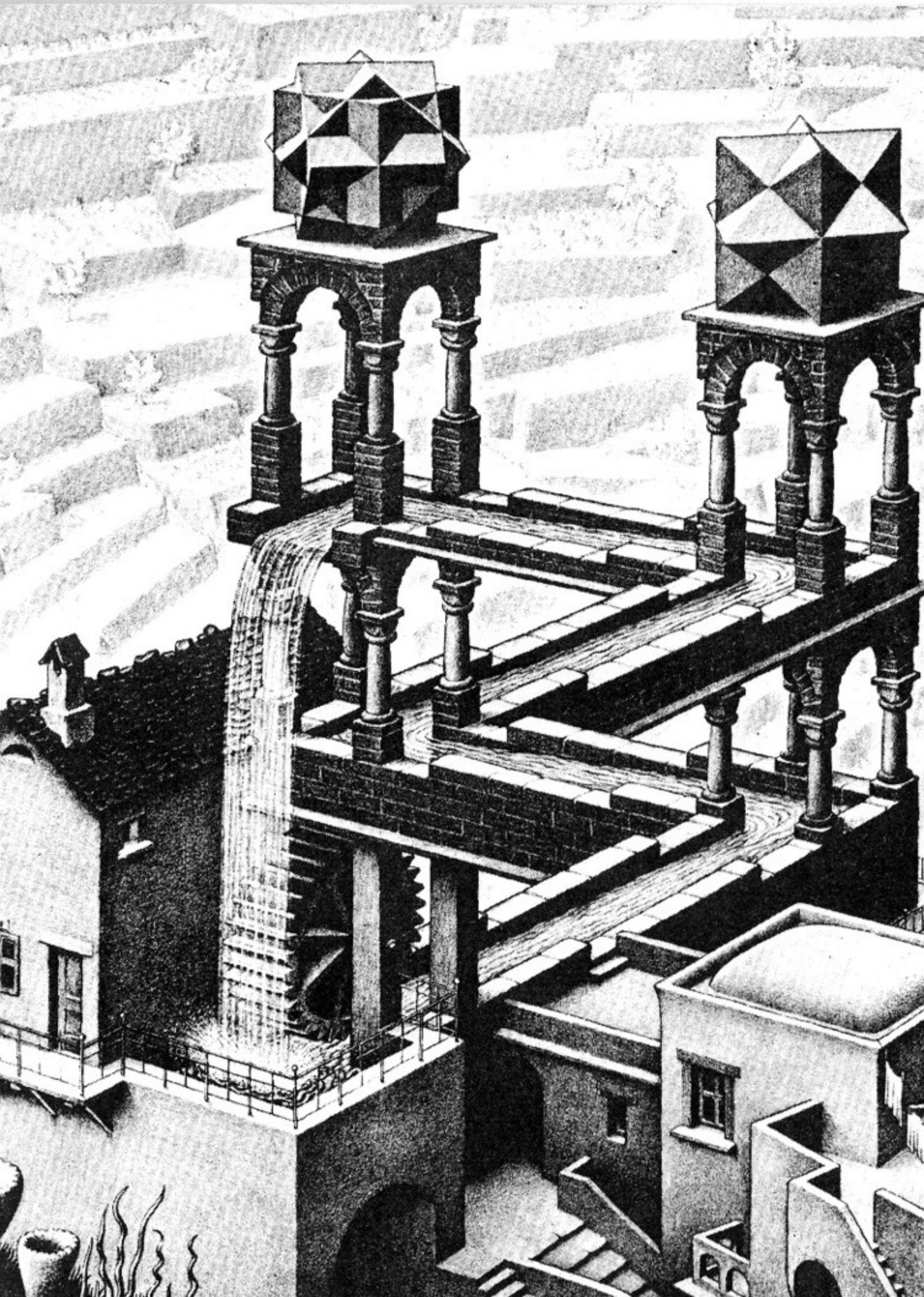
LOGAN, Ian (Dr.). UNDERSTANDING YOUR ZX-81 ROM. Australia, Melbourne House Ltd., 1981.

BAKER, Tony. MASTERING MACHINE CODE ON YOUR ZX-81. 2nd. edition, London, Interface, 1981.

LOGAN, Ian (Dr.). SINCLAIR ZX-81 ROM DISSASSEMBLY. Australia, Melbourne House Ltda., 1981.

ROSSINI, Flavio. LINGUAGEM DE MÁQUINA PARA O TK. São Paulo, Editora Moderna Ltda., 1983.

