livraria sistema ltda.

Primeiros Passos na Programação em Linguagem de Máquina — M. Silveira

2ª EDIÇÃO REVISADA

Primeiros Passos na Programação em Linguagem de Máquina

M. Silveira

Especialmente para TK-82C, TK-83, TK-85, CP-200

livraria sistema ltda.

ÍNDICE

INTRODUÇÃO

O objetivo do presente livro é possibilitar ao leitor programar em linguagem de máquina. Para isto, parte-se da suposição de que ele já consegue escrever seus programas em BASIC. As observações e instruções que são aqui fornecidas supõem que seu computador possua um microprocessador Z80A (fabricado pela Zilog) e, em especial, que o ROM (a memória escrita pelo fabricante do computador) seja semelhante ao do ZX-81 ou do TIMEX 1000 produzidos pela Sinclair e pela Timex, respectivamente. No Brasil isto indica os TKs 82-C, 83 e 85, o CP-200 e o NE-Z8000.

A aquisição desta nova linguagem permitirá, no presente caso, uma rapidez maior na execução dos programas, assim como uma ocupação menor da área onde se pode escrever (o RAM).

A razão disto é fácil de entender. Quando se escreve um programa em BASIC, ele tem de ser guardado e traduzido em linguagem de máquina, a única que o Z80 entende. Ora, a tradução leva um certo tempo para ser executada. Podendo-se programar em linguagem de máquina, se estará não só utilizando uma menor área de programação (não há necessidade de guardar as instruções em BASIC), como poupando um considerável tempo (não havendo necessidade de tradução, o tempo é, em média, vinte vezes menor).

Além disso, a linguagem de máquina também possibilita uma proteção maior para o **software** que por ventura se venha a criar.

Os capítulos devem ser lidos na sua ordem seqüencial e o leitor só deve passar para o seguinte se for capaz de resolver todos os exercícios propostos no final de cada parte. Para solucioná-los, deve-se utilizar o computador. Só assim o programador será capaz de adquirir domínio da linguagem de máquina.

Todos os programas fornecidos no presente livro têm um objetivo pedagógico. Convém procurar modificá-los ou utilizá-los em seus próprios programas.

É preciso muita paciência. Diferentemente do que ocorre com BASIC, não há aqui um leitor de erros. Se ocorrer algum, na tela aparecerão as mais variadas coisas ou se perderá o controle do teclado, ou se deparará com ambas as situações. A única maneira de contornar esse desastre (**crash**), é desligar o computador e recomeçar tudo. Portanto, antes de rodar o programa, procura-se gravá-lo, pois, caso suceda um **crash**, bastará recolocá-lo e, através de seu estudo, detectar a possível causa do erro. Em qualquer caso, não há motivo para preocupar-se — é impossível danificar o aparelho, mesmo quando os programas são escritos em linguagem de máquina.

Nunca é demais insistir: a simples leitura do livro não irá transformar o leitor num bom programador. Apenas a prática constante pode contribuir para torná-lo competente no uso da linguagem de máquina. Por conseguinte, mãos à obra!

CAPÍTULO I:
NÚMEROS BINÁRIOS E HEXADECIMAIS

No manual recebido juntamente com o seu computador existe um capítulo sobre o emprego de linguagem de máquina. Nele você aprende que as rotinas nesta linguagem podem ser executadas a partir da utilização da função USR. Durante o texto explicaremos todas as considerações que são feitas a respeito do uso dessa função. Por ora, queremos salientar apenas o fato de que o emprego de linguagem de máquina é feito através do uso de uma rotina que deve, de alguma forma, retornar ao programa em BASIC.

Por exemplo, digite e rode o seguinte programa:

```
10 POKE 18000,62
20 POKE 18001,1
30 POKE 18002,205
40 POKE 18003,8
50 POKE 18004,8
60 POKE 18005,24
70 POKE 18006,249
80 RAND USR 18000
```

O que aconteceu?

A tela ficou rapidamente ocupada com o caractere número 1

(Apêndice 1), dando indicação de erro tipo 5, na linha 80. Se você digitar CONT, a situação se repetirá. Em outras palavras, não há mais espaço na tela. A instrução CONT limpa a tela, mas esta fica de novo imediatamente repleta.

Mais tarde, você entenderá por que isto está ocorrendo. Mas antes de continuarmos, substitua a linha 20 por 20 POKE 18001,3 e rode o programa. Agora, a tela está totalmente ocupada pelo caractere número 3.

O que estamos fazendo?

Simplesmente colocando numa série de endereços, de 18000 a 18006, certos valores em decimal. O conjunto deles é traduzido em seqüências de números binários (números da forma 0 ou 1), ou seja, num programa em linguagem de máquina. A execução é feita através da CPU diretamente, isto é, através da unidade central de processamento, sem necessidade de ser traduzida pelo programa residente (o programa contido no ROM). No caso presente, teríamos:

```
0011 1110
0000 0001
1100 1101
0000 1000
0000 1000
0001 1000
1111 1001
```

Cada um desses dígitos, um ou zero, é um dígito binário, um BIT (em inglês **binary digit**). Estão divididos em seqüências de 8 BITs contíguos, isto é, cada seqüência constitui um BYTE. Portanto, nosso programa consiste de sete BYTEs. Contudo, como podemos facilmente observar, não é muito viável programar em números binários. No entanto, como a CPU só interpreta este tipo de número (criado fisicamente por diferença de voltagem), é preciso saber como passar de decimais, utilizados nas instruções POKE, para binários.

Como fazer a conversão?

Basta recordar que os números decimais, números de base 10, podem ser escritos como somatória de potências de dez. Por exemplo, 62 é igual a: $6x10^1 + 2 \times 10^0$.

No caso do número binário, número de base 2, teremos uma somatória de potências de dois. Por exemplo, 62 em decimal é igual a 0011 1110 em binário; ou seja, igual a:

$$0 \times 2^7 + 0 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0.$$

Por que estamos utilizando as potências de dois até a sétima casa?

A resposta é simples: o Z80A pode manipular simultaneamente até 8 BITs, ou seja, a palavra do Z80A tem a extensão de um BYTE.

Portanto, só podemos fazer POKES de valores entre 0 e 255 (valores entre -255 e -1 são aceitos, contudo, antes de serem manipulados, eles são somados a 256). 255 tem a seguinte representação binária:

```
BIT    7 6 5 4 3 2 1 0
       | | | | | | | |
       1 1 1 1 1 1 1 1
```

O BIT 7 é chamado de BIT mais significativo (MSB), enquanto o BIT 0 é o menos significativo (LSB).

Entretanto, como se pode notar, estamos separando um BYTE em dois conjuntos de quatro BITs. Qual a razão disto?

A resposta mais uma vez é bastante simples. Como não podemos trabalhar com dígitos binários (a possibilidade de cometermos um erro é muito grande, aliada à dificuldade em detectá-lo posteriormente), a mente humana necessita de um tipo de número que esteja mais de acordo com a sua natureza e que seja facilmente convertido em binário e vice-versa. Para isto existem os números hexadecimais.

Eles são números da forma 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E e F. Portanto, coincidem com os decimais até a representação do número 9. A partir daí, de 10 a 15, são representados pelas letras A, B, C, D, E e F, respectivamente. À semelhança dos números vistos até agora, eles também podem ser expressos como uma somatória de potência; neste caso, potências de dezesseis (números de base dezesseis).

Por exemplo, 62 em decimal é igual a 3E em hexadecimal, isto é, $3E = 0 \times 16^3 + 0 \times 16^2 + 3 \times 16^1 + E \times 16^0$.

Procure entender como chegamos a esse número. Partimos do fato de que 3 multiplicado por 16 é igual a 48; para chegarmos a 62, faltam 14. A representação hexadecimal de 14 é E. Logo, 62 em HEX (hexadecimal) é representado por 003E ou simplesmente 3E. O fato de antepormos dois zeros à frente de 3E está ligado ao uso que faremos desses números no decorrer desta explanação. Como o Z80A utiliza palavras de oito BITs, ele pode referir-se diretamente a endereços de até 16 BITs, ou seja, a 65536 localidades distintas. Em HEX isto indica desde o endereço 0000H até FFFF, onde FFFF é igual a 65535 em decimal. O primeiro par de números constitui o BYTE mais significativo (abreviado por HI), enquanto o segundo é o menos significativo (LO). Em outros termos, palavras de 16 BITs são tratadas como se fossem duas palavras de 8 BITs cada uma.

A relação entre números binários e hexadecimais é bastante simples. Procure justificar a seguinte tabela.

BINÁRIO	HEXADECIMAL
0000 0000	00
0000 0001	01
0000 0010	02
0000 0011	03
0000 0100	04
0000 0101	05
0000 0110	06
0000 0111	07
0000 1000	08
0000 1001	09
0000 1010	0A
0000 1011	0B
0000 1100	0C
0000 1101	0D
0000 1110	0E
0000 1111	0F

Para encontrar o binário equivalente a 3E em hexadecimal, basta consultar a tabela e teremos 0011 1110. Em outras palavras, para cada grupo de quatro dígitos binários em seqüência, colocamos um dígito em HEX; e vice-versa: para cada dígito em HEX, colocamos uma seqüência de quatro dígitos em binário.

Também é possível programar o seu computador para que ele faça essas conversões. Digite o seguinte programa:

```
 1 PRINT TAB 1; "DEC";TAB 8;"HEX"
 5 DIM H(2)
10 DIM H$(2)
15 FOR F=0 TO 255
20 LET N=F
25 LET H(1)=INT (N/16)
30 LET N=N-H(1)*16
35 LET H(2)=N
40 FOR A=1 TO 2
45 LET H$(A)=CHR$ (H(A)+28)
50 NEXT A
55 PRINT TAB 1;F;TAB 9;H$
60 NEXT F
```

Ele converte números decimais em hexadecimais no intervalo compreendido entre 0 e 255. Você encontra a tabela gerada por este programa no Apêndice 2. Ela será de grande utilidade no decorrer da explanação.

Agora tente o seguinte programa:

```
10 DIM H$(4)
15 INPUT H$
20 LET D=0
25 FOR I=1 TO 4
30 IF CODE H$(I) <28 OR CODE H$
(I) > 43 THEN GOTO 15
35 LET D=D+16**(4-I)*(CODE H$(I) -28)
40 NEXT I
45 PRINT D
```

Ele converte números em hexadecimal (no intervalo compreendido entre 0000H a FFFF) em dígitos decimais.

Finalmente digite este programa:

```
10 INPUT X
20 FOR N=7 TO 0 STEP -1
30 LET D=2**N
40 PRINT CHR$ (28+INT (X/D));
50 LET X=X-INT (X/D)*D
60 NEXT N
```

Ele converte números decimais em dígitos binários.

Todos estes programas estão baseados no fato de que a função CHR$ **n** permite obter o caractere cujo código numérico é **n**, bem como no fato de que a função CODE **x** (onde **x** é uma variável alfanumérica) possibilita saber o código numérico do primeiro caractere contido na variável alfanumérica **x**. Desta forma, o primeiro programa (DEX) associa a cada número decimal dois valores guardados no **array** H. A instrução na linha 45 permite passar de H para H$. O número 28 é somado para que desta forma, possamos obter todos os caracteres desde 0 até F. No segundo programa, dimensionamos um **array** alfanumérico H$ e testamos se o valor colocado está compreendido entre 0000 e FFFF (linha 30). Em seguida, calculamos o valor em decimal. O último programa calcula o número binário correspondente a X e impreme 0 (código numérico=28) ou 1 (código numérico=29). conforme a instrução INT(X/D) resulte zero ou um.

EXERCÍCIOS

1. Escreva um programa semelhante a DEX, que converta números decimais em hexadecimais, no intervalo compreendido entre 0 e 65535.
2. Suponha que você deva colocar certas instruções em linguagem de

máquina nos endereços 16514, 30000, 65535. Qual será a representação hexadecimal desses endereços?

3. Seu programa em linguagem de máquina necessita saber o conteúdo de certas variáveis do sistema colocadas nos endereços 16396, 16438 e 16439. Qual a representação hexadecimal desses endereços?
4. Sabe-se que a rotina que verifica se uma tecla foi apertada está em 02BB no ROM. Qual o seu endereço em decimal?
5. O seguinte programa permite realizar SCROLL. Suas instruções são: 2A0C40E511210019D101D602EDB0C9. Coloque-as em representação decimal. (Dica: a cada par de número hexadecimal — composto sempre de dois dígitos — corresponde um número decimal.)

CAPÍTULO II:
CARREGAMENTO DE PROGRAMAS EM LINGUAGEM DE MÁQUINA

Como o seu computador não possui um **assembler** (um programa que converta mnemônicos, isto é, instruções em símbolos facilmente reconhecíveis pela mente humana, em código binário, para que ele seja executado), é preciso fazê-lo manualmente. Isto é chamado de **assembly** manual e trata-se de traduzir cada instrução em hexadecimal. Para isto, vamos precisar de um programa que carregue esses números. Neste capítulo, você aprenderá a fazer este programa, assim como terá indicações sobre os locais do RAM que deve utilizar para colocar suas rotinas em linguagem de máquina.

No manual que acompanha o seu computador há diversas sugestões. Vamos examiná-las. A primeira refere-se ao uso de instruções REM. Procure entendê-la, a partir das considerações que faremos a respeito do seguinte programa que você deve digitar.

```
  1 REM 1234567
 90 LET E=16514
100 LET M$=""
110 IF M$="" THEN INPUT M$
120 IF M$="P" THEN STOP
130 POKE E,16*CODE M$+CODE M$(2) -476
140 LET E=E+1
150 LET M$=M$(3 TO)
160 GOTO 110
```

A primeira instrução é REM seguida de sete números. Esses números estão no lugar das instruções que iremos digitar. Como são sete as instruções, colocamos números de 1 até 7. Poderíamos ter colocado quaisquer caracteres. A forma escolhida tem a vantagem de evitar erros. Se fossem vinte instruções, teríamos digitado:

1 REM 12345678901234567890

A segunda instrução dá à variável E o endereço da primeira localidade após a instrução de REM.

Colocar o REM no início do programa apresenta vantagens práticas. Pode-se facilmente localizar o endereço do primeiro BYTE do programa BASIC. Ele é obtido através da instrução PRINT PEEK 16396 + 256* PEEK 16397 que dá 16509 como resultado. A este número devemos somar 5, pois, dois BYTEs são gastos para definir o número da linha (no endereço 16510 fica o valor HI e no endereço 16510 o valor LO), dois BYTEs para dar o comprimento da linha (no endereço 16511 fica o valor LO e no endereço 16512 fica o valor HI) e um BYTE para escrever REM. Se você quiser testar, basta entrar com PEEK 16514 e encontrará o valor 29 que é o código numérico do caractere 1.

As três instruções seguintes, nas linhas de números 100, 110 e 120, definem uma **string**, inicialmente vazia, o que permitirá a você introduzir a instrução em hexadecimal; através da digitação da tecla "P", será possível interromper o carregamento do seu **assembly** manual.

A linha 130 permite colocar no endereço E a instrução em hexadecimal. O resultado final será do tipo POKE em E um certo número em decimal. Portanto, a seqüência 16* CODE M$+CODE M$(2)-476 transforma um número hexadecimal em um número decimal. Para entender como se chegou a essa fórmula, basta recordar que as instruções podem ir desde 00 até FF. Se você consultar a tabela de caracteres, verá que o código de 0 é 28 e que o de F é 43. Como a função CODE dá o código numérico do primeiro caractere contido na variável alfanumérica M$, o resultado no caso da instrução 00 em hexadecimal será: 16* CODE 0 + CODE 0 - 476, ou seja, zero. Este era exatamente o resultado esperado, pois do número 00 em HEX corresponde 0 em decimal. Caso a instrução fosse FF, teríamos: 16* CODE F + CODE F - 476, ou seja, 255. Ora, 255 é o maior número que você pode utilizar numa instrução "POKE E, v"onde E é o endereço em decimal, e v, uma quantidade numérica também em decimal, até 255. Por outro lado, poderíamos ter usado a expressão "(CODE M$-28)*16 + CODE M$(2)-28". Contudo como se pode facilmente verificar, a expressão que estamos utilizando nada mais é do que o desenvolvimento desta.

A linha 140 permite passar do endereço E para o endereço seguinte, E+ 1, enquanto a instrução 150 possibilita a leitura do restante da **string**. A última instrução permite a repetição do programa até que se digite a letra "P".

Entra-se agora com as instruções em hexadecimal da rotina apresentada no início do capítulo anterior; ou seja, quando o programa de carregamento pedir para colocar a **string**, **digite** . 3E01CD080818F9P. Ele terminará com a indicação 9/120. Digite agora RAND USR 16514. Perceba que a linha 1, após o carregamento, apresenta o seguinte aspecto:

1 REM Y LN / RAND

A segunda sugestão refere-se ao uso de uma **string**. Para exemplificá-la, vamos recorrer à área onde o computador guarda as variáveis para nela colocar a nossa rotina. Antes de fazê-lo, é preciso dimensionar a área a ser ocupada. No caso presente, isto equivale a digitar DIM A$(7). Em seguida, precisamos estabelecer o endereço inicial. Para tanto, recorremos à variável VARS que está no endereço 16400 conforme se pode constatar consultando o Apêndice 3. Cada instrução será introduzida através da função CHR$ **n**, onde . é o número em decimal correspondente a um certo mnemônico do **assembly** manual. Desta forma, o mesmo programa apresentará agora as seguintes instruções:

```
100 DIM A$(7)
110 LET E=PEEK 16400+256*PEEK 16401+6
120 LET A$(1) =CHR$ 62
125 LET A$(2) =CHR$ 1
130 LET A$(3) =CHR$ 205
135 LET A$(4) =CHR$ 8
140 LET A$(5) =CHR$ 8
145 LET A$(6) =CHR$ 24
150 LET A$(7) =CHR$ 249
160 RAND USR E
```

Na linha 110 fomos obrigados a somar 6 ao endereço fornecido pela variável do sistema VARS, porque era necessário levar em conta os BYTEs que gastamos para dimensionar a **string** A$, ou seja: 1 BYTE para dar nome à **string**; 2 BYTEs para dar o número total de elementos; 1 BYTE para escrever o número de dimensões; 2 BYTEs para dar a dimensão da **string**.

A terceira sugestão consiste em reservar um espaço no topo do RAM. Para isto, devemos calcular este endereço através da variável do sistema RTP, e subtrair o resultado do número de instruções que iremos utilizar.

Com este objetivo em mente, digite e dê entrada ao seguinte programa:

```
100 LET E=PEEK 16388+256*PEEK 16389
200 LET E=E-7
300 POKE 16388,E-256*INT (E/256)
400 POKE 16389,INT (E/256)
500 NEW
```

Ele irá redefinir o topo do RAM deixando sete espaços livres para a rotina em linguagem de máquina. Agora basta entrar com este programa:

```
100 LET E=PEEK 16388+256*PEEK 16389
110 LET M$=""
120 IF M$="" THEN INPUT M$
130 IF M$="P" THEN STOP
140 POKE E,16*CODE M$+CODE M$(2)-476
150 LET E=E+1
160 LET M$=M$(3 TO )
170 GOTO 120
```

Quando ele pedir para entrar com a **string** M$, digite 3E01CD0 80818F9P. Para chamar esta rotina em linguagem de máquina, entre com RAND USR (E-7). Note que o endereço inicial é E menos 7, ou seja, E menos o **tamanho da rotina**.

Você certamente está curioso em saber se tais sugestões são equivalentes, ou se há um ganho real ao se usar uma em detrimento de outra.

A pior sugestão é a segunda, a de colocar a rotina em linguagem de máquina na área reservada para as variáveis. Pois, embora você a proteja de ser listada diretamente e possa gravá-la em fita magnética e retorná-la posteriormente ao computador junto com o programa em BASIC, não poderá utilizar os comandos RUN e CLEAR (ambos colocariam todos os valores da **string** iguais a zero). Além disso, os endereços dentro da rotina em linguagem de máquina irão mudar durante a sua execução, o que limitará o uso a um só tipo de instrução de salto (salto relativo, como veremos adiante).

Na primeira sugestão, a rotina em linguagem de máquina está num lugar fixo da memória, será gravada juntamente com o programa BASIC, podendo retornar, quando se desejar, ao computador. A única desvantagem é a de poder ser listada, embora isso não deva causar muita preocupação, pois, como já vimos, além de ela consistir numa série de caracteres (o que a torna ininteligível para o não iniciado), há

um tipo de instrução a ser examinada adiante, que não aparece na listagem assim como as cinco instruções que lhe seguem.

A última sugestão tem a vantagem imediata de ser a prova de comandos do tipo RUN, CLEAR e, mesmo, NEW. Contudo, a rotina não pode ser gravada em fita magnética diretamente, e os endereços serão modificados caso se passe a utilizar uma extensão de memória. Também aqui, quando se desejar fazer um programa que rode com ou sem extensão, devemos nos limitar a usar apenas saltos relativos na rotina em linguagem de máquina.

EXERCÍCIOS

1. Carregue a rotina em linguagem de máquina dada no exercício 5 do primeiro capítulo, utilizando as três sugestões de carregamento que acabamos de estudar.
2. Escreva um programa de carregamento que utilize a área das variáveis para guardar a rotina em linguagem de máquina, porém usando a instrução POKE.

CAPÍTULO III:

ALGUNS MNEMÔNICOS EM LINGUAGEM DE MÁQUINA E A SUA UTILIZAÇÃO

Até agora só lidamos com números. Contudo, eles correspondem a mnemônicos. Pode-se verificar isto consultando o apêndice do manual que veio junto com o seu computador. Por exemplo, a rotina em linguagem de máquina que utilizamos nos dois primeiros capítulos apresenta os seguintes mnemônicos:

número em HEX	**mnemônico**
3E	LD A, N
Ø1	N = 1
CD	CALL NN
Ø8	N = 8
Ø8	N = 8
18	JR DIS
F9	DIS = 7

Vamos demoninar a tabela formada por números hexadecimais de código objeto (o código que colocaremos no computador) e a tabela dos mnemônicos de código fonte (as instruções dadas numa forma que fica fácil de ser compreendida pela mente humana).

No decorrer desta explanação, estudaremos algumas das diversas instruções do Z8ØA, que na sua forma básica são 245, e que, através

das diversas permutações, chegam a mais de 800. Não é preciso se esforçar em decorá-las; para isto existem tabelas (ver Apêndice 4). Veremos, nesta obra de caráter introdutório, apenas um subconjunto dessas instruções, deixando, para uma abordagem posterior, a investigação de todo o conjunto.

Cada instrução define uma seqüência de operações a serem realizadas. Elas podem ter o comprimento variando entre um a quatro BYTEs. Podemos reuni-las em seis grupos (dos quais aqui estudaremos apenas os cinco primeiros):

I) Grupo de Transferência de Dados;
II) Grupo de Operações Aritméticas e Lógicas;
III) Grupo de Salto, Chamada e Retorno;
IV) Grupo de Rotação e de Deslocamento;
V) Grupo de Manipulação de BIT;
VI) Grupo de Controle de Entrada e Saída.

Os BYTEs presentes nas instruções podem ser divididos em BYTEs de:

a) Instrução;
b) Dados;
c) Endereços;
d) Referências a Dispositivos;
e) Deslocamentos.

Há doze formas de combinar esses BYTEs em conjuntos de um a quatro BYTEs conforme a tabela abaixo revela.

Número de BYTEs	Tipo de BYTE
1	Operação
2	Operação-Operação
	Operação-Dado
	Operação-Deslocamento
	Operação-Dispositivo
3	Operação-Dado-Dado
	Operação-Endereço(LO)-Endereço(HI)
	Operação-Operação-Deslocamento
4	Operação-Operação-Dado-Dado
	Operação-Operação-Endereço(LO)-Endereço(HI)
	Operação-Operação-Deslocamento-Dado
	Operação-Operação-Deslocamento-Operação

A seguir, daremos dois exemplos com o objetivo de explicar como se passa de seqüências de números HEX para mnemônicos, e como através disto pode-se chegar a compreender aquilo que a rotina em

linguagem de máquina pretende realizar. Você não precisa se inquietar caso não entenda as instruções — elas serão explicadas, em detalhe, nos capítulos seguintes.

Mas, antes de passar aos exemplos, cumpre fazer as seguintes observações:

a) operações consistem em ações que desejamos implementar no computador;
b) a CPU (unidade central de processamento) é composta de uma unidade lógico-aritmética, de uma unidade de controle e de um **clock** (relógio) que marca o tempo que cada ação leva para ser executada; neste sentido, mesmo a instrução NOP (não faça nada) leva um certo tempo para ser implementada;
c) a CPU está ligada a dispositivos de entrada (geralmente o teclado), a dispositivos de saída (em geral, um monitor ou uma impressora) e a unidades de memória (o ROM, o RAM, ou, mesmo, a memória guardada em fita magnética);
d) conforme você pode constatar, toda vez que uma operação envolve um endereço ou um dado de 16 BITs, o primeiro BYTE refere-se **sempre** ao BYTE **menos** significativo, e não ao mais significativo, como se poderia intrisicamente pensar.

Vamos analisar, em primeiro lugar, a rotina que usamos até este momento. Ela pode ser colocada na seguinte forma:

código objeto	**código fonte**	**rótulo**	**comentário**
3E01	LD A,N	INÍCIO	oper-dado
CD0808	CALLNN		oper-end (LO)-end(HI)
18F9	JR INÍCIO		oper-desl

A primeira instrução — a instrução LOAD, abreviada LD — pode ser entendida, dentro de certos limites, como semelhante à instrução LET em BASIC. Assim, LD A,01 consiste em colocar em A, chamado de ACUMULADOR, o valor 01. Pertence, portanto, ao grupo I: o grupo das operações de transferências de dados.

A segunda instrução — CALL NN — pode ser considerada correspondente à instrução GOSUB em BASIC. Como é evidente, pertence ao grupo V. Através dela estamos chamando uma sub-rotina no ROM, presente no endereço 8+8*256, isto é, no endereço 2056 em decimal. Note, mais uma vez, que o primeiro BYTE do endereço é o BYTE menos significativo.

A última instrução — JR DIS — refere-se à instrução salte ("JUMP"). Ela altera o PC (contador de programa), fazendo com que o programa passe a ser executado num local determinado pelo deslocamento. No caso presente, como veremos mais adiante, trata-se de um salto relativo, que dá ao PC um valor que leva o programa de

novo para a instrução LD A,N. A instrução "JUMP" é, dentro de certos limites, semelhante à instrução GOTO em BASIC.

Em suma, nossa rotina consiste em carregar o acumulador de um certo valor, em seguida, ela chama uma sub-rotina no ROM que imprime o caractere correspondente a esse valor, para, finalmente, repetir o procedimento, indefinidamente, através de um salto relativo que a leva de novo para o seu início. É exatamente por isso que a execução da rotina é sempre interrompida com indicação de erro do tipo 5.

Esse trabalho de desmontar a rotina em linguagem de máquina chama-se **disassembly**. Vamos realizá-lo em relação à rotina, dada no quinto exercício do capítulo I. Se a colocarmos no endereço 30 000 (supondo que sua máquina possui 16 K de memória), teremos a seguinte tabela:

```
ENDERECO  COD.OBJETO  COD.FONTE
30000     2A0C40      LD HL, (16396)
30003     E5          PUSH HL
30004     112100      LD DE, 33
30007     19          ADD HL,DE
30008     D1          POP DE
30009     01D602      LD BC, 726
30012     EDB0        LDIR
30014     C9          RET
```

A primeira instrução — LD HL, (NN) — consiste em colocar no par de registradores HL o conteúdo do endereço indicado por NN. Este endereço é 12 (0C em HEX) + 64 (40 em HEX) * 256 = 16396. Consultando o Apêndice 3, descobre-se que 16396 é o endereço da variável do sistema DFILE, o qual dá o endereço do primeiro caractere no arquivo de imagem. Portanto, o par de registradores HL está apontando para esse endereço. Como já vimos, os endereços são sempre dados em 16 BITs. Logo, a instrução LD HL (NN) é semelhante em BASIC a:

LET L=PEEK N LET L=PEEK 16396

no caso presente , a

LET H=PEEK N+1 LET H=PEEK 16397

A segunda instrução — PUSH HL (acrescente) — refere-se à pilha de dados ("STACK"). Esta pilha nada mais é do que uma forma de guardar os dados. Ela obedece a uma regra chamada "LIFO", abreviatura da expressão inglesa **last in, first out** (o último a entrar é o primeiro a sair).

Em outras palavras, pode-se compará-la a uma pilha de livros: para retirar um, é preciso começar pegando o que está sobre todos os outros, a fim de evitar o risco de ver a pilha desabar. A instrução PUSH HL é semelhante em BASIC a fazer um POKE no endereço dado pelo SP (ponteiro da pilha); isto é, a entrar com:

POKE SP-1, H
POKE SP-2, L

Portanto, quando acrescentamos algo na pilha, o SP diminui. O novo SP é igual ao anterior subtraído de dois. Como o valor L foi o último a entrar, ele será o primeiro a sair.

A terceira instrução - LD DE, NN — coloca no par de registradores DE o valor 33 (21 em HEX). Note que o BYTE mais significativo é zero, e que ele vem **depois** do BYTE menos significativo.

A quarta instrução — ADD HL, DE adiciona ao conteúdo de HL o valor presente em DE. Desta forma, HL contém agora o endereço do último BYTE da primeira linha do arquivo de imagem.

Na quinta instrução — POP DE (remonte) — estamos, inversamente à segunda instrução, retirando da pilha dados para colocar no par de registradores DE. Como POP aumenta o SP de dois, teremos que a instrução POP DE, no presente caso, é semelhante em BASIC a:

LET D = PEEK (SP-1) = H
LET E = PEEK (SP-2) = L

Logo, DE guarda agora o valor de HL anterior à quarta instrução, ou seja, o endereço do primeiro BYTE do arquivo de imagem. Note que, após esta instrução, o SP voltou ao seu valor original, isto é, anterior à execução da segunda instrução. Quando isto não ocorrer, o sistema irá sofrer um desastre (um **crash**).

A sexta instrução — LD BC, NN — coloca, no par de registradores BC, o valor 214 (D6 em HEX) + 2 * 256 = 726.

A penúltima instrução — LDIR —, que contém dois BYTEs, vai transferir um bloco de dados, que se inicia no endereço indicado por HL, para o endereço iniciado com o endereço indicado por DE. O número total de transferências a serem feitas é dado pelo par de registradores BC. Durante a execução desta instrução, os valores de HL e de DE são incrementados, e o de BC, diminuídos até chegar a zero.

A última instrução — RET — possibilita que haja um retorno para o programa em BASIC.

O que realiza, em suma, este conjunto de instruções em linguagem de máquina?

Ele, simplesmente, faz um SCROLL.

Podemos agora, descrever, com maiores detalhes, a organização do Z80A.

Ele possui dois conjuntos de oito registradores cada um. O principal, formado pelos registradores A, F, B, C, D, E, H e L, pode ser afetado diretamente pela rotina. O alternativo, composto pelos registradores A', F', B', C', D', E', H' e L', só pode ser manipulado indiretamente, como veremos mais adiante.

Como já foi assinalado, A é o **acumulador**. B, C, D, E, H e L podem ser usados como registradores de oito BITs, ou aos pares, BC, DE e HL, como registradores de 16 BITs. Neste caso, B, D e H contêm o BYTE mais significativo, enquanto C, E e L, o menos significativo. O registrador F guarda os diversos "FLAGS" (sinalizadores). Estes indicam as condições internas do microprocessador e podem ser usados, como veremos, nas instruções de Salto, Chamada e Retorno. Há ainda quatro registradores de 16 BITs (isto é, só podem ser usadas aos pares): IX, IY, SP e PC. Os dois primeiros, denominados registradores indexados, serão descritos no próximo capítulo; os outros dois, SP e PC, como já foi assinalado, referem-se, respectivamente, ao ponteiro da pilha de dados e ao contador de programa.

Finalmente, há dois outros registradores de 8 BITs, o registrador de interrupção I e o de **refresh** R, que só serão estudados em obra posterior, de caráter não introdutório.

CAPÍTULO IV:
GRUPO DE TRANSFERÊNCIA DE DADOS

Nosso objetivo, neste capítulo e nos quatro subseqüentes, é o de examinar as diversas instruções que podem ser implementadas no Z80A. A melhor forma de estudar tais instruções consiste em manipulá-las, através de exercícios. Portanto, para cada tipo de instrução, existe uma rotina em linguagem de máquina. Ela deve ser implementada com o auxílio do programa de carregamento dado no capítulo II. (Utilizaremos aqui a terceira sugestão; ou seja, modificaremos o topo do RAM, e o primeiro endereço livre para linguagem de máquina será 30000. Caso você não tenha 16K de memória, reserve o espaço acima de 18000. Se você tiver 48K, então o endereço 60000 será apropriado. **Lembre-se de fazer as modificações necessárias**).

A vantagem deste método está no fato de ele procurar, tanto quanto possível, aproximar a linguagem de máquina da programação em BASIC. As rotinas em linguagem de máquina são pequenas a fim de reduzir a possibilidade de ocorrência de um **crash**. Elas serão, na medida do possível, progressivamente ampliadas conforme se avança os capítulos. Desta forma, aprende-se a dividir um programa em pequenas unidades e a combiná-las de modo a produzir, no final, o resultado desejado.