PROGRAMMING Z80 CPU. 2nd. edition, Zilog Inc. SGS-ATES COMPONENTI ELECTRONIC SpA.



APÊNDICE A

CRIANDO

UMA LINHA REM

Muitos programas em linguagem de máquina exigem uma linha REM muito longa para introduzir os códigos do programa, o que torna muitas vezes o trabalho cansativo.

O programa a seguir cria uma linha REM com o número de zeros necessários para se introduzir o código de máquina.

Utilize o programa mestre apresentado no capítulo 2 para introduzir os códigos hexadecimais do programa apresentado a seguir. Ajuste a linha 1 para 96 zeros.

16514		00
16515		00
16516	CALL FAST	CD230F
	LD BC,6	010500
	LD HL,(16514)	2A3240
	PUSH HL	E5
	ADD HL,BC	09
	LD B,H	44
	LD C,L	4D
	LD HL,(16425)	2A2940
	ADD HL,BC	09
	LD (16425),HL	222940
	LD HL,16396	210040 *
	LD A,9	3E09
	(B) LD E,(HL)	5E
	INC HL	23
	LD D,(HL)	56
	PUSH DE	D5
	EX DE,HL	EB
	ADD HL,BC	09
	EX DE,HL	EB
	LD (HL),D	72
	DEC HL	2B
	LD (HL),E	73
	INC HL	23
	INC HL	23
	DEC A	3D
	JR Z	2803
	POP DE	D1
	JR	18EE
	POP HL	E1
	PUSH HL	E5
	LD BC,16611	01E240
	AND A	A7
	SBC HL,BC	ED42

	LD B,H	44
	LD C,L	40
	POP HL	E1
	LDDR	EDB8
	LD HL,16512	21E340
16575	LD (HL),00	3600
	INC HL	23
16578	LD (HL),00	3600
	INC HL	23
	POP BC	C1
	INC BC	03
	INC BC	03
	LD (HL),C	71
	INC HL	23
	LD (HL),B	70
	INC HL	23
	LD (HL),234	36EA
	DEC BC	0B
	DEC BC	0B
	INC HL	23
	LD DE,1	110100
	EX DE,HL	EB
	ADD HL,DE	19
	EX DE,HL	E6
	LD (HL),28	361C
	LDIR	EDB0
	LD (HL),117	3675
	INC (HL)	34
	CALL SLOW	CD2B0F
	RET	C9

Para rodar o programa digite as linhas em BASIC a seguir.

```

7990 STOP
8000 PRINT "N° LINHA=";
8010 INPUT X
8020 PRINT X
8030 POKE 16576,INT (X/256)
8040 POKE 16579,X-256*INT (X/256)
)
8050 PRINT "N° ZEROS=";
8060 INPUT N
8070 PRINT N
8080 POKE 16514,N-256*INT (N/256)
)
8090 POKE 16515,INT (N/256)
8100 RAND USR 16516
8110 LIST X

```

É aconselhável que este programa mestre, com o criador de linha REM, seja gravado numa fita cassete, para que ele possa ser usado todas as vezes que você for digitar um programa em linguagem de máquina.

Rode o programa digitando RUN 8000. O primeiro valor a ser digitado é o número que deve ter a linha REM a ser criada. Este número deverá ser maior do que 1 e menor do que 7990. O segundo valor a ser digitado corresponde ao número de zeros que a nova linha REM deve ter.

Retire a linha 1 REM e sua nova linha REM estará no início da região reservada ao BASIC, ou seja, o primeiro zero se encontra no endereço 16514.

Se você achar melhor modificar o número da nova linha REM digite:

POKE 16510, número da linha

Agora para você introduzir os códigos do seu programa em linguagem de máquina digite:

RUN 9000.

Caso você deseje fazer algumas modificações no programa criador de linha REM, veja os endereços com os pontos fundamentais do programa:

POKE 16510, número da linha

Agora para você introduzir os códigos do seu programa em linguagem de máquina digite:

RUN 9000

Caso você deseje fazer algumas modificações no programa criador de linha REM, veja os endereços com os pontos fundamentais do programa:

16514 — byte menos significativo do número de zeros da linha

16576 — byte mais significativo do número da linha a ser criada

16579 — byte menos significativo do número da linha a ser criada

16600 — código do caractere na nova linha REM

16515 — byte mais significativo do número de zeros da linha

NOTA: no Ringo o código do REM é 249.