Antes de dar a **string** para o programa de carregamento, a rotina é apresentada sob a forma de endereço, código objeto, código fonte.

Aprenda a ler a rotina, pois, a partir do próximo capítulo, não será mais dada a **string**. Cada rotina é explicada em detalhe, embora ela não ilustre todos os exemplos possíveis. Essa tarefa é deixada para você.

EXERCÍCIOS ILUSTRATIVOS

I - Instrução NOP

Esta instrução, como já foi dito, significa **no operation** ou seja, não realize nenhuma operação. Escreva a seguinte rotina:

```
30000    00        NOP
30001    C9        RET
```

Para isto, entre com a **string** "00C9P". Digite PRINT USR 30000. Qual o resultado?

Você obtém 30000, isto é, 117* 256+48. Em outras palavras, USR retorna com o valor do par de registradores BC. O único efeito causado por NOP foi o de produzir um certo atraso no retorno — o tempo gasto para implementá-la.

II - Instrução LD registrador, dado.

Todo registrador do conjunto principal pode ser carregado diretamente com dados. Isto é ilustrado pela seguinte rotina:

```
30000    0600      LD B,0
30002    0E0A      LD C,10
30004    C9        RET
```

Entre com a seguinte **string**: "06000E0AC9P". Digite agora PRINT USR 30000. Qual o resultado?

Obtém-se 10, ou seja, o valor colocado no registrador C, uma vez que B foi carregado com zero. Para poder variar os valores de B e de C, entre com o seguinte programa:

```
10 INPUT X
20 INPUT Y
30 POKE 30001,X
40 POKE 30003,Y
50 PRINT "VALOR DE B= ";PEEK 30001
60 PRINT "VALOR DE C= ";PEEK 30003
70 PRINT USR 30000
80 GOTO 10
```

Através dele, obtém-se, na terceira linha, o resultado do registrador B multiplicado por 256 somado ao registrador C.

O que ocorrè quando você entra com um número negativo?

Ele é somado a 256 e, em seguida, processado. Experimente, por

exemplo, -255 e -1 para X e Y, respectivamente. Qual o resultado?

511, isto é, [(-255+256) * 256 + (256+(-1)].

A tabela abaixo fornece o código em HEX dessas instruções.

```
LD A,N    3EN
LD B,N    06N
LD C,N    0EN
LD D,N    16N
LD E,N    1EN
LD H,N    26N
LD L,N    2EN
```

(N é um número em HEX no intervalo entre 00 e FF)

Pode-se entender esta instrução como igualmente a instrução LET em BASIC ou seja, LD A, N é semelhante a LET A=N onde N varia no intervalo entre zero e duzentos cinqüenta e cinco.

III - Instrução LD par de registradores, dado

O carregamento aqui, ainda que imediato, utiliza-se de registradores de 16 BITs, ou seja, BC, DE, HL, SP, IX e IY. Para ilustrá-lo, entre com a seguinte rotina:

```
30000    012000    LD BC,32
30003    C9        RET
```

Digite a **string** "012000C9P" e entre com PRINT USR 30000. Qual o resultado?

Exatamente aquele que está presente no par de registradores BC, ou seja, 32.

Elabore, como exercício, um programa em BASIC que lhe permita variar o valor colocado no par BC.

A tabela abaixo fornece o código em HEX dessas instruções.

```
LD BC, NN     01 NN
LD DE, NN     11 NN
LD HL, NN     21 NN
LD SP, NN     31 NN
LD IX, NN  DD 21 NN
LD IY, NN  FD 21 NN
```

(NN é um par de números em HEX variando no intervalo 0000-FFFF)

Podemos entender essa instrução como sendo semelhante à instrução LET em BASIC, ou seja. LD BC, NN é semelhante a LET BC = NN, onde NN varia entre 0 e 65535.

IV — **Instrução LD dado de um registrador (fonte) para outro registrador (destino)**

Pode-se transferir o valor contido num registrador para outro registrador. Apenas este último é alterado pela instrução. A rotina abaixo ilustra isto, ao passar o conteúdo de A para o registrador C.

```
30000    3E10    LD A,16
30002    0600    LD B,0
30004    4F      LD C,A
30005    C9      RET
```

§ **String:** "3E1006004FC9P".

O resultado de PRINT USR 30000 é precisamente o valor contido no acumulador A. Escreva um programa em BASIC que permita variar o valor de A.

A tabela abaixo fornece o código em HEX dessas instruções.

LD A,A : 7F	LD B,B : 40	LD C,C : 49	LD D,D : 52
LD A,B : 78	LD B,A : 47	LD C,A : 4F	LD D,A : 57
LD A,C : 79	LD B,C : 41	LD C,B : 48	LD D,B : 50
LD A,D : 7A	LD B,D : 42	LD C,D : 4A	LD D,C : 51
LD A,E : 7B	LD B,E : 43	LD C,E : 4B	LD D,E : 53
LD A,H : 7C	LD B,H : 44	LD C,H : 4C	LD D,H : 54
LD A,L : 7D	LD B,L : 45	LD C,L : 4D	LD D,L : 55
LD E,E : 5B	LD H,H : 64	LD L,L : 6D	
LD E,A : 5F	LD H,A : 67	LD L,A : 6F	
LD E,B : 58	LD H,B : 60	LD L,B : 68	
LD E,C : 59	LD H,C : 61	LD L,C : 69	
LD E,D : 5A	LD H,D : 62	LD L,D : 6A	
LD E,H : 5C	LD H,E : 63	LD L,E : 6B	
LD E,L : 5D	LD H,L : 65	LD L,H : 6C	

Todas essas instruções podem ser consideradas semelhantes a LET X=Y em BASIC, onde X e Y são variáveis cujo valor está sempre no intervalo entre 0 e 255. Quando um acumulador é carregado com o seu próprio conteúdo, temos uma instrução equivalente a NOP.

V - **Instrução LD dado, a partir de uma localidade na memória, no par de registradores HL** (endereçamento absoluto)

Diferentemente do que estudamos até agora, esta instrução carrega, o par de registradores HL, com o conteúdo do endereço

indicado por NN. Para diferenciá-la da instrução LD HL, NN, colocamos o par NN entre parênteses. Experimente a seguinte rotina:

```
30000    2A0C40    LD HL,(16396)
30003    44        LD B,H
30004    4D        LD C,L
30005    C9        RET
```

String - "2A0C40444DC9P"

PRINT USR 30000 indica o endereço do primeiro BYTE no arquivo da imagem. Para mostrar isto, digite este programa:

```
10 LET A=USR 30000+1
20 POKE A,1
30 POKE A+31,1
40 POKE A+21*33,1
50 POKE A+21*33+31,1
```

Ele irá colocar o caractere número 1 nos quatro cantos da tela.

O código desta instrução em HEX é 2A(NN), onde (NN) designa um endereço. Ela pode ser entendida como semelhante ao seguinte conjunto de instruções em BASIC:

```
LET H=PEEK N+1
LET L=PEEK N
```

VI - **Instrução LD dado, a partir de uma localidade na memória, num par de registradores** (endereçamento absoluto).

A única diferença entre esta instrução e a anterior é o seu tamanho. Todas elas têm quatro BYTEs. A rotina abaixo exemplifica este tipo de instrução.

```
30000    ED4B3440    LD BC,(16436)
30004    0600        LD B,0
30006    C9          RET
```

String - "ED4B34400600C9P" Colocamos no par de registradores BC o conteúdo do endereço indicado por (NN), no caso presente, a variável do sistema FRAMES. A instrução LD B,0 permite que PRINT USR 30000 gere um número aleatório entre 0 e 255.

A tabela abaixo fornece o código em HEX dessas instruções.

LD BC, (NN)	ED4B NN
LD DE, (NN)	ED5B NN
LD HL, (NN)	ED6B NN
LD SP, (NN)	ED7B NN

(NN é um endereço entre 0000 e FFFF)

Observações:

a) é sempre preferível utilizar LD HL, (NN) com três BYTEs do que com quatro;

b) todas essas instruções são semelhantes em BASIC, por exemplo, ao seguinte conjunto de instruções:

```
LET B = PEEK N+1
LET C = PEEK N
```

VII - **Instrução LD dado, a partir de uma localidade na memória, num registrador** (endereçamento indireto).

Atente para a rotina abaixo:

```
30000    210040    LD HL,16384
30003    4E        LD C,(HL)
30004    0600      LD B,0
30006    C9        RET
```

String:"2100404E0600C9P" Veja se você já consegue antecipar o resultado. Demos a HL o endereço 16384, o da variável do sistema ERR.NR. A seguir, carregamos o registrador C com o conteúdo desse endereço. Desta forma, PRINT USR 30000 dará 255.

A tabela abaixo fornece o código em HEX dessas instruções.

LD A, (NN) 3A(NN)LD A, (HL) 7E LD A, (BC)0A LD A, (DE) 1A

```
LD B, (HL)   46
LD C, (HL)   4E
LD D, (HL)   56
LD E, (HL)   5E
LD H, (HL)   66
LD L, (HL)   6E
```

Observações:

a) a instrução 7E é de uso freqüente nas rotinas em linguagem de máquina; contudo, seu código é utilizado pelo programa monitor para caracterizar que um número está na forma binária de ponto flutuante. A conseqüência disto é não podermos listar nem esta instrução nem as cinco seguintes — elas ficam invisíveis na listagem, embora estejam presentes no programa;

b) essas instruções são semelhantes a instruções do tipo

```
LET C = PEEK HL
```

c) note que o acumulador pode ser carregado, não apenas de (HL), mas de (BC), (DE) ou mesmo de (NN), diferentemente dos demais registradores de oito BITs.

VIII - **Instrução LD dado, a partir de uma localidade na memória, num registrador** (endereçamento indexado)

Além dos endereçamentos absoluto e do indireto, pode-se utilizar o indexado, através dos registradores indexados IX e IY. Eles são idênticos quanto às funções; portanto, o que for descrito em relação a um é válido em relação a outro.

A grande vantagem na sua utilização reside na possibilidade de somar constantes aos endereços que eles indicam. Contudo, no caso particular dos computadores que estamos estudando, não se pode utilizar o registrador IX no modo "SLOW". Em compensação, IY aponta para o endereço 16384, ou seja, o início das variáveis do sistema.

Para demonstrar o seu uso, entre com a seguinte rotina:

```
30000    FD7E00    LD A,(IY)
30003    4F        LD C,A
30004    0600      LD B,0
30006    C9        RET
```

String: "FD7E004F0600C9P"

A instrução PRINT USR 30000 fornece o valor da variável do sistema ERR NR, ou seja, 255, como já foi visto.

A tabela abaixo fornece o código dessas instruções em HEX.

```
LD A, (IY+dis)    FD7E N
LD B,(IY+dis)     FD46 N
LD C,(IY+dis)     FD4E N
LD D,(IY+dis)     FD56 N
LD E,(IY+dis)     FD5E N
LD H,(IY+dis)     FD66 N
LD L,(IY+dis)     FD6E N
```

(N é um número em HEX no intervalo entre 00 e FF)

Observações:

a) as instruções que usam IX têm o mesmo código que IY, com exceção do primeiro BYTE, que, em vez de FD, é DD;

b) o deslocamento pode-se dar no intervalo compreendido entre -128 e +127, conforme veremos no capítulo VI, ao estudarmos a convenção complemento de dois.

c) estas instruções são semelhantes a instruções do tipo
LET registrador = PEEK (IY+dis)

IX — **Instrução LD dado, a partir do acumulador, numa localidade de memória** (endereçamento absoluto)

Ao contrário das instruções anteriores, as quais carregavam o

acumulador a partir de uma localidade na memória. trata-se aqui de colocar num endereço o valor presente no acumulador.

Isto pode ser ilustrado pela seguinte rotina:

```
30000    3E10      LD A,16
30002    323A75    LD (30010),A
30005    C9        RET
```

String: "3E10323A75C9P"

Para testar estas instruções, entre com o seguinte programa:

```
10 INPUT X
20 POKE 30001,X
30 RAND USR 30000
40 PRINT PEEK 30010
50 GOTO 10
```

Ele possibilita-lhe colocar diversos valores no acumulador e verificar a rotina através da instrução presente na linha 40. Este tipo de instrução, cujo código em HEX é 32 (NN) pode ser interpretado como semelhante ao seguinte conjunto de instruções em BASIC:

```
LET A = X
POKE NN, A
```

(NN especifica um endereço no intervalo entre 16384 a 32768, caso se possua uma extensão de 16K).

X — **Instrução LD, a partir do par do registradores, HL, numa localidade na memória** (endereçamento absoluto).

Análoga ao tipo de instrução anterior, com uma única diferença: o conteúdo é transferido a partir do par de registradores HL.

```
30000    210440    LD HL,16388
30003    223A75    LD (30010),HL
30006    C9        RET
```

String: "210440223A75C9P"

Para constatar o que a rotina acima realiza, entre com o seguinte programa:

```
10 RAND USR 30000
20 PRINT PEEK 30010+256*PEEK 30011
```

Qual o resultado obtido? Encontra-se 16388, ou seja, o valor

contido no par de registradores HL. Isto indica que esta instrução — cujo código em HEX é 22 (NN) onde (NN) indica um endereço no intervalo entre 16384 e 32768 — é semelhante ao seguinte conjunto de instruções em BASIC:

```
LET H = A
LET L = B
POKE (NN),    B
POKE (NN+1),A
```

XI — **Instrução LD, a partir de registradores de 16 BITs, numa localidade na memória** (endereçamento absoluto).

O mesmo tipo de instrução que a anterior — a única diferença reside no comprimento das instruções: todas elas possuem 4 BYTEs.

A tabela abaixo fornece o código em HEX dessas instruções.

```
LD (NN), BC    ED43( NN)
LD (NN), DE    ED53 (NN)
LD (NN), HL    ED63 (NN)
LD (NN), SP    ED73 (NN)
```

(NN é um endereço no intervalo compreendido entre 16384 e 32768)

XII — **Instrução LD dado, a partir de um registrador, numa localidade na memória** (endereçamento indireto)

É necessário que o local já tenha sido especificado, para que se possa utilizar este tipo de instrução.

```
30000    213A75    LD HL,30010
30003    3E20      LD A,32
30005    77        LD (HL),A
30006    C9        RET
```

String : "213A753E2077C9P" Para testar esta rotina, entre com o programa abaixo.

```
10 INPUT X
20 POKE 30004,X
30 RAND USR 30000
40 PRINT PEEK 30010
50 GOTO 10
```

Este permite variar o valor colocado no acumulador e testar se ele é realmente transferido para o endereço indicado por HL.

Observações:

a) o acumulador, diferentemente dos outros registradores de 8 BITs,

pode transferir dados, não só para o endereço indicado por HL, como para os que são indicados por BC e DE;

b) o endereço indicado por HL também pode ser carregado diretamente, sem o uso de quaisquer registradores, através da instrução LD (HL), N (código em HEX: 36 N), onde N é um número em HEX no intervalo 00 a FF;

c) as instruções deste tipo podem ser vistas como semelhantes ao seguinte conjunto de instruções em BASIC:

LET registrador = X
POKE endereço indicado por HL, X

A tabela abaixo fornece o código em HEX destas instruções.

LD (HL), A	77
LD (HL), B	70
LD (HL), C	71
LD (HL), D	72
LD (HL), E	73
LD (HL), H	74
LD (HL), L	75

XIII — **Instrução LD dado, a partir de um registrador indexado, numa localidade na memória** (endereçamento indexado)

Antes de realizar o exercício, releia com atenção o oitavo exercício deste capítulo.

```
30000    3E05      LD A,5
30002    FD7700    LD (IY),A
30005    C9        RET
```

String: "3E05FD7700C9P"

Não é necessário especificar IY porque, como já foi estudado, ele automaticamente aponta para o início das variáveis do sistema. Para estudar as diversas denotações de erro, entre com o seguinte programa e rode-o várias vezes, variando o valor de X.

```
10 INPUT X
20 POKE 30001,X
30 RAND USR 30000
```

Antes de dar valor a X, leia as observações que o manual do seu computador tece a respeito dos valores que podem ser colocados neste endereço (16384). A propósito, qual o único valor de X que tornaria possível que uma instrução GOTO 10 colocada na linha 40 fosse obedecida?

A tabela abaixo fornece o código em HEX dessas instruções.

LD(IY+dis),A	FD77N
LD(IY+dis),B	FD70N
LD(IY+dis),C	FD71N
LD(IY+dis),D	FD72N
LD(IY+dis),E	FD73N
LD(IY+dis),H	FD74N
LD(IY+dis),L	FD75N

(N é um número em HEX no intervalo entre 00 e FF)

As mesmas observações feitas em relação ao exercício VIII são válidas aqui.