APÊNDICE B

TABELA DE CARACTERES

0	00		24	18	✓	48	30	K
1	01	■	25	19	:	49	31	L
2	02	■	26	1A	,	50	32	M
3	03	■	27	1B	.	51	33	N
4	04	■	28	1C	0	52	34	O
5	05	■	29	1D	1	53	35	P
6	06	■	30	1E	2	54	36	Q
7	07	■	31	1F	3	55	37	R
8	08	■	32	20	4	56	38	S
9	09	■	33	21	5	57	39	T
10	0A	■	34	22	6	58	3A	U
11	0B	■	35	23	7	59	3B	V
12	0C	■	36	24	8	60	3C	W
13	0D	■	37	25	9	61	3D	X
14	0E	■	38	26	A	62	3E	Y
15	0F	■	39	27	B	63	3F	Z
16	10	(40	28	C	64	40	RND
17	11)	41	29	D	65	41	INKEY\$
18	12	>	42	2A	E	66	42	PI
19	13	<	43	2B	F	67	43	?
20	14	=	44	2C	G	68	44	?
21	15	+	45	2D	H	69	45	?
22	16	-	46	2E	I	70	46	?
23	17	*	47	2F	J	71	47	?

72	4B	?	96	60	?	120	78	?
73	49	?	97	61	?	121	79	?
74	4A	?	98	62	?	122	7A	?
75	4B	?	99	63	?	123	7B	?
76	4C	?	100	64	?	124	7C	?
77	4D	?	101	65	?	125	7D	?
78	4E	?	102	66	?	126	7E	?
79	4F	?	103	67	?	127	7F	?
80	50	?	104	68	?	128	80	■
81	51	?	105	69	?	129	81	▣
82	52	?	106	6A	?	130	82	▤
83	53	?	107	6B	?	131	83	▥
84	54	?	108	6C	?	132	84	▦
85	55	?	109	6D	?	133	85	▧
86	56	?	110	6E	?	134	86	▨
87	57	?	111	6F	?	135	87	▩
88	58	?	112	70	?	136	88	▪
89	59	?	113	71	?	137	89	▫
90	5A	?	114	72	?	138	8A	▬
91	5B	?	115	73	?	139	8B	▭
92	5C	?	116	74	?	140	8C	▮
93	5D	?	117	75	?	141	8D	▯
94	5E	?	118	76	?	142	8E	▰
95	5F	?	119	77	?	143	8F	▱

144	90	I	158	A8	C	192	C0	"
145	91	J	159	A9	D	193	C1	AT
146	92	K	170	AA	E	194	C2	TAB
147	93	L	171	AB	F	195	C3	?
148	94	M	172	AC	G	196	C4	CODE
149	95	N	173	AD	H	197	C5	VAL
150	96	O	174	AE	I	198	C6	LEN
151	97	P	175	AF	J	199	C7	SIN
152	98	Q	176	B0	K	200	C8	COS
153	99	R	177	B1	L	201	C9	TAN
154	9A	S	178	B2	M	202	CA	ASN
155	9B	T	179	B3	N	203	CB	ACS
156	9C	U	180	B4	O	204	CC	ATN
157	9D	V	181	B5	P	205	CD	LN
158	9E	W	182	B6	Q	206	CE	EXP
159	9F	X	183	B7	R	207	CF	INT
160	A0	Y	184	B8	S	208	D0	SGR
161	A1	Z	185	B9	T	209	D1	SGN
162	A2	[186	BA	U	210	D2	ABS
163	A3]	187	BB	V	211	D3	PEEK
164	A4	^	188	BC	W	212	D4	USR
165	A5	_	189	BD	X	213	D5	STR\$
166	A6	`	190	BE	Y	214	D6	CHR\$
167	A7	{	191	BF	Z	215	D7	NOT

216 D8 **	229 E5 FAST	242 F2 PAUSE
217 D9 OR	230 E6 NEW	243 F3 NEXT
218 DA AND	231 E7 SCROLL	244 F4 POKE
219 DB <=	232 E8 CONT	245 F5 PRINT
220 DC >=	233 E9 DIM	246 F6 PLOT
221 DD <>	234 EA REM	247 F7 RUN
222 DE THEN	235 EB FOR	248 F8 SAVE
223 DF TO	236 EC GOTO	249 F9 RAND
224 E0 STEP	237 ED GOSUB	250 FA IF
225 E1 LPRINT	238 EE INPUT	251 FB CLS
226 E2 LLIST	239 EF LOAD	252 FC UNPLOT
227 E3 STOP	240 F0 LIST	253 FD CLEAR
228 E4 SLOW	241 F1 LET	254 FE RETURN
		255 FF COPY

APÊNDICE C **APÊNDICE D**

VALORES **DO TECLADO** **&** **VARIÁVEIS** **DO SISTEMA**

VALORES DO TECLADO

Como foi mostrado no capítulo 5, quando da explicação da sub-rotina KSCAN, é o seguinte o conteúdo que retorna no par de registradores HL.