Esta instrução é semelhante em BASIC ao conjunto de instruções do tipo LET registrador = X POKE endereço indicado por (IY+dis), X

XIV - **Instruções que se referem ao "STACK"**

Como já foi visto antes (capítulo III), podemos colocar ou retirar coisas do "STACK", isto é, podemos fazer um PUSH (acrescente) par de registradores ou um POP (remonte) par de registradores.

Todo o cuidado em manipular o "STACK" está em não alterar o valor do seu ponteiro após o uso da rotina.

```
30000    112000    LD DE,32
30003    D5        PUSH DE
30004    C1        POP BC
30005    0600      LD B,0
30007    C9        RET
```

String: "112000D5C10 600C9P"

Veja se consegue antecipar o resultado. Ele é o valor colocado no registrador E. Note que o SP permanece o mesmo após o término da rotina. Escreva um programa em BASIC que permita variar o valor de E e retorná-lo através do registrador C.

A tabela abaixo fornece o código em HEX destas instruções.

PUSH BC	C5	POP BC	C1
PUSH DE	D5	POP DE	D1
PUSH HL	E5	POP HL	E1
PUSH AF	F5	POP AF	F1
PUSH IX	DDE5	POP IX	DDE1
PUSH IY	FDE5	POP IY	FDE1

Observações:

a) pode-se utilizar a instrução PUSH para salvar valores presentes nos

registradores de 16 BITs, após o que eles retornam através da instrução POP;

b) o registrador de 16 BITs AF é formado pelo acumulador A e pelo registrador do estado dos "FLAGs" (sinalizadores) F (este último será estudado no capítulo VI);

c) a semelhança com instruções em BASIC encontra-se no final do capítulo III.

XV - Instruções que trocam os registradores principais pelos alternativos.

Acabamos de mencionar (na primeira observação do último exercício) que as instruções PUSH e POP podem ser utilizadas para salvar os valores contidos nos registradores. Contudo, existe uma instrução bem mais simples que permite realizar esta tarefa, ou seja, EXX (código em XEX: D9). Se for necessário preservar também o conteúdo do acumulador, ou o estado presente dos "FLAGs", ou ambos, devemos acrescentar à instrução EXX a instrução EX AF, AF' (código em HEX: 08). A rotina abaixo exemplifica a forma correta de se efetuar a troca.

```
30000    2600      LD H,0
30002    2E20      LD L,32
30004    08        EX AF,AF'
30005    D9        EXX
30006    D9        EXX
30007    08        EX AF,AF'
30008    4D        LD C,L
30009    44        LD B,H
30010    C9        RET
```

String: "26002E2008D9D9084D44C9P"

PRINT USR 30000 dará como resultado o valor colocado no registrador L.

Também é possível trocar o valor contido no par DE, pelo presente no par HL, através da instrução EX DE, HL (código em HEX: EB).

XVI — Instruções que transferem blocos de dados

Antes de descrever LDIR e LDDR (ver capítulo III), vamos estudar as mesmas instruções sem repetição automática, isto é, LDI e LDD.

Estas duas instruções, cada uma com comprimento de dois BYTEs, transferem o conteúdo de uma localidade na memória, indicada por HL, para outra, indicada por DE. Se a instrução for LDI,

os registradores HL e DE são incrementados de um; se for LDD, eles são diminuídos de um. Em ambos os casos, BC atua como um "registrador de contagem" e é diminuído de um cada vez que a instrução é implementada.

LDIR e LDDR fazem a mesma coisa, de forma automática, até que BC tenha o valor zero.

```
30000    014A01    LD BC,330
30003    2A0C40    LD HL,(16396)
30006    09        ADD HL,BC
30007    EB        EX DE,HL
30008    2A0C40    LD HL,(16396)
30011    016B01    LD BC,363
30014    09        ADD HL,BC
30015    EDB0      LDIR
30017    0E015     LD C,21
30019    C9        RET
```

String: "014A012A0C4009EB2A0C40 016B0109ED-B00E15C9P"

Esta rotina faz um "SCROLL" no sentido inverso da parte inferior da tela (as dez primeiras linhas não são alteradas). Ela pode ser utilizada a partir do seguinte programa em BASIC:

```
10 DIM C$(32)
15 PRINT  AT 9,0;"FICO IMOVEL"
20 INPUT C$
30 PRINT AT USR 30000,0;C$(TO 32)
40 GOTO 20
```

Você já deve estar capacitado a acompanhar toda a rotina. Para ajudá-lo, vamos dar os valores de HL, DE e BC, antes de a instrução LDIR ser implementada.

HL - aponta para o 363º lugar no arquivo de imagem, ou seja, para o início da 12ª linha do vídeo (note que HL é a fonte);

DE - aponta para 330º lugar no arquivo de imagem, ou seja, para o início da 11ª linha do vídeo (note que DE é o destino);

BC - aponta a quantidade de transferências a serem feitas: 363 transferências.

A tabela abaixo fornece o codigo em HEX destas instruções.

LDI	EDA0
LDD	EDA8
LDIR	EDB0
LDDR	EDB8

Este tipo de instrução (por exemplo, LDI) é semelhante ao seguinte conjunto de instruções em BASIC

```
LET DE=X
LET HL=Y
POKE X=PEEK Y
LET BC=BC -1
LET DE=DE +1
LET HL=HL +1
```

XVII — **Instruções que aumentam (INC) ou diminuem (DEC)**

Estas instruções podem ser divididas de acordo com aquilo que elas aumentam ou diminuem.

A — **Conteúdo de um registrador de oito BITs**

```
30000    0C        INC C
30001    C9        RET
```

String: "0CC9P"

A instrução PRINT USR 30000 fornece 30001 como resultado, uma vez que o registrador C sofreu um acréscimo igual a um. Neste sentido, INC registrador pode ser entendido como semelhante em BASIC à instrução: LET registrador = registrador + um.

O que ocorre se o valor de C for 255?

```
30000    0EFF      LD C,255
30002    0C        INC C
30003    0600      LD B,0
30005    C9        RET
```

String: "0EFF0C0600C9P"

A rotina acima mostra que o valor em retorno é igual a zero. Portanto, é possível limpar o registrador com a instrução INC registrador, desde que o valor anterior seja 255.

Experimente a rotina abaixo:

```
30000    0E00      LD C,0
30002    0600      LD B,0
30004    0D        DEC C
30005    C9        RET
```

String: "0E0006000DC9P"

Qual o resultado da instrução PRINT USR 30000? Ela retorna com 255, ou seja, se um registrador contém 255 (FF em HEX) após a instrução DEC registrador, é porque o seu valor anterior era zero.

A tabela abaixo fornece o código em HEX destas instruções.

INC A	3C	DEC A	3D
INC B	04	DEC B	05
INC C	0C	DEC C	0D
INC D	14	DEC D	15
INC E	1C	DEC E	1D
INC H	24	DEC H	25
INC L	2C	DEC L	2D

Esta instrução é semelhante em BASIC a:

LET registrador = registrador - um.

B - **Conteúdo de um registrador de 16 BITs**

```
30000    03        INC BC
30001    C9        RET
```

String: "03C9P"

A rotina acima retorna com o valor 30001 portanto, INC BC, acrescenta um ao endereço de retorno. Isto é feito aumentando-se o BYTE menos significativo. Se este contém 255, o acréscimo é feito da seguinte maneira: o BYTE menos significativo é zerado, e o mais significativo é adicionado de um.

```
30000    0B        DEC BC
30001    C9        RET
```

"String": "0BC9P"

O resultado em retorno é evidentemente 29999. O processo aqui é similar ao anterior: se o BYTE menos significativo é nulo, ele passa a ser 255, e o mais significativo sofre um decréscimo de um.

Escreva uma rotina onde BC tem valor zero e, em seguida, é implementada a instrução DEC BC. Qual o resultado que você obtém?

A tabela abaixo fornece o código em HEX destas instruções.

INC BC	03	DEC BC	0B
INC DE	13	DEC DE	1B
INC HL	23	DEC HL	2B
INC SP	33	DEC SP	3B
INC IX	DD23	DEC IX	DD2B
INC IY	FD23	DEC IY	FD2B

Estas instruções são semelhantes em BASIC, respectivamente, a:

```
LET registrador de 16 BITs = registrador de 16 BITs +1
LET registrador de 16 BITs = registrador de 16 BITs -1
```

C - **Conteúdo de um local na memória**

```
30000    213A75     LD HL,30010
30003    3E20       LD A,32
30005    323A75     LD (30010),A
30008    34         INC (HL)
30009    C9         RET
```

"String": 213A753E2Ø323A7534C9P"

Para variar o conteúdo de A e testar o efeito sobre o conteúdo da memória, indicado pelo par de registradores HL, entre com o seguinte programa:

```
10 INPUT X
20 POKE 30004,X
30 RANDUSR 30000
40 PRINT PEEK 30010
50 GOTO 10
```

Escreva uma rotina, similar à dada acima, que utilize a instrução DEC (HL).

A tabela abaixo fornece o código em HEX destas instruções.

INC(HL) 34	DEC (HL) 35
INC (IX+dis) DD34N	DEC (IX+dis) DD35N
INC (IY+dis) FD34N	DEC (IY+dis) FD35N

(N é um número em HEX no intervalo entre ØØ e FF)

Estas instruções são semelhantes em BASIC, respectivamente, a:

```
LET endereço = endereço + 1
LET endereço = endereço - 1
```

EXERCÍCIOS

1) Elabore uma rotina que faça **SCROLL** ao contrário da tela toda.
2) Repita os exercícios do presente capítulo variando as instruções.

CAPÍTULO V :
GRUPO DE OPERAÇÕES ARITMÉTICAS E LÓGICAS

Neste capítulo, onde estudaremos as diversas instruções lógicas e aritméticas, você entenderá por que o acumulador desempenha um papel fundamental na programação em linguagem de máquina.

INSTRUÇÕES ARITMÉTICAS

Através destas instruções, pode-se realizar operações aritméticas entre os diversos registradores e o acumulador. Elas afetam o estado dos "FLAGs", contudo, estes somente serão estudados no próximo capítulo.

Cada exercício ensina um tipo de instrução e deve ser executado com o auxílio do computador. Para entrar com as rotinas em linguagem de máquina, utilize o mesmo programa de carregamento.

EXERCÍCIOS ILUSTRATIVOS

I — Adicionar, ao conteúdo do acumulador, o conteúdo de um registrador

```
30000    0E32       LD C,50
30002    3E16       LD A,22
30004    81         ADD A,C
30005    4F         LD C,A
30006    0600       LD B,0
30008    C9         RET
```

PRINT USR 30000 retorna com 72. Experimente ultrapassar 255 com o auxílio do seguinte programa:

```
10 INPUT X
20 INPUT Y
30 POKE 30001,X
40 POKE 30003,Y
50 PRINT USR 30000
60 GOTO 10
```

Toda vez que se ultrapassa 255 há um transbordamento (um **overflow**) e o resultado é subtraído de 256.

Que instrução permite duplicar o valor contido num registrador, se ele estiver no intervalo entre 1 e 127?

A tabela abaixo fornece o código em HEX destas instruções.

ADD A,A	87
ADD A,B	80
ADD A,C	81
ADD A,D	82
ADD A,E	83
ADD A,H	84
ADD A,L	85

Estas instruções são semelhantes em BASIC a:

LET acumulador = acumulador + registrador

II - Adicionar com "CARRY" o conteúdo de um registrador, ao conteúdo do acumulador

```
30000    06FA      LD B,250
30002    3E06      LD A,6
30004    88        ADC A,B
30005    3E00      LD A,0
30007    8F        ADC A,A
30008    4F        LD C,A
30009    0600      LD B,0
30011    C9        RET
```

Na rotina acima, pode-se constatar que o valor colocado em A ultrapassou a 255; isto é, houve um transbordamento. Isto afeta o "CARRY FLAG", (sinalizador de transporte), que fica igual a 1. Afirma-se que ele foi ativado (**set**). Com o auxílio do programa anterior pode-se variar os valores carregados em B e A, e constatar, através desta rotina, quando ocorre um "CARRY". (um transporte)

A tabela abaixo fornece o código em HEX destas instruções.

ADC A,A	8F
ADC A,B	88
ADC A,C	89
ADC A,D	8A
ADC A,E	8B
ADC A,H	8C
ADC A,L	8D

Observações:

a) estas instruções são semelhantes em BASIC ao seguinte conjunto:
LET acumulador = acumulador + registrador + "CARRY"
LET "CARRY" = INT ((acumulador + registrador)/256)

b) há duas instruções que controlam diretamente o "CARRY FLAG":
— SCF (código em HEX: 37) - ativa o "CARRY FLAG", isto é, ele fica igual a um;
— CCF (código em HEX: 3F) - fornece o complemento do "CARRY FLAG", ou seja, um menos o estado presente do "CARRY FLAG".

III - Subtrair o conteúdo de um registrador, do conteúdo do acumulador.

Este e o seguinte exercícios são complementares aos dois primeiros. Portanto, utilize o programa dado naqueles exercícios para testar as rotinas que serão oferecidas.

```
30000    0E10      LD C,16
30002    3E20      LD A,32
30004    91        SUB A,C
30005    4F        LD C,A
30006    0600      LD B,0
30008    C9        RET
```

Se o resultado foi um número negativo, isto é, se ocorrer um "empréstimo", o resultado é somado a 256. A instrução SUB A torna o conteúdo do acumulador igual a zero.

A tabela abaixo fornece o código em HEX destas instruções.

SUB A	97
SUB B	90
SUB C	91
SUB D	92
SUB E	93
SUB H	94
SUB L	95

Estas instruções são equivalentes em BASIC a:

LET acumulador = acumulador - registrador

IV - Subtrair com "CARRY" o conteúdo de um registrador, do conteúdo do acumulador

```
30000    0E20        LD C,32
30002    3E06        LD A,6
30004    99          SBC A,C
30005    3E00        LD A,0
30007    8F          ADC A,A
30008    4F          LD C,A
30009    0600        LD B,0
30011    C9          RET
```

Variando os valores carregados no registrador C e no acumulador, obtém-se um, se C for maior do que A (isto é, se houver um empréstimo e o "CARRY FLAG" for ativado); e zero, se C for menor ou igual a A (ou seja, se não houver empréstimo e o "CARRY FLAG" for restaurado).

Por que a rotina acima não controlaria o "CARRY FLAG" se a instrução LD A, 0 fosse substituída por SUB A?

A tabela abaixo fornece o código em HEX destas instruções.

SBC A,A	9F
SBC A,B	98
SBC A,C	99
SBC A,D	9A
SBC A,E	9B
SBC A,H	9C
SBC A,L	9D

V — Adição e Subtração envolvendo registradores de 16 BITs

Neste tipo de instrução, o par de registradores HL toma o lugar do acumulador.

Adição sem "CARRY"

```
30000    012909      LD BC,2345
30003    21BD45      LD HL,17305
30006    09          ADD HL,BC
30007    44          LD B,H
30008    4D          LD C,L
30009    C9          RET
```

O programa abaixo permite variar os valores colocados em BC e HL.


```
10 INPUT C
15 INPUT B
20 INPUT L
25 INPUT H
30 POKE 30001,C
35 POKE 30002,B
40 POKE 30004,L
45 POKE 30005,H
50 PRINT PEEK 30001+256*PEEK 30002;
" + ";PEEK 30004+256*PEEK 30005;" = ";USR 30000
55 GOTO 10
```

O que acontece se o valor da soma ultrapassar a 65535?

A tabela abaixo fornece o código em HEX destas instruções.

ADD HL, BC	09
ADD HL, DE	19
ADD HL, HL	29
ADD HL, SP	39

Estas instruções são semelhantes em BASIC a:

LET par de registradores HL=par de registradores HL
+ par de registradores de 16 BITs

Adição com "CARRY"

```
30000    110000      LD DE,0
30003    210000      LD HL,0
30006    EDSA        ADC HL,DE
30008    3E00        LD A,0
30010    8F          ADC A,A
30011    4F          LD C,A
30012    0600        LD B,0
30014    C9          RET
```

O resultado de PRINT USR 30000 será zero, se não houver um transbordamento (isto é, o resultado não for superior a 65535); caso contrário, será um.

Subtração com "CARRY"

Não existe subtração sem "CARRY"

```
30000    110000    LD DE,0
30003    210000    LD HL,0
30006    ED52      SBC HL,DE
30008    3E00      LD A,0
30010    8F        ADC A,A
30011    4F        LD C,A
30012    0600      LD B,0
30014    C9        RET
```

O resultado de PRINT USR 30000 será zero, se DE for menor ou igual a HL; se diferente, será um: houve um empréstimo.

A tabela abaixo fornece o código em HEX dos dois últimos tipos de instrução.