TECLA	CONTEÚDO DE HL EM HEXADECIMAL	
	(sem shift)	FCF7 (com shift)
1	FDE7	
2	FBF7	FAF7
3	F7F7	F6F7
4	EFF7	EEF7
5	DFF7	DEF7
6	Dfef	DEEF
7	EFEF	EEEF
8	F7EF	F6EF
9	FBEF	FAEF
Ø	FDEF	FCEF
Q	FDBF	FCBF
W	F8FB	FAFB
E	F7FB	F6FB
R	EFFB	EEFB
T	DFFB	DEFB
Y	DFDF	DEDF
U	EFDF	EEDF
I	F7DF	F6DF
O	F8DF	FADF
P	FDDF	FCDF
A	FDFD	FCFD
S	F8FD	FAFD
D	F7FD	F6FD
F	EFFD	EEFD
G	DFFD	DEFD
H	DFBF	DEBF
J	EFBF	EEBF
K	F7BF	F6BF
L	F8BF	FABF
N/L	FDBF	FCBF
Z	F8FE	FAFE
X	F7FE	F6FE
C	EFFE	EEFE
V	DFFE	DEFE
B	DF7F	DE7F
N	EF7F	EE7F
M	F77F	F67F
•	F87F	FA7F
espaço	FD7F	FC7F

VARIÁVEIS DO SISTEMA

Damos a seguir uma tabela que resume as principais características das variáveis do sistema operacional dos microcomputadores compatíveis com o ZX-81. Para maiores detalhes sobre o significado exato de cada variável recomendamos ao leitor a consulta ao manual do microcomputador.

As variáveis do sistema operacional ocupam as posições de memória RAM 16384 a 16508, inclusive. Essas variáveis são escritas na RAM pelo programa monitor e servem para indicar à máquina aquelas condições operacionais que variam em função do programa.

NOME	ENDEREÇO		N.º DE BYTES	COMENTÁRIOS
	DECIMAL	HEXADECIMAL		
ERR.NR	16384	4000	1	
FLAGS	16385	4001	1	Flags do Basic
ERR.SP	16386	4002	2	Início do Stack
RAMTOP	16388	4004	2	Endereço do primeiro byte do BASIC
MODE	16390	4006	1	Indica tipo de cursor
PPC	16391	4007	2	
VERSN	19393	4009	1	
E.PPC	16394	400C	2	N.º da linha atual
D.FILE	16396	400A	2	Início da área do PRINT
DF.CC	16398	400E	2	Endereço do PRINT
VARs	16400	4010	2	Início da área de variáveis
DEST	16402	4012	2	Endereço da variável a ser atribuída
E.LINE	16404	4014	2	
CH.ADD	16406	4016	2	Endereço do próximo caracter a ser interpretado
X.PTR	16408	4018	2	
PIL FUN	16410	401A	2	Início da pilha para cálculo
PIL MIM	16412	401C	2	Fim da pilha para cálculo

BERG	16414	401E	1	Registrador do calculador
MEM	16415	401F	2	
SPARE 1	16417	4021	1	Byte vago
DF.SZ	16418	4022	1	N.º de linhas na parte inferior da tela
S.TOP	16419	4023	2	
LAST.K	16421	4025	2	Última tecla pressionada
DB.ST	16423	4027	1	
MARGIN	16424	4028	1	
NXT LIN	16425	4029	2	Próxima linha a ser executada
OLD PPC	16427	402B	2	
FLAGX	16429	402D	1	Flags do Basic
STRLEN	16430	402E	2	
T.ADDR	16432	4030	2	
SEED	16434	4232	2	Semente de RND
FRAMES	16436	4234	2	Contador de tempo
COORX	16438	4036	1	Coord X do último PLOT
COORY	16439	4037	1	Coord. Y do último PLOT
PR.CC	16440	4038	1	
COLPR	16441	4039	1	N.º de coluna para PRINT
LINPR	16442	403A	1	N.º de linha para PRINT
CDFLAG	16443	403B	1	Flags
PRBUFF	16444	403C	33	Buffer para impressora
MEMBOT	16477	405D	30	Área de memória para cálculo
SPAREZ	16507	407B	2	Bytes vagos