ADC HL, BC	ED4A	SBC HL, BC	ED42
ADC HL, DE	ED5A	SBC HL, DE	ED52
ADC HL, HL	ED6A	SBC HL, HL	ED62
ADC HL, SP	ED7A	SBC HL, SP	ED72

VI — **Adição e Subtração envolvendo o acumulador e uma localidade na memória.**

Pode-se realizar operações aritméticas entre o acumulador e uma localidade na memória apontada por HL.

```
30000    213A75    LD HL,30010
30003    3E00      LD A,0
30005    86        ADD A,(HL)
30006    4F        LD C,A
30007    0600      LD B,0
30009    C9        RET
```

Teste esta rotina a partir do seguinte programa:

```
10 INPUT X
20 INPUT Y
30 POKE 30004,X
40 POKE 30010,Y
50 PRINT PEEK 30004;" ";PEEK 30010;
"SOMA = "; USR 30000
60 GOTO 10
```


Também é possível realizar adição com "CARRY".

```
30000    213F75    LD HL,30015
30003    3E00      LD A,0
30005    8E        ADC A,(HL)
30006    3E00      LD A,0
30008    8F        ADC A,A
30009    4F        LD C,A
30010    0600      LD B,0
30012    C9        RET
```

Para testá-la, altere as linhas 40 e 50 do programa anterior para:

```
40 POKE 30015,Y
50 PRINT PEEK 30004;" ";PEEK 30015;
" CARRY= ";USR 30000
```

Também é possível subtrair com ou sem "CARRY" o acumulador de uma localidade na memória. Escreva, como exercício, rotinas que utilizam este tipo de instrução.

A tabela abaixo fornece o código em HEX destas instruções.

ADD A, (HL)	86
ADC A, (HL)	8E
SUB (HL)	96
SBC A, (HL)	9E

VII — **Adição e Subtração de constantes ao acumulador**

Também se pode realizar operações aritméticas entre o acumulador e constantes.

```
30000    3E00      LD A,0
30002    CE16      ADC A,22
30004    4F        LD C,A
30005    0600      LD B,0
30007    C9        RET
```

Escreva um programa em BASIC que permita variar os valores colocados no acumulador e na instrução de ADC A, N. Elabore uma rotina que utilize a instrução SBC A, N.

A tabela abaixo fornece o código em HEX destas instruções.

ADD A,N	C6 N
ADC A, N	CE N
SUB N	D6 N
SBC A, N	DE N

(N é um número em HEX no intervalo entre 00 e FF)

VIII — **Instruções de Comparação**

O valor presente no acumulador pode ser comparado com valores carregados em registradores de oito BITs, ou com valores presentes em localidades na memória, ou, finalmente, com constantes atribuídas pelo programador. O resultado desta comparação não é guardado em lugar algum, mas afeta o registrador de "FLAGs", o qual será estudado no próximo capítulo. Comparar significa subtrair algo do acumulador.

```
30000    060A      LD B,10
30002    3E14      LD A,20
30004    B8        CP B
30005    3E00      LD A,0
30007    CE00      ADC A,0
30009    4F        LD C,A
30010    0600      LD B,0
30012    C9        RET
```

A rotina acima testa o "CARRY FLAG". Ela retorna 1, quando o valor em B é maior do que o valor contido em A, e zero, quando é menor ou igual ao presente em A. Escreva um programa que permita testar esta rotina para diferentes valores de A e de B.

A tabela abaixo fornece o codigo em HEX destas instruções.

CP A	BF	CP (HL) BE
CP B	B8	CP N FE N
CP C	B9	CP(IX+dis) DDBE N
CP D	BA	CP (IY+dis) FDBE N
CP E	BB	
CP H	BC	
CP L	BD	

INSTRUÇÕES LÓGICAS

As instruções lógicas envolvem o acumulador e valores presentes nos registradores de oito BITSs, ou em localidades na memória, ou constantes atribu'das pelo programador. Elas estão baseadas na lógica de BOOLE. Antes de examiná-las, vamos estudar os três tipos de operações lógicas presentes no uso destas instruções. As operações são:

a) **operação AND**
b) **operação OR**
c) **operação XOR**

OPERAÇÃO AND

Suponhamos, para simplificar, que as palavras do nosso processador têm comprimento igual a 1 BIT, isto é, variam no intervalo de 0 a 1. Se aplicarmos a operação AND a duas palavras quaisquer, teremos o seguinte resultado:

0 AND 0 = 0
0 AND 1 = 0
1 AND 0 = 0
1 AND 1 = 1

Em outras palavras, só obteremos 1 quando ambas as palavras estiverem no estado um. Caso contrário, o resultado será zero.

OPERAÇÃO OR

Partindo das mesmas premissas citadas no caso anterior, teremos:

0 OR 0 = 0
0 OR 1 = 1
1 OR 0 = 1
1 OR 1 = 1

Ou seja, o resultado será sempre um quando pelo menos um dos termos estiver no estado um; caso contrário, será zero.

OPERAÇÃO EXCLUSIVE OR - XOR

Partindo ainda das mesmas premissas, teremos:

0 XOR 0 = 0
0 XOR 1 = 1
1 XOR 0 = 1
1 XOR 1 = 0

Portanto, o resultado será igual a zero todas as vezes que ambos os termos estiverem no mesmo estado; caso contrário, será um.

Podemos passar agora ao estudo das diversas instruções que utilizam essas operações lógicas.

1 - **Instruções que utilizam a operação booleana AND**

```
30000    3E00      LD A,0
30002    0600      LD B,0
30004    A0        AND B
30005    4F        LD C,A
30006    0600      LD B,0
30008    C9        RET
```

Note que a instrução AND B afeta apenas o conteúdo do acumulador; o valor presente em B permanece o mesmo. Para tornar bastante claro o que esta instrução realiza, entre com o seguinte programa:

```
10 INPUT X
20 INPUT Y
30 POKE 30001,X
40 POKE 30003,Y
50 LET A=X
55 GOSUB 100
60 LET A=Y
65 GOSUB 100
70 LET A=USR 30000
75 GOSUB 100
80 PRINT
90 GOTO 10
100 PRINT A;TAB 10; " : "
105 FOR N=7 TO 1 STEP -1
110 LET D=2**N
115 PRINT CHR$(28 + INT(A/D));
120 LET A=A- INT(A/D)*D
125 NEXT N
130 PRINT A
135 RETURN
```

Ele permite controlar os valores presentes em A e B. Além disto, imprime estes valores em BINÁRIO, assim como o resultado da instrução AND B (em decimal e em BINÁRIO).

Observações:

a) a instrução AND A não altera o conteúdo do acumulador, mas reajusta o "CARRY FLAG";

b) as instruções AND 0F (15 em decimal) e AND F0 (240 em decimal) servem, respectivamente, para zerar os quatro BITs mais significativos e os quatro BITs menos significativos presentes no acumulador, e para manter inalterada a parte restante. Por exemplo:

123 = BIN 01111011	123 = BIN 01111011
15 = BIN 00001111	240 = BIN 1110000
AND 0F = BIN 00001011	AND F0 = BIN 01110000

c) a instrução AND é utilizada para zerar os BITs escolhidos pelo programador.

A tabela abaixo fornece o código em HEX destas instruções.

AND A A7	AND N E6 N	AND (HL) A6
AND B A0		AND (IX+dis) DDA6N
AND C A1		AND (IY+dis) FDA6N
AND D A2		
AND E A3		
AND H A4		
AND L A5		

II — Instruções que utilizam a operação booleana OR

```
30000   3E00    LD A,0
30002   0600    LD B,0
30004   B0      OR B
30005   4F      LD C,A
30006   0600    LD B,0
30008   C9      RET
```

Teste esta rotina a partir do programa anterior.

Observações:

a) a instrução OR A não altera o conteúdo do acumulador, mas restaura o "CARRY FLAG";

b) a instrução OR 80 (128 em decimal) irá sempre ativar o BIT mais significativo presente no acumulador, isto é, BIT 7 = 1;

c) a instrução OR é utilizada para ativar os BITs escolhidos pelo programador.

A tabela abaixo fornece o código em HEX destas instruções.

OR A B7	OR N F6 N	OR(HL)	B6
OR B B0		OR(IX+dis)	DDB6 N
OR C B1		OR(IY+dis)	FDB6 N
OR D B2			
OR E B3			
OR H B4			
OR L B5			

III - Instruções que utilizam a instrução booleana XOR

```
30000    3E00      LD A,0
30002    0600      LD B,0
30004    AB        XOR B
30005    4F        LD C,A
30006    0600      LD B,0
30008    C9        RET
```

Teste esta rotina do mesmo modo que a anterior.

Observações:

a) a instrução XOR A limpa o acumulador e restaura o "CARRY FLAG";
b) A instrução XOR 0 deixa o conteúdo do acumulador inalterado, mas restaura o "CARRY FLAG";
c) esta instrução é utilizada, como veremos mais tarde, para testar o conteúdo de um BIT.

A tabela abaixo fornece o código em HEX destas instruções.

XOR A	AF	XOR N EE N	XOR (HL)	AE
XOR B	A8		XOR(IX+dis)	DDAEN
XOR C	A9		XOR(IY+dis)	FDAEN
XOR D	AA			
XOR E	AB			
XOR H	AC			
XOR L	AD			

(onde N é um número em HEX no intervalo entre 00 e FF)

Exercícios:

1. Escreva uma rotina que imprima na tele o alfabeto de A até Z.
2. Escreva uma rotina que efetue a soma de N números, e depois teste se ela ultrapassa ou não 65535.
3. Escreva uma rotina que limpe uma linha na tela escolhida por você.

CAPÍTULO VI:
GRUPO DE SALTO, CHAMADA E RETORNO

Neste capítulo, antes de examinarmos o novo grupo de instruções, vamos estudar as convenções complemento de dois e o registrador de "FLAGs".

COMPLEMENTO DE DOIS

Podemos utilizar os binários compreendidos entre 0000 e 1111 para representar os números decimais entre 7 e -8.

A tabela abaixo já foi estudada.

```
7 - 0111
6 - 0110
5 - 0101
4 - 0100
3 - 0011
2 - 0010
1 - 0001
0 - 0000
```

Para determinar o complemento de dois de cada um desses números achamos o complemento do número binário (no caso presente, estamos trabalhando com quatro BITs) e somamos a ele 0001.

Por exemplo, queremos determinar o complemento de dois de um com quatro BITs. Para isto, precisamos executar os seguintes passos:

a) tomamos a representação positiva de 1, isto é, 0001;
b) achamos o complemento de 0001, ou seja, 1110 (para encontrar o complemento basta substituir zero por um, e vice-versa);
c) somamos tal complemento ao resultado 0001, o que resulta 1111.

Portanto, 1111 é complemento de dois com quatro BITs de 0001.

Para somar, é suficiente atentar para o quadro abaixo:

0 + 0 = 0
1 + 0 = 1
0 + 1 = 1
1 + 1 = 0 e leva 1 para a coluna seguinte.

Experimente encontrar o complemento de dois com quatro BITs de sete.

Passos:

a) 7 = 0111;
b) complemento de 7 = 1000;
c) somado com 0001, resulta 1001.

Logo, -7 representado com 4 BITs é igual a 1001.

A tabela abaixo fornece os números entre -1 e -8.

```
-1 : 1111
-2 : 1110
-3 : 1101
-4 : 1100
-5 : 1011
-6 : 1010
-7 : 1001
-8 : 1000
```

Constate que os números positivos têm o BIT mais significativo igual a zero, enquanto os negativos, igual a um.

Contudo, quando se procura obter o complemento de dois de 1000 obtém-se 1000 — ocorreu um transbordamento (um **overflow**). Ora, isto não poderia ocorrer: o complemento de dois de um número negativo deve ser um número positivo. O que aconteceu?

Isto ocorreu porque só podemos representar com quatro BITs números no intervalo entre -8 e 7. Logo, 8 não pode ser representado na convenção complemento de dois com apenas quatro BITs; são necessários mais BITs. Com oito BITs, ampliamos o intervalo em que podemos representar números através da convenção complemento de dois.

Não é difícil entender por que, com oito BITS, o maior número positivo é 127. Como ele é positivo, o seu BIT mais significativo é igual a zero. Portanto, o maior número com oito BITs em representação binária é 01111111: (127 em decimal). O complemento de 127 é 10000000 que, somado com 00000001, resulta 10000001 (-127 em decimal). O maior número negativo é 10000000, isto é, -128 em decimal. Logo, com oito BITs podemos representar números entre -128 a 127, ou seja, um total de 256 representações.

Com 16 BITs podemos utilizar a convenção de dois para representar números dentro de qual intervalo?

As considerações acima devem ter evidenciado a necessidade de se especificar o número de BITs, quando se trabalhar com a convenção complemento de dois.

A rotina abaixo encontra o complemento de dois (com oito BITs).

```
30000   3E00    LD A,0
30002   2F      CPL
30003   3C      INC A
30004   4F      LD C,A
30005   0600    LD B,0
30007   C9      RET
```

Entre com o seguinte programa, visando obter os números negativos entre -1 e -128.

```
 10 FOR D=1 TO 128
 20 POKE 30001,D
 30 LET N=USR 30000
 40 PRINT -256+N;" = ";
 50 GOSUB 100
 60 NEXT D
100 FOR S=7 TO 1 STEP -1
105 LET X=2**S
110 PRINT CHR$(28+INT(N/X));
115 LET N=N- INT(N/X)*X
120 NEXT S
125 PRINT N
130 RETURN
```

Na rotina acima pode-se substituir as instruções CPL e INC A por NEG (código em HEX-ED44). O resultado será idêntico. Experimente. No Apêndice 5, encontra-se em HEX a tabela produzida pelo programa acima.

REGISTRADOR DE "FLAGS"

O registrador de "FLAGs" fornece informações relevantes sobre as condições internas do microprocessador. Ele possui oito BITs, embora sejam usados apenas seis: os BITs 5 e 3 têm o conteúdo sempre igual a zero.

Vamos descrever os diversos "FLAGs" (sinalizadores), partindo do BIT 7.

BIT 7 — Controla o estado do sinal. Se o número for positivo, então BIT 7 será igual a zero; caso contrário, ele será um. Portanto, a "FLAG" de sinal opera baseada na convenção complemento de dois.

BIT 6 — Controla o estado de zero. Se o número presente for zero, então BIT 6 será igual a um. Se o número presente for diferente de zero, então BIT 6 será igual a zero. Logo, o "ZERO FLAG" atua de forma inversa à que seria esperada.

BIT 5 — Não é usado.

BIT 4 — Controla se ocorre ou não um transbordamento no BIT 3. O "HALF CARRY FLAG" é usado pela CPU, e o seu estado não pode ser testado por nenhuma instrução à disposição do programador.

BIT 3 — Não é usado.

BIT 2 — Controla duas coisas distintas, dependendo da natureza da operação efetuada. Se ela for lógica, o controle será exercido sobre o estado de paridade — se ele é ímpar ou par — isto é, se o número de zeros (ou de uns) é ímpar ou par. Se a operação for aritmética, ela controlará se houve ou não um transbordamento, baseada na convenção complemento de dois. A "Parity/Overflow Flag" será zero quando a paridade for ímpar ou quando não ocorrer um transbordamento; caso contrário, ela será um.

BIT 1 — Controla a ocorrência de operações de subtração. É usada pela CPU e, do mesmo modo que o BIT 4, não é acessível ao programador. Quando ocorrer uma subtração BIT 1 será igual a 1; caso contrário BIT 1 será igual a zero.

BIT 0 — Controla se ocorreu ou não um "CARRY". Foi examinado no capítulo anterior.

I — Operações de Salto

São instruções semelhantes, dentro de certos limites, ao GOTO do BASIC. Contudo, a forma de endereçamento é diferente, podendo ser de dois tipos:

a) absoluto;
b) relativo.

Em ambos os tipos podemos fixar condições para alterar o PC (contador de programa). Esta é uma das utilidades do registrador de "FLAGs" — permitite controlar em que condições o PC deve ser modificado.

SALTOS ABSOLUTOS

a) **Sem condições** — neste caso, toda vez que a rotina atingir esta instrução, o PC será alterado para o endereço indicado.

```
30000    0600      LD B,0
30002    0E05      LD C,5
30004    C33A75    JP 30010
30007    0E10      LD C,16
30009    C9        RET
30010    C9        RET
```

PRINT USR 3 **0000** resultará em 5, e não em 16, devido à instrução de salto.
(código em HEX: C3NN, onde NN é um endereço no intervalo **0000** a FFFF)

b) **Condicionados** - o salto só será efetuado se a condição estabelecida for preenchida; caso contrário, a instrução será ignorada.

A tabela abaixo fornece o código em HEX destas instruções.