APÊNDICE E

MAPA
DA MEMÓRIA

- 0000 — Inicialização do Sistema
- 0008 — Manipulação de erros
- 0010 — Rotina que imprime um caractere
- 0018 — Carrega no acumulador o byte apontado pela variável no endereço 16406
- 0020 — Carrega no acumulador o próximo byte apontado pela variável no endereço 16406
- 0028 — Rotina de Cálculo
- 0030 — Incrementa a área de variáveis com os números de byte em BC
- 0038 — Rotina de interrupção para mostrar na tela uma linha
- 0066 — Rotina de interrupção para mostrar a tela no modo SLOW
- 007E — Tabelas dos caracteres normas do teclado
- 00A5 — Tabela dos caracteres SHIFT do teclado
- 00CC — Tabela das funções do BASIC
- 00F3 — Tabela dos caracteres gráficos
- 0111 — Tabela das palavras-chaves do BASIC
- 01FC — Rotina de atualização dos comandos SAVE e LOAD
- 0207 — Rotina de determinação da velocidade (SLOW ou FAST)
- 0229 — Rotina principal do display
- 0292 — Rotina do display no modo SLOW
- 02B5 — Rotina do display no modo FAST
- 02BB — Rotina de varredura do teclado
- 02E7 — Rotina de RESET do SCL
- 02F4 — Comando SAVE
- 0340 — Comando LOAD
- 03A2 — Teste de BREAK do comando LOAD
- 03C3 — Comando NEW
- 0419 — Rotina de edição das linhas de programação
- 0454 — Rotina do cursor
- 04B2 — Rotina de execução do programa em BASIC
- 052B — Rotina de construção do Sistema E-LINE
- 05C4 — Rotina de ordenação da edição
- 063E — Rotina principal de edição
- 072C — Comando LLIST
- 0730 — Comando LIST
- 0745 — Rotina de impressão de uma linha BASIC
- 07BD — Rotina de decodificação do teclado
- 07F1 — Rotina de impressão de um caractere
- 0808 — Rotina de impressão de um caractere no vídeo
- 0851 — Coloca um caractere no buffer da impressora
- 0869 — Comando COPY
- 08F5 — Teste dos parâmetros de PRINT AT

- 0918 — Rotina de expansão do display
- 094B — Rotina de impressão das palavras-chaves do BASIC
- 09AD — Rotina de organização das variáveis
- 09D8 — Determinação do endereço de uma linha de programa
- 0A2A — Comando CLS
- 0A98 — Rotina de impressão do número da linha
- 0ACB — Comando LPRINT
- 0ACF — Comando PRINT
- 0B6B — Rotina de impressão de uma string
- 0BAF — Comando PLOT e UNPLOT
- 0C0E — Comando SCROLL
- 0C29 — Tabela de sintaxe dos comandos
- 0CDC — Comando STOP
- 0DAB — Comando IF
- 0DB9 — Comando FOR
- 0E2E — Comando NEXT
- 0E6C — Comando RAND
- 0E7C — Comando CONT
- 0E81 — Comando GOTO
- 0E92 — Comando POKE
- 0ED8 — Comando RETURN
- 0F23 — Comando FAST
- 0F2B — Comando SLOW
- 0F32 — Comando PAUSE
- 0F46 — Teste de BREAK do comando SAVE
- 1321 — Comando LET
- 1409 — Comando DIM
- 149A — Comando CLEAR
- 151D — Arquivo o acumulador do Stack do calculador
- 1520 — Arquivo o par BC no Stack do calculador
- 158A — Rotina de manipulação dos cálculos de ponto flutuante
- 174C — Rotina de subtração para números de 5 bytes
- 1755 — Rotina de adição para números de 5 bytes
- 1706 — Rotina de multiplicação para números de 5 bytes
- 1882 — Rotina de divisão para números de 5 bytes
- 1815 — Tabela das funções
- 199D — Calculador de ponto flutuante
- 1E00 — Tabela de definição dos caracteres
- 4000 — Início da RAM
- 4082 — Início do programa em BASIC
- DFILE — Início do arquivo de tela
- VARS — Início da região de variáveis

ELINE — Linha sendo digitada + espaço de trabalho
PILFUN — Início da pilha de cálculo
PILFIM — Início da pilha de cálculo
SP — Início da pilha para endereçar sub-rotinas
RAMTOP — Início do espaço que pode ser reservado para rotinas
em linguagem de máquina

APÊNDICE F
SUBROTINAS
DE CÁLCULO

LITERAL	NOME	ENDEREÇO
(hexadecimal)		(hexadecimal)
01	TROCA	1A72
02	APAGA	19E3
03	SUBTRAÇÃO	174C
04	MULTIPLICAÇÃO	17C6
05	DIVISÃO	1882
06	POTENCIAÇÃO	1DE2
07	OU	1AED
08	E	1AF3
0F	SOMA	1755
18	MUDA SINAL	1AA0
1C	SENO	1D49
1D	COSSENO	1D3E
1E	TANGENTE	1D6E
1F	ARCO-SENO	1DC4
20	ARCO-COSSENO	1DD4
21	ARCO-TANGENTE	1D76
22	LN (Logaritmo neperiano)	1CA9
23	EXPOENTE	1C5B
24	INTEIRO	1C46
25	RAIZ QUADRADA	1DDB
26	SINAL	1AAF
27	MÓDULO	1AAA
28	PEEK	1ABE
29	USR	1AC5
2C	NÃO	1AD5
2D	DUPLICAÇÃO	19F6
2E	Y MÓDULO X	1C37
32	MENOR QUE ZERO	1ADB
33	MAIOR QUE ZERO	1ACE
34	FIM CÁLCULO	002B
35	ARGUMENTO REDUZIDO	1D18
36	ARREDONDAMENTO PARA ZERO	18E4
A0	COLOCA ZERO	1A51
A1	COLOCA UM	1A51
A2	COLOCA MEIO	1A51
A3	COLOCA $\pi/2$	1A51
A4	COLOCA DEZ	1A51
CO a C5	ARMAZENA NA MEMÓRIA PARA CÁLCULO	1A63
E0 a E5	LÊ NA MEMÓRIA PARA CÁLCULO	1A45

Caro leitor:

*Se você quiser receber
gratuitamente o boletim
informativo da editora
URANIA, contendo jogos,
programas e novidades,
envie seu nome e endereço
completos para:*

**URANIA - RAND USR
Av. Faria Lima, 1451 cj 31
CEP 01451**



impresso na
planimpress gráfica e editora
rua anhaia, 247 - s.p.

USANDO LINGUAGEM DE MÁQUINA

- Uma obra original realmente didática para aprendizado e consulta.
- Muitas dicas para o uso de linguagem de máquina em computadores compatíveis com Sinclair (Ringo, ZX-81, CP-200, TS-1000, NEZ-8000, TK-82C, TK-85, TK-83).
- Como usar as sob-rotinas da ROM, inclusive para cálculos científicos.
- Sugestões para o uso "balanceado" do BASIC e da linguagem de máquina.
- Muitos programas em linguagem de máquina disassemblados e comentados linha por linha:

MESTRE 2 — Um monitor totalmente em linguagem de máquina que se auto-desloca para a posição mais conveniente da RAM.

JOGO DE PALITOS — Um exemplo interessantíssimo de algoritmo binário para estabelecer uma estratégia inteligente.

UM PLOT RÁPIDO — Como desenvolver uma sub-rotina muito melhor que a já existente na ROM e muitos outros, inclusive com um jogo auto-programável (inteligência artificial).

Mario Schaeffer é o que poderíamos chamar de "executivo de alto nível".

Formado em Engenharia Eletrônica no ITA e com curso de mestrado na POLI (SP), trabalhou como analista de sistemas e pesquisador em projetos espaciais (INPqE).

No Brasil, porém, todo engenheiro bem sucedido deixa de ser engenheiro. Seguindo este roteiro, Mário passou a ser o eficiente gerente de Marketing de uma Multinacional, dedicando-se mais à análise de mercado que ao cálculo diferencial.

Mesmo no Brasil, porém, as pessoas inteligentes não deixam de sê-lo, e o Mário, para sorte de muitos aficionados conservou o hobby da computação.

Este volume é o resultado da paixão que o Mário dedicou a um microcomputador da linha SINCLAIR com 2k de RAM.

Nele o Mário mostra, com muita inteligência e didática como fazer verdadeiros milagres de programação utilizando Linguagem de Máquina, fornecendo muitíssimos exemplos de aplicações e brindando o leitor com uma série de programas úteis e divertidos.