JP NZ, NN: C2NN - salta, se o último resultado não for nulo.
JP Z, NN: CANN - salta, se o último resultado for nulo.
JP NC, NN: D2NN - salta, se no último resultado não ocorrer um "CARRY".
JP C, NN: DANN - salta, se no último resultado ocorrer um "CARRY".
JP PO, NN: E2NN - salta, se no último resultado não ocorrer um transbordamento e a paridade for ímpar.
JP PE, NN:EANN - salta, se no último resultado ocorrer um transbordamento e a paridade for par.
JP P, NN: F2NN - salta, se o último resultado for positivo.
JP M, NN:FANN - salta, se o último resultado for negativo.

Para testar estas diversas instruções, vamos dar um modelo de rotina e um programa em BASIC para cercá-la. Procure modificá-lo de modo a testar o maior número possível de instruções.

```
30000    0E00      LD C,0
30002    3E00      LD A,0
30004    B9        CP C
30005    C23875    JP NZ,30011
30008    0600      LD B,0
30010    C9        RET
```

```
30011    0600        LD B,0
30013    0EFF        LD C,255
30015    C9          RET
```

Esta rotina resultará em 255 quando a instrução de salto for executada; caso contrário, ela retornará o valor colocado no registrador C.

```
10 INPUT X
20 INPUT Y
30 POKE 30001,X
40 POKE 30003,Y
50 PRINT X;" ";Y;" ";USR 30000
60 GOTO 10
```

No exemplo fornecido, se X for igual a Y, o valor em retorno será X. Caso contrário, será 255. Para alterar a rotina, basta modificar o endereço 30005.

SALTOS RELATIVOS

A grande vantagem em se usar saltos relativos está na possibilidade de alterar-se o endereçamento da rotina em linguagem de máquina sem a necessidade de adaptá-la.

a) **sem condições** — Toda vez que a rotina atingir este tipo de instrução, o PC será incondicionalmente alterado. O novo endereço será determinado pela magnitude do deslocamento, fixado na convenção complemento de dois. Portanto, é possível saltar-se até 129 endereços para frente ou 126 endereços para trás. Note que, quando o salto for para frente, o novo endereço será o endereço do último BYTE da instrução, somado ao deslocamento mais um. No exemplo abaixo:

último BYTE da instrução — endereço =30005
deslocamento =2 BYTEs
novo endereço=30005 + 2 + 1, isto é, o endereço 30008.

Quando o salto for para trás, o novo endereço será o endereço do último BYTE, subtraído do deslocamento. Logo, JR-1 levará a uma repetição infinita. Em outras palavras, gastamos dois deslocamentos quando queremos ir para trás; por conseguinte, o máximo de deslocamentos para trás é 126.

```
30000    0600        LD B,0
30002    0E05        LD C,C
30004    1802        JR 2
30006    0E10        LD C,16
30008    C9          RET
```


PRINT USR 30000 retornará com 5, e não com 16. Note que a instrução que utiliza salto relativo é menor do que a de salto absoluto. Ela tem um comprimento igual a dois BYTEs.

b) **condicionados** — A tabela abaixo fornece o código em HEX destas instruções.

JR NC, dis 30N JR NZ, dis 20N
JR C, dis 38 N JR Z, dis 28 N

(N é um número em HEX baseado na convenção complemento de dois)

c) **Tipo especial de salto relativo** — Também é possível usar uma instrução de salto relativo que simultaneamente irá alterar o valor presente no registrador B. Cada vez que a instrução for executada, B sofrerá um decréscimo de um. A instrução será repetida até que B seja zero.
(código em HEX— DJNZ, dis: 10 N, onde N é um número em HEX baseado na convenção complemento de dois.)

As instruções de salto relativo estão ilustradas pelas rotinas abaixo. Procure compreendê-las.

ROTINA 1: inverte os caracteres presentes na tela, desde que eles estejam no intervalo entre 0 e 63.

```
30000    2A0C40      LD HL,(16396)
30003    0C16        LD B,22
30005    23          INC HL
30006    7E          LD A,(HL)
30007    FE76        CP 118
30009    2805        JR Z,5
30011    C6B0        ADD A,128
30013    77          LD (HL),A
30014    18F5        JR -11
30016    77          LD (HL),A
30017    10F2        DJNZ -14
30019    C9          RET
```

Entre com RAND USR 30000 e a tela ficará imeditamente invertida. Escreva uma outra rotina que remova a restrição acima.

ROTINA 2: possibilita um SCROLL lateral, da esquerda para a direita.

```
30000    2A0C40      LD HL,(16396)
30003    010100      LD BC,1
30006    09          ADD HL,BC
30007    1E16        LD E,22
30009    E5          PUSH HL
30010    061F        LD B,31
30012    7E          LD A,(HL)
30013    23          INC HL
30014    56          LD D,(HL)
30015    77          LD (HL),A
30016    7A          LD A,D
30017    10FA        DJNZ -6
30019    C1          POP BC
30020    02          LD (BC),A
30021    23          INC HL
30022    23          INC HL
30023    10          DEC E
30024    20EF        JR NZ,-17
30026    C9          RET
```

Para testar a rotina 2, entre com o seguinte programa:

```
10 LET A$="LINGUAGEM DE MAQUINA"
20 PRINT A$
30 RAND USR 30000
40 GOTO 30
```

Procure escrever uma outra rotina, baseada nesta, que faça um SCROLL lateral da direita para a esquerda.

ROTINA 3: possibilita colocar em qualquer lugar da tela, com o tamanho desejado, um conjunto de caracteres. Pode ser usada para limpar a tela no local que o programador determinar.

```
30000    2A0C40      LD HL,(16396)
30003    23          INC HL
30004    1600        LD D,0
30006    3E00        LD A,0
30008    BA          CP D
30009    3006        JR NC,C
30011    012100      LD BC,33
30014    09          ADD HL,BC
```



```
30015    1814        JR 20
30017    0620        LD B,32
30019    3E00        LD A,0
30021    B8          CP B
30022    3809        JR C,9
30024    3E00        LD A,0
30026    B8          CP B
30027    3004        JR NC,4
30029    7E          LD A,(HL)
30030    C600        ADD A,0
30032    77          LD (HL),A
30033    23          INC HL
30034    10EF        DJNZ -17
30036    23          INC HL
30037    15          DEC D
30038    20DE        JR NZ,-34
30040    C9          RET
```

O programa abaixo vai ajudá-lo a compreender a rotina 3.

```
 10 PRINT AT 0,5;"COORDENADAS"
 15 PRINT AT 1,0;"LINHA INICIAL = ";
 20 INPUT X
 25 IF X>=25 THEN GOTO 20
 30 PRINT X
 35 PRINT AT 2,0;"NUMERO DE LINHAS = ";
 40 INPUT X1
 45 IF X1>(X+1) THEN GOTO 40
 50 PRINT X1
 55 PRINT AT 3,0;"COLUNA INICIAL = ";
 60 INPUT Y
 65 IF (33-Y)<1 THEN GOTO 60
 70 PRINT Y
 75 PRINT AT 4,0;"NUMERO DE COLUNAS = ";
 80 INPUT Y1
 85 IF (Y+Y1)>=33 THEN GOTO 80
 90 PRINT Y1
 95 PRINT AT 5,0;"NUMERO DO CARACTERE DESEJADO =";
100 INPUT S
```

```
105 IF S>65 AND S<128 AND S>192 THEN GOTO 100
110 POKE 30005,X
115 POKE 30007,X1
120 POKE 30020,(33-Y)
125 POKE 30025,(33-Y-Y1)
130 POKE 30031,S
135 CLS
140 RAND USR 30000
```

II - Instruções de chamada (**CALL**)

São instruções semelhantes dentro de certos limites, ao GOSUB em BASIC. Da mesma forma que a instrução de SALTO, há dois tipos de chamada:

a) incondicionada;
b) condicionada.

CHAMADA INCONDICIONADA

Toda vez que o programa em linguagem de máquina encontrar este tipo de instrução, haverá uma alteração no PC. Contudo, o seu valor anterior será guardado no "STACK". Logo, toda vez que se chamar uma sub-rotina, o SP (STACK POINT) sofrerá um decréscimo de dois. Assim que for encontrada a instrução "RETURN", o endereço será retirado do "STACK", e o programa retornará a este endereço.

Um exemplo deste tipo de instrução foi dado no capítulo inicial deste livro. Nele, chamávamos uma rotina presente no ROM, que possibilitava imprimir um caractere, cujo código era guardado no acumulador. Antes de aprendermos como é possível imprimir novas coisas na tela, vamos examinar duas outras rotinas presentes no ROM.

```
30000    CDBB02     CALL 699
30003    44         LD B,H
30004    4D         LD C,L
30005    2C         INC L
30006    28F8       JR Z,-8
30008    CDBD07     CALL 1981
30011    4E         LD C,(HL)
30012    0600       LD B,0
30014    C9         RET
```

Teste a rotina acima com o seguinte programa:

```
10 LET A$= CHR$(USR 30000)
20 PRINT A$
30 GOTO 10
```


Ele permite-lhe saber qual tecla foi acionada. A primeira rotina no ROM coloca em HL o código da tecla acionada. A segunda descodifica este valor e guarda o código do catactere em (HL).

CALL CONDICIONADO

A tabela abaixo fornece o código em HEX destas instruções.
CALL NZ,NN: C4NN — a rotina será chamada, se o último resultado for diferente de zero.
CALL Z,NN: CCNN — a rotina será chamada, se o último resultado for igual a zero.
CALL NC,NN : D4NN — a rotina será chamada, se no último resultado não ocorrer um "CARRY".
CALL C,NN : DCNN — a rotina será chamada, se no último resultado ocorrer um "CARRY".
CALL PO,NN : E4NN — a rotina será chamada, se no último resultado não ocorrer um transbordamento ou se a paridade for ímpar.
CALL PE,NN : ECNN — a rotina será chamada, se no último resultado ocorrer um transbordamento ou se a paridade for par.
CALL P,NN : F4NN — a rotina será chamada, se o último resultado for positivo.
CALL M,NN : FCNN — a rotina será chamada, se o último resultado for negativo.

Podemos testar estas diversas instruções através da seguinte rotina:

```
30000    0600       LD B,0
30002    3E00       LD A,0
30004    B8         CP B
30005    C4007D     CALL NZ,32000
30008    0600       LD B,0
30010    4F         LD C,A
30011    C9         RET
```

No endereço 32000 coloque a seguinte sub-rotina:

```
32000    3E0D       LD A,13
30002    CD0808     CALL 2056
32005    C9         RET
```

Escreva um programa em BASIC que permita variar os valores colocados em A e B, assim como variar o tipo de condição imposta para chamar a rotina presente em 32000. Se esta for chamada, PRINT USR 30000 imprimirá $, seguido do valor presente em A; caso contrário, apenas este último retornará.

III — Instruções de Retorno

Além do retorno incondicional RET (código em HEX: C9), já utilizado inúmeras vezes, há retornos condicionados.

A tabela abaixo fornece o código em HEX destas instruções.

RET	NZ	C0
RET	Z	C8
RET	NC	D0
RET	C	D8
RET	PO	E0
RET	PE	E8
RET	P	F0
RET	M	F8

A rotina abaixo possibilita colocar na tela uma **string** com o tamanho desejado.

```
30000    E1          POP HL
30001    7E          LD A,(HL)
30002    23          INC HL
30003    E5          PUSH HL
30004    FE43        CP 67
30006    C8          RET Z
30007    CD0808      CALL 2056
30010    18F4        JR -12
30012    CD3075      CALL 30000
```

Observações:

a) a rotina chamada no endereço 30000 contém 12 BYTES — vai desde a instrução POP HL até a instrução JR F4;

b) esta rotina está no endereço 30012; portanto, para chamá-la no BASIC, digite RAND USR 30012;

c) a instrução POP HL aponta para o endereço logo após a instrução CALL 30000; logo, é exatamente a partir daí, no endereço 30015, que se iniciam os dados;

d) não é necessário salvar os parâmetros antes de chamar a rotina no ROM, porque as instruções POP HL e PUSH HL fazem exatamente isto; caso elas não estivessem presentes, seria necessário trocar o conjunto principal pelo alternativo, antes de chamar a rotina presente no endereço (0808 em HEX) 2056.

Coloque no endereço 30015 a seguinte **string**:

```
STRING = "808080B1AEB3ACBAA6ACAAB2B0A9AA
80B2A6B6AEB3A6808080043C9"
```

O que se obtém quando se digita RAND USR 30012?

IV - Sub-rotinas de "RESTART"

Essas rotinas estão colocadas em lugar fixo no ROM: nos endereços 00, 08, 10, 18, 20, 28, 30 e 38 (em HEX). O seu conteúdo depende do fabricante. No caso presente, elas fazem o seguinte:

RST 00 — é ativada quando se liga o computador; calcula qual o espaço disponível no RAM e confere valores às variáveis do sistema; a digitação de RAND USR 0 equivale a desligar e religar o aparelho.

RST 08 — rotina de erro; fornece o código do erro cometido.

RST 10 — coloca na tela o caractere cujo código está no acumulador.

RST 18 — coloca no acumulador o caractere cujo endereço é dado pela variável do sistema presente no endereço 16406.

RST 20 — obtém o caractere seguinte ao apontado pela variável do sistema presente no endereço 16406.

RST 28 — salta para a rotina de cálculo em ponto flutuante.

RST 30 — organiza o espaço na área de trabalho.

RST 38 — rotina de interrupção, onde o registrador B guarda o número da linha, e o registrador C, o número da linha investigada.

A investigação detalhada destas rotinas e do ROM é feita em obra posterior, a ser publicada, de caráter não introdutório. Contudo, experimente esta pequena rotina.

```
30000    3E80        LD A,128
30002    D7          RST 10H
30003    C9          RET
```

A tabela abaixo fornece o código em HEX destas instruções.

RST 00	C7
RST 08	CF
RST 10	D7
RST 18	DF
RST 20	E7
RST 28	EF
RST 30	F7
RST 38	FF

EXERCÍCIOS:

1. Procure realizar todas as sugestões do exercício que foram dadas neste capítulo.
2. Coloque em linguagem máquina o programa em BASIC fornecido no EPÍLOGO. Sugestão de endereço: 32000.

CAPÍTULO VII
GRUPO DE ROTAÇÃO E DE "DESLOCAMENTO"

I - INSTRUÇÃO DE ROTAÇÃO

Não iremos estudar, em detalhe, todos os tipos de rotação. Contudo, explicaremos, através de diversos exercícios, certas operações de rotação.

Vamos supor que se deseje fazer uma rotação circular para a esquerda, do registrador C, e que este contém o valor 1.

	B7	B6	B5	B4	B3	B2	B1	BØ
antes:	Ø	Ø	Ø	Ø	Ø	Ø	Ø	1
depois:	Ø	Ø	Ø	Ø	Ø	Ø	1	Ø

Pode-se constatar que o valor contido no BIT Ø passou para o BIT 1. Qual a conseqüência disto?

O valor presente no registrador C foi duplicado. Portanto, uma rotação para a esquerda multiplica o valor presente no registrador. Contudo, o que ocorrerá se o valor for superior a 127?

Sabemos que um registrador de oito BITs pode ser carregado no máximo com 255. Logo, se ele possuir um valor superior a 127, a instrução de rotação para a esquerda acarretará um "CARRY", e o valor em retorno será o valor original multiplicado por dois, subtraído de 256. O valor do "CARRY" será somado ou não ao resultado,

dependendo do tipo de instrução, ou seja, se ela for do tipo circular ou não.

Os exercícios abaixo ilustram as considerações feitas até agora.

EXERCÍCIOS ILUSTRATIVOS

a) **Rotação circular para a esquerda**

```
30000    0600        LD B,0
30002    0E00        LD C,0
30004    CB01        RLC C
30006    3E00        LD A,0
30008    8F          ADC A,A
30009    323F75      LD (30015),A
30012    C9          RET
```

```
PROGRAMA:

10 INPUT X
20 POKE 30003,X
30 PRINT USR 30000;" CARRY = ";
PEEK 30015
40 GOTO 10
```

Através da variação do valor de X, pode-se constatar que, quando o "CARRY" é igual a um, o valor em retorno é o dobro, subtraído de 256, somado a um.

A tabela abaixo fornece o código em HEX destas instruções.

RLC A	CB07
RLC B	CB00
RLC C	CB01
RLC D	CB02
RLC E	CB03
RLC H	CB04
RLC L	CB05

b) **Rotação para a esquerda através do "CARRY"**

Substitua na rotina anterior a instrução RLC por RL C. Pode-se constatar que, diferentemente do caso anterior, quando o "CARRY" é igual a um, o valor em retorno é tão-somente o dobro do valor original, subtraído de 256.

A tabela abaixo fornece o código em HEX destas instruções.

RL A	CB 17
RL B	CB 10
RL C	CB 11
RL D	CB 12
RL E	CB 13
RL H	CB 14
RL L	CB 15

c) **Rotação circular para a direita**

Suponhamos que o registrador C contém dois, antes da RRC C. Após a sua implementação, ele conterá um; ou seja, ele foi dividido por dois. Contudo, isto só ocorre quando o valor presente no registrador for par e maior ou igual a 2.

	B7	B6	B5	B4	B3	B2	B1	B0
antes	0	0	0	0	0	0	1	0
depois	0	0	0	0	0	0	0	1

Para comprovar as considerações que acabamos de fazer, e testar o que ocorre quando o valor for ímpar ou menor do que 2, entre com a seguinte rotina:

```
30000    0600        LD B,0
30002    0E00        LD C,0
30004    CB09        RRC C
30006    3E00        LD A,0
30008    8F          ADC A,A
30009    323F75      LD (30015),A
30012    C9          RET
```

Utilize o mesmo programa dado no primeiro exercício.

Constata-se que, se o valor presente em C for nulo, o resultado é igualmente nulo, e o "CARRY" é zero. Se o valor for ímpar, "CARRY" é igual a um, e o valor em retorno é o inicial somado a 255, dividido por dois. Portanto, se o registrador C for carregado com 255, após RRC C, teremos 255.

A tabela abaixo fornece o código em HEX destas instruções.

RRC A	CB0F
RRC B	CB08
RRC C	CB09
RRC D	CB0A
RRC E	CB0B
RRC H	CB0C
RRC L	CB0D

d) **Rotação para a direita através do "CARRY".**

Substitua na rotina anterior a instrução RRC C por RC C. Pode-se constatar que a única diferença reside no tratamento que recebem os valores ímpares. Aqui, o valor em retorno será o valor contido no registrador C, subtraído de um, dividido por dois. Portanto, se o registrador C for carregado com 255, o valor em retorno será 127.

A tabela abaixo fornece o código em HEX destas instruções.

```
RR A   CB 1F
RR B   CB 18
RR C   CB 19
RR D   CB 1A
RR E   CB 1B
RR H   CB 1C
RR L   CB 1D
```

A rotina abaixo utiliza este tipo de instrução para converter números decimais em hexadecimais, no intervalo entre 0 e 255.

```
30000   3E00      LD A,0
30002   F5        PUSH AF
30003   E6F0      AND 240
30005   1F        RRA
30006   1F        RRA
30007   1F        RRA
30008   1F        RRA
30009   C61C      ADD A,28
30011   D7        RST 10H
30012   F1        POP AF
30013   E60F      AND 15
30015   C61C      ADD A,28
30017   D7        RST 10H
30018   C9        RET
```

Estude cuidadosamente cada instrução, procurando entender o que ela está realizando. O programa abaixo permite a você variar o decimal.

```
10 INPUT X
20 POKE 30001,X
30 RAND USR 30000
40 PRINT
50 GOTO 10
```


II — Instruções de DESLOCAMENTO (SHIFT).

Não iremos estudar todos os tipos de deslocamento. Contudo, explicaremos, através de exercícios, certas operações de **SHIFT**.

a) **Deslocamento aritmético para a esquerda**

Vamos supor que desejamos efetuar um deslocamento aritmético para a esquerda do registrador C, que contém o valor 1.

	B7	B6	B5	B4	B3	B2	B1	B0
antes:	0	0	0	0	0	0	0	1
depois:	0	0	0	0	0	0	1	0

Após SLA (SHIFT LEFT ARITHMETIC), o valor presente em C foi duplicado, e o "CARRY" ficou igual a zero; isto é, o valor presente no BIT 7 é transferido para o "CARRY FLAG", e o BIT 0 foi zerado.

A rotina abaixo permite testar este tipo de instrução.

```
30000   0600      LD B,0
30002   0E00      LD C,0
30004   CB21      SLA C
30006   3E00      LD A,0
30008   8F        ADC A,A
30009   323F75    LD (30015),A
30012   C9        RET
```

O programa dado no primeiro exercício de rotação, permite variar o valor presente em C e controlar o "CARRY FLAG".

Se este valor for superior a 127, o resultado de SLA C será o dobro do valor presente em C, subtraído de 256, e o "CARRY FLAG" será ajustado, isto é, igual a um.

A tabela abaixo fornece o código em HEX destas instruções.

```
SLA A   CB 27
SLA B   CB 20
SLA C   CB 21
SLA D   CB 22
SLA E   CB 23
SLA H   CB 24
SLA L   CB 25
```

b) **Deslocamento aritmético para a direita**

Para testar esta instrução, entre com a rotina dada no exercício anterior, substituindo a instrução SLA C por SRA C.

Pode-se constatar que:

i) para valores pares até 126, o resultado em retorno é o valor presente em C, dividido por dois. O BIT 7 não é alterado, e o valor do BIT Ø vai para o "CARRY". Por exemplo, se C for carregado com 12Ø, teremos:

	B7	B6	B5	B4	B3	B2	B1	BØ	
antes:	Ø	1	1	1	1	Ø	Ø	Ø	= 12Ø
depois:	Ø	Ø	1	1	1	1	Ø	Ø	= 6Ø

ii) para valores pares maiores do que 126, o resultado será o valor contido em C, dividido por dois, somado a 128. O "CARRY FLAG" é reajustado, ou seja, igual a zero.

	B7	B6	B5	B4	B3	B2	B1	BØ
antes:	1	Ø	Ø	Ø	Ø	Ø	Ø	Ø
depois:	1	1	Ø	Ø	Ø	Ø	Ø	Ø

Em outras palavras, não há alteração no valor do BIT 7, e o valor presente no BIT Ø é copiado pelo "CARRY FLAG";

iii) para valores ímpares até 127, o resultado da instrução SRA C será o valor contido em C, subtraído de um, dividido por dois;

IV) para valores ímpares superiores a 127, o resultado será o valor contido em C, subtraído de um, dividido por dois, somado a 128.

Em suma, em todos os casos, o BIT 7 é conservado, o "CARRY FLAG" copia o BIT Ø, e há um deslocamento para a direita.

A tabela abaixo fornece o código em HEX destas instruções.

SRA A	CB 2F
SRA B	CB 28
SRA C	CB 29
SRA D	CB 2A
SRA E	CB 2B
SRA H	CB 2C
SRA L	CB 2D

c) Deslocamento lógico para a direita

Para testar esta instrução, utilize a mesma rotina, substituindo a instrução SRA C por SRL C. O programa em BASIC é o mesmo.

Pode-se verificar que se o número for par, ele será dividido por dois. Se for ímpar, será subtraído de um e, em seguida, dividido por dois. Apenas neste caso, o "CARRY FLAG" será igual a um.

Portanto, o BIT 7 será reajustado e o BIT Ø copiado pelo "CARRY FLAG". Todos os conteúdos serão alterados um BIT para a direita.

A tabela abaixo fornece o código em HEX destas instruções.

SRL A	CB 3F
SRL B	CB 38
SRL C	CB 39
SRL D	CB3A
SRL E	CB 3B
SRL H	CB 3C
SRL L	CB 3D

EXERCÍCIOS

1) Escreva uma rotina que converta números decimais em hexadecimais, no intervalo, entre Ø e 65535.

2) Escreva uma rotina que permita encontrar o quadrado de um número dado.

CAPÍTULO VIII
GRUPO DE MANIPULAÇÃO DE BIT

Neste último capítulo, desejamos indicar a possibilidade de se manipular diretamente os BITs. Podemos dividir este grupo em três partes, de acordo com o tipo de manipulação envolvida.

a) **Ativação do BIT** — as instruções são da forma SET **x, y**, onde **x** representa o número do BIT envolvido (portanto, varia entre 0 e 7), e **y** está no lugar, ou de um registrador, simples ou indexado (neste caso, o registrador está apontando para o conteúdo de uma localidade na memória), ou do conteúdo de uma localidade na memória. Por exemplo, a instrução SET 6,A resulta na ativação do BIT 6 do acumulador; isto é, após SET 6,A o BIT 6 do acumulador será um.

A rotina abaixo permite inverter todos os caracteres na tela que estão no intervalo entre 0 e 63 através da ativação do BIT 7 do acumulador.

```
30000    2A0C40      LD HL,(16396)
30003    0616        LD B,22
30005    23          INC HL
30006    7E          LD A,(HL)
30007    FE76        CP 118
30009    2805        JR Z,5
```

```
30011    CBFF        SET 7,A
30014    18F5        JR -11
30016    77          LD (HL),A
30017    10F2        DJNZ -14
30019    C9          RET
```

Escreva um programa em BASIC que utilize a rotina acima.

A tabela abaixo fornece o código em HEX destas instruções.

SET 0, A	CB C7	SET 1,A	CB CF	SET 2,A	CB D7	SET 3,A	CB DF
SET 0,B	CB C0	SET 1,B	CB C8	SET 2,B	CB D0	SET 3,B	CB D8
SET 0,C	CB C1	SET 1,C	CB C9	SET 2,C	CB D1	SET 3,C	CB D9
SET 0,D	CB C2	SET 1,D	CB CA	SET 2,D	CB D2	SET 3,D	CB DA
SET 0,E	CB C3	SET 1,E	CB CB	SET 2,E	CB D3	SET 3,E	CB DB
SET 0,H	CB C4	SET 1,H	CB CC	SET 2,H	CB D4	SET 3,H	CB DC
SET 0,L	CB C5	SET 1,L	CB CD	SET 2,L	CB D5	SET 3,L	CB DD

SET 4,A	CB E7	SET 5,A	CB EF	SET 6,A	CB F7	SET 7,A	CB FF
SET 4,B	CB E0	SET 5,B	CB E8	SET 6,B	CB F0	SET 7,B	CB F8
SET 4,C	CB E1	SET 5,C	CB E9	SET 6,C	CB F1	SET 7,C	CB F9
SET 4,D	CB E2	SET 5,D	CB EA	SET 6,D	CB F2	SET 7,D	CB FA
SET 4,E	CB E3	SET 5,E	CB EB	SET 6,E	CB F3	SET 7,E	CB FB
SET 4,H	CB E4	SET 5,H	CB EC	SET 6,H	CB F4	SET 7,H	CB FC
SET 4,L	CB E5	SET 5,L	CB ED	SET 6,L	CB F5	SET 7,L	CB FD

b) **Restauração do BIT** — semelhante ao anterior, diferindo no fato de o BIT indicado pela instrução ser restaurado, ou seja, zerado.

A rotina abaixo permite inverter todos os caracteres na tela que possuam código superior a 127.

```
30000    2A0C40      LD HL,(16396)
30003    0616        LD B,22
30005    23          INC HL
30006    7E          LD A,(HL)
30007    FE76        CP 118
30009    2805        JR Z,5
30011    CBBF        RES 7,A
30013    77          LD (HL),A
30014    18F5        JR -11
30016    77          LD (HL),A
30017    10F2        DJNZ -14
30019    C9          RET
```


A tabela abaixo fornece em HEX o código destas instruções.

RES 0,A	CB87	RES 1,A	CB8F	RES 2,A	CB97	RES 3,A	CB9F
RES 0,B	CB80	RES 1,B	CB88	RES 2,B	CB90	RES 3,B	CB98
RES 0,C	CB81	RES 1,C	CB89	RES 2,C	CB91	RES 3,C	CB99
RES 0,D	CB82	RES 1,D	CB8A	RES 2,D	CB92	RES 3,D	CB9A
RES 0,E	CB83	RES 1,E	CB8B	RES 2,E	CB93	RES 3,E	CB9B
RES 0,H	CB84	RES 1,H	CB8C	RES 2,H	CB94	RES 3,H	CB9C
RES 0,L	CB85	RES 1,L	CB8D	RES 2,L	CB95	RES 3,L	CB9D

RES 4,A	CBA7	RES 5,A	CBAF	RES 6,A	CBB7	RES 7,A	CBBF
RES 4,B	CBA0	RES 5,B	CBA8	RES 6,B	CBB0	RES 7,B	CBB8
RES 4,C	CBA1	RES 5,C	CBA9	RES 6,C	CBB1	RES 7,C	CBB9
RES 4,D	CBA2	RES 5,D	CBAA	RES 6,D	CBB2	RES 7,D	CBBA
RES 4,E	CBA3	RES 5,E	CBAB	RES 6,E	CBB3	RES 7,E	CBBB
RES 4,H	CBA4	RES 5,H	CBAC	RES 6,H	CBB4	RES 7,H	CBBC
RES 4,L	CBA5	RES 5,L	CBAD	RES 6,L	CBB5	RES 7,L	CBBD

c) **Teste do BIT** — as instruções são da forma BIT **x,y** onde **x** representa o número do BIT (intervalo entre 0 e 7), e **y** pode ser um registrador, simples ou indexado (neste caso, o registrador aponta para o conteúdo de uma localidade na memória), ou o conteúdo de uma localidade na memória. Por exemplo, BIT 0, A testa o estado do BIT 0 do acumulador. Se ele for zero, então o "ZERO FLAG" será ativado; caso contrário, será restaurado.

A rotina abaixo permite converter números decimais em binários, no intervalo entre 0 e 255, através do teste do BIT 7 do registrador C.

```
30000    0E00        LD C,0
30002    0608        LD B,8
30004    CB79        BIT 7,C
30006    3E1C        LD A,28
30008    2801        JR Z,1
30010    3C          INC A
30011    D7          RST 10H
30012    CB11        RL C
30014    10F4        DJNZ -12
30016    C9          RET
```

Escreva um programa em BASIC que permita variar o número em decimal.

A tabela abaixo fornece em HEX o código destas instruções.

BIT 0,A CB 47	BIT 1,A CB 4F	BIT 2,A CB 57	BIT 3,A CB 5F
BIT 0,B CB 40	BIT 1,B CB 48	BIT 2,B CB 50	BIT 3,B CB 58
BIT 0,C CB 41	BIT 1,C CB 49	BIT 2,C CB 51	BIT 3,C CB 59
BIT 0,D CB 42	BIT 1,D CB 4A	BIT 2,D CB 52	BIT 3,D CB 5A
BIT 0,E CB 43	BIT 1,E CB 4B	BIT 2,E CB 53	BIT 3,E CB 5B
BIT 0,H CB 44	BIT 1,H CB 4C	BIT 2,H CB 54	BIT 3,H CB 5C
BIT 0,L CB 45	BIT 1,L CB 4D	BIT 2,L CB 55	BIT 3,L CB 5D
BIT 4,A CB 67	BIT 5,A CB 6F	BIT 6,A CB 77	BIT 7,A CB 7F
BIT 4,B CB 60	BIT 5,B CB 68	BIT 6,B CB 70	BIT 7,B CB 78
BIT 4,C CB 61	BIT 5,C CB 69	BIT 6,C CB 71	BIT 7,C CB 79
BIT 4,D CB 62	BIT 5,D CB 6A	BIT 6,D CB 72	BIT 7,D CB 7A
BIT 4,E CB 63	BIT 5,E CB 6B	BIT 6,E CB 73	BIT 7,E CB 7B
BIT 4,H CB 64	BIT 5,H CB 6C	BIT 6,H CB 74	BIT 7,H CB 7C
BIT 4,L CB 65	BIT 5,L CB 6D	BIT 6,L CB 75	BIT 7,L CB 7D

Exercícios:

1) Escrevauma rotina que converta números decimais em binários no intervalo entre 0 e 65535.
2) Escreva uma rotina que inverta a tela, não importando se os caracteres já estão invertidos ou não.

EPÍLOGO

Digite a rotina abaixo.

```
30000   00          NOP
30001   00          NOP
30002   3E00        LD A,0
30004   329275      LD (30098),A
30007   3A3440      LD A,(16436)
30010   329875      LD (30104),A
30013   2A0C40      LD HL,(16396)
30016   118500      LD DE,181
30019   19          ADD HL,DE
30020   3634        LD (HL),52
30022   223075      LD (30000),HL
30025   3E00        LD A,0
30027   0630        LD B,48
30029   0EFF        LD C,255
30031   0D          DEC C
30032   20FD        JR NZ,-3
30034   10F9        DJNZ -7
30036   CDBB02      CALL 699
30039   44          LD B,H
30040   4D          LD C,L
30041   2C          INC L.
30042   28F8        JR Z,-8
30044   CDBD07      CALL 1981
30047   4E          LD C,(HL)
30048   79          LD A,C
30049   2A3075      LD HL,(30000)
30052   FE21        CP 33
30054   280E        JR Z,14
30056   FE22        CP 34
30058   2814        JR Z,20
30060   FE23        CP 35
30062   2815        JR Z,21
30064   FE24        CP 36
30066   2807        JR Z,7
30068   18DE        JR -34
30070   3600        LD (HL),0
30072   2B          DEC HL
30073   1810        JR 16
```

```
30075    3600        LD (HL),0
30077    23          INC HL
30078    180B        JR 11
30080    112100      LD DE,33
30083     1803        JR 3
30085     110FFF      LD DE,-33
30088     3600        LD (HL),0
30090     19          ADD HL,DE
30091     7E          LD A,(HL)
30092     FE00        CP 0
30094     C0          RET NZ
30095     3614        LD (HL),20
30097     3E00        LD A,0
30099     3C          INC A
30100     329275      LD (30098),A
30103     3E00        LD A,0
30105     47          LD B,A
30106     F6DF        OR 223
30108     EEDF        XOR 223
30110     CB07        RLC A
30112     CB07        RLC A
30114     CB08        RRC B
30116     A8          XOR B
30117     329875      LD (30104),A
30120     22CF75      LD (30159),HL
30123     FEC0        CP 192
30125     3803        JR C,3
30127     2B          DEC HL
30128     1815        JR 21
30130     FE80        CP 128
30132     3806        JR C,6
30134     112100      LD DE,33
30137     19          ADD HL,DE
30138     1808        JR 11
30140     FE40        CP 64
30142     3806        JR C,6
30144     11DFFF      LD DE,-33
30147     19          ADD HL,DE
30148     1801        JR 1
30150     23          INC HL
```



```
30151    3697        LD (HL),151
30153    2ACF75      LD HL,(30159)
30156    C34675      JP 30022
30159    00          NOP
30160    00          NOP
```

Para chamá-la, digite RAND USR 30002. Ela deve ser cercada com o seguinte programa em BASIC (modo SLOW):

```
10 LET S$ ="32 espaços no modo gráfico"
20 LET L$ ="modo gráfico 5,30 espaços, modo gráfico 8"
30 PRINT S$
40 FOR T =1 T0 9
50 PRINT L$
60 NEXT T
70 PRINT S$
80 RAND USR 30002
90 PRINT PEEK 30098
```

O que ocorre? Descubra investigando a rotina acima. Dica: as teclas 5, 6, 7 e 8 controlam certas ocorrências na tela. (Caso você não consiga entender, leia o Apêndice 6).

APÊNDICE 1
TABELA DE CARACTERES

0		27	↑	54	6	145	
1	▝	28	0	55	7	146	
2	▘	29	1	56	8	147	
3	▀	30	2	57	9	148	
4	▗	31	3	58	:	149	
5	▐	32	4	59	;	150	
6	▚	33	5	60	<	151	
7	▛	34	6	61	=	152	
8	▒	35	7	62	>	153	
9	▖	36	8	63	?	154	
10	▌	37	9	128		155	
11	▜	38	:	129	▟	156	
12	£	39	;	130	▙	157	
13	$	40	<	131	▄	158	
14	:	41	=	132	▜	159	
15	?	42	>	133	▌	160	
16	(	43	?	134	▚	161	
17	)	44	@	135	▗	162	
18	>	45	[	136	▒	163	
19	<	46	\	137	▒	164	
20	=	47	]	138	▀	165	
21	+	48	↑	139	▀	166	
22	-	49	£	140	▐	167	
23	*	50	A	141	▐	168	
24	/	51	B	142	▀	169	
25	;	52	C	143	▀	170	
26	,	53	D	144	▌	171	

172		190	
173		191	
174			
175			
176			
177			
178			
179			
180			
181			
182			
183			
184			
185			
186			
187			
188			
189			
190			
191			

APÊNDICE 2
TABELA DE CONVERSÃO DE DECIMAL EM HEX

0=00	38=26	76=4C	114=72	152=98	190=BE	228=E4
1=01	39=27	77=4D	115=73	153=99	191=BF	229=E5
2=02	40=28	78=4E	116=74	154=9A	192=C0	230=E6
3=03	41=29	79=4F	117=75	155=9B	193=C1	231=E7
4=04	42=2A	80=50	118=76	156=9C	194=C2	232=E8
5=05	43=2B	81=51	119=77	157=9D	195=C3	233=E9
6=06	44=2C	82=52	120=78	158=9E	196=C4	234=EA
7=07	45=2D	83=53	121=79	159=9F	197=C5	235=EB
8=08	46=2E	84=54	122=7A	160=A0	198=C6	236=EC
9=09	47=2F	85=55	123=7B	161=A1	199=C7	237=ED
10=0A	48=30	86=56	124=7C	162=A2	200=C8	238=EE
11=0B	49=31	87=57	125=7D	163=A3	201=C9	239=EF
12=0C	50=32	88=58	126=7E	164=A4	202=CA	240=F0
13=0D	51=33	89=59	127=7F	165=A5	203=CB	241=F1
14=0E	52=34	90=5A	128=80	166=A6	204=CC	242=F2
15=0F	53=35	91=5B	129=81	167=A7	205=CD	243=F3
16=10	54=36	92=5C	130=82	168=A8	206=CE	244=F4
17=11	55=37	93=5D	131=83	169=A9	207=CF	245=F5
18=12	56=38	94=5E	132=84	170=AA	208=D0	246=F6
19=13	57=39	95=5F	133=85	171=AB	209=D1	247=F7
20=14	58=3A	96=60	134=86	172=AC	210=D2	248=F8
21=15	59=3B	97=61	135=87	173=AD	211=D3	249=F9
22=16	60=3C	98=62	136=88	174=AE	212=D4	250=FA
23=17	61=3D	99=63	137=89	175=AF	213=D5	251=FB
24=18	62=3E	100=64	138=8A	176=B0	214=D6	252=FC
25=19	63=3F	101=65	139=8B	177=B1	215=D7	253=FD
26=1A	64=40	102=66	140=8C	178=B2	216=D8	254=FE
27=1B	65=41	103=67	141=8D	179=B3	217=D9	255=FF
28=1C	66=42	104=68	142=8E	180=B4	218=DA	
29=1D	67=43	105=69	143=9F	181=B5	219=DB	
30=1E	68=44	106=6A	144=90	182=B6	220=DC	
31=1F	69=45	107=6B	145=91	183=B7	221=DD	
32=20	70=46	108=6C	146=92	184=B8	222=DE	
33=21	71=47	109=6D	147=93	185=B9	223=DF	
34=22	72=48	110=6E	148=94	186=BA	224=E0	
35=23	73=49	111=6F	149=95	187=BB	225=E1	
36=24	74=4A	112=70	150=96	188=BC	226=E2	
37=25	75=4B	113=71	151=97	189=BD	227=E3	

APÊNDICE 3: VARIÁVEIS DO SISTEMA

Endereço		Função
16384	(4 000)	Código de erro.
16385	(4 001)	"Sinalizadores" Bit 0 - Controla espaço. Bit 1 - Controla impressora. Bit 2 - Controla modo de funcionamento: K, L, F ou G. Bit 6 - Controla parâmetros. Bit 7 - Controla sintaxe.
16386	(4 002)	Aponta para o topo do "Stack" após uma
	(4 003)	Instrução de Gosub.
16388	(4 004)	O topo do RAM existente ou
	(4005)	fixado pelo usuário
16390	(4 006)	Especifica o cursor que está sendo usado.
16391	(4 007)	Número da linha que está sendo executada.
	(4 008)	
16393	(4009)	O ponto inicial do RAM que será guardado em tape. A partir de um comando "Save".
16394	(4 00A)	A linha de BASIC onde está o cursor.
	(4 00B)	
16396	(4 00C)	Ponteiro para o arquivo de imagem.
	(4 00D)	
16398	(4 00E)	Endereço que indica o local para "Print At".
	(400F)	
16400	(4010)	Ponteiro para a área de variáveis.
	(4011)	
16402	(4012)	Endereço da variável sendo executada dentro da
	(4013)	área de programa.
16404	(4014)	Ponteiro para área de trabalho.
	(4015)	
16406	(4016)	Ponteiro que procura uma linha dentro da
	(4017)	Área de programa ou da área de trabalho.
16408	(4018)	Endereço do erro de sintaxe.
	(4019)	
16410	(401A)	Ponteiro que aponta o início
	(401B)	do "stack" de cálculo.
16412	(401C)	Ponteiro que aponta para o
	(401D)	final do "stack" de cálculo.
16414	(401E)	Usado pelo calculador em ponto flutuante.
16415	(401F)	Endereço que aponta para a área onde
	(4020)	serão feitos os cálculos.
6417	(4021)	Não é usado.

16418	(4022)	Número de linhas na parte inferior da tela.
16419	(4023)	A linha a ser chamada automaticamente na
	(4024)	listagem.
16421	(4025)	Valor da última tecla pressionada.
	(4026)	
16423	(4027)	Estado de não oscilação do teclado.
16424	(4028)	Números de linhas acima ou abaixo da imagem.
16425	(4029)	Endereço da próxima linha de BASIC a ser
	(402A)	interpretada.
16427	(402B)	Número da linha para a qual uma
	(402C)	instrução "Cont" salta.
16429	(402D)	"**Sinalizadores**": Bit 0 - 0 indica variável indexada. Bit 1 - 0 indica que uma variável dada existe. Bit 5 - 1 indica instrução Input em funcionamento. Bit 6 - 1 indica que o comando Input espera um valor numérico.
16430	(402E)	Extensão de uma **string** ou de uma linha de BASIC.
	(402F)	
16432	(4030)	Ponteiro para tabela de sintaxe.
(4031)		
16434	(4032)	O valor originário colocado na função RND.
	(4033)	
16436	(4034)	O contador de quadros gerados na tela.
	(4035)	
16438	(4036)	Coordenada x do último ponto colocado em tela.
	(4037)	Coordenada Y do último ponto colocado na tela.
16440	(4038)	Contador para a posição da Impressora.
16441	(4039)	Número da linha e da coluna, respectivamente para
	(403A)	o comando Print At.
16443	(4030)	**Sinalizadores:** BIT 0 - 1 quando é pressionada uma tecla. BIT 6 - controla "SLOW"/"FAST". BIT 7 - controla "SLOW"/"FAST" durante execução de "COPY"; 0 quando no modo "FAST"
16444	(403C)	Armazenamento da impressora
16477	(405D)	Área da memória que pode guardar até seis números representados em ponto flutuante.
16507	(407B)	Não são usados.
	(407c)	

A primeira listagem decorre do seguinte programa:

```
10 For n=16384 TO 16507
20 Print N; " = "; Peek N, N + 1; "="; Peek (N+1)
```

```
30 LET N=N + 1
40 NEXT N
```

A listagem seguinte resulta do programa acima, substituindo-se a linha 10 por 10 FOR N=16509 TO 16608.
Procure interpretar o conteúdo dos endereços em termos das variáveis do sistema. Use a segunda listagem para entender como é estruturada a área de programa.

VARIÁVEIS DO SISTEMA

```
16384 = 255   16415 = 93    16446 = 0   16477 = 0
16385 = 192   16416 = 64    16447 = 0   16478 = 0
16386 = 252   16417 = 0     16448 = 0   16479 = 0
16387 = 127   16418 = 2     16449 = 0   16480 = 0
16388 = 0     16419 = 0     16450 = 0   16481 = 0
16389 = 128   16420 = 0     16451 = 0   16482 = 13
16390 = 0     16421 = 255   16452 = 0   16483 = 0
16391 = 20    16422 = 255   16453 = 0   16484 = 0
16392 = 0     16423 = 0     16454 = 0   16485 = 0
16393 = 0     16424 = 31    16455 = 0   16486 = 0
16394 = 40    16425 = 201   16456 = 0   16487 = 128
16395 = 0     16426 = 64    16457 = 0   16488 = 128
16396 = 225   16427 = 0     16458 = 0   16489 = 128
16397 = 64    16428 = 0     16459 = 0   16490 = 128
16398 = 209   16429 = 0     16460 = 0   16491 = 128
16399 = 65    16430 = 115   16461 = 0   16492 = 128
16400 = 250   16431 = 230   16462 = 0   16493 = 128
16401 = 67    16432 = 107   16463 = 0   16494 = 128
16402 = 250   16433 = 12    16464 = 0   16495 = 128
16403 = 67    16434 = 0     16465 = 0   16496 = 134
16404 = 13    16435 = 0     16466 = 0   16497 = 153
16405 = 68    16436 = 21    16467 = 0   16498 = 148
16406 = 171   16437 = 192   16468 = 0   16499 = 150
16407 = 64    16438 = 00    16469 = 0   16500 = 145
16408 = 167   16439 = 188   16470 = 0   16501 = 144
16409 = 64    16440 = 9     16471 = 0   16502 = 0
16410 = 13    16441 = 17    16472 = 0   16503 = 0
16411 = 68    16442 = 192   16473 = 0   16504 = 0
16412 = 13    16443 = 0     16474 = 0   16505 = 0
16413 = 68    16444 = 0     16475 = 0   16506 = 0
16414 = 64    16445 = 0     16476 = 11  16507 = 0
```


ÁREA DO PROGRAMA

```
16509 = 0
16510 = 10
16511 = 27
16512 = 0
16513 = 235
16514 = 51
16515 = 20
16516 = 29
16517 = 34
16518 = 33
16519 = 28
16520 = 37
16521 = 126
16522 = 143
16523 = 0
16524 = 230
16525 = 0
16526 = 0
16527 = 223
16528 = 29
16529 = 34
16530 = 34
16531 = 28
16532 = 36
16533 = 126
16534 = 143
16535 = 1
16536 = 192
16537 = 0
16538 = 0
16539 = 118
16540 = 0
16541 = 20
16542 = 41
16543 = 0
16544 = 245
16545 = 51
16546 = 25
16547 = 11
16548 = 0
16549 = 20
16550 = 0
16551 = 11
16552 = 26
16553 = 211
16554 = 51
16555 = 26
16556 = 51
16557 = 21
16558 = 29
16559 = 126
16560 = 129
16561 = 0
16562 = 0
16563 = 0
16564 = 0
16565 = 25
16566 = 11
16567 = 0
16568 = 20
16569 = 0
16570 = 11
16571 = 25
16572 = 211
16573 = 16
16574 = 51
16575 = 21
16576 = 29
16577 = 126
16578 = 129
16579 = 0
16580 = 0
16581 = 0
16582 = 0
16583 = 17
16584 = 118
16585 = 0
16586 = 30
16587 = 13
16588 = 0
16589 = 241
16590 = 51
16591 = 20
16592 = 51
16593 = 21
16594 = 29
16595 = 126
16596 = 129
16597 = 0
16598 = 0
16599 = 0
16600 = 0
16601 = 118
16602 = 0
16603 = 40
16604 = 3
16605 = 0
16606 = 243
16607 = 51
16608 = 118
```

APÊNDICE 4: CÓDIGO DE INSTRUÇÕES DO Z80A

		CB	ED	FD	FDCB dis
00	NOP	RLC B			
01	LD BC, NN	RLC C			
02	LD(BC), A	RLC D			
03	INC BC	RLC E			
04	INC B	RLC H			
05	DEC B	RLC L			
06	LDB, N	RLC (HL)			RLC (IY+- d)
07	RLCA	RLC A			
08	EXAF, AF'	RRC B			
09	ADD HL,BC	RRC C		ADD IY, BC	
0A	LD A, (BC)	RRC D			
0B	DEC BC	RRC E			
0C	INC C	RRC H			
0D	DEC C	RRC L			
0E	LD C, N	RRC (HL)			RRC (IY+-d)
0F	RRCA	RRC A			
10	DJNZ dis	RL B			
11	LD DE, NN	RL C			
12	LD (DE), A	RL D			
13	INC DE	RL E			
14	INC D	RL H			
15	DEC D	RL L			
16	LDD, N	RL (HL)			RL (IY+-d)
17	RLA	RL A			
18	JR dis	RR B			
19	ADD HL, DE	RR C		ADD IY, DE	
1A	LD A, (DE)	RR D			
1B	DEC DE	RR E			
1C	INC E	RR H			
1D	DEC E	RR L			
1E	LD E, N	RR (HL)			RR (IY+-dis)
1F	RRA	RRA			
20	JR NZ, dis	SLA B			
21	LD HL, NN	SLA C		LD IY, NN	
22	LD (NN), HL	SLA D		LD(NN), IY	
23	INC HL	SLA E		INC IY	
24	INC H	SLA H			
25	DEC H	SLA L			
26	LD H, N	SLA (HL)			SLA (IY +- dis)
27	DAA	SLA A			

		CB	ED	FD	FDCB dis
28	JRZ, dis	SRA B			
29	ADD HL, HL	SRA C		ADD IY, IY	
2A	LD HL, (NN)	SRA D		LD IY, (NN)	
2B	DEC HL	SRA E		DEC IY	
2C	INC L	SRA H			
2D	DEC L	SRA L			
2E	LD L, N	SRA (HL)			SRA (IY +- dis)
2F	CPL	SRA A			
30	JR NC, dis				
31	LD SP, NN				
32	LD (NN), A				
33	INC SP				
34	INC (HL)			INC (IY +- dis)	
35	DEC (HL)			DEC (IY +- dis)	
36	LD(HL), N			LD (IY +- dis),N	
37	SCF				
38	JRC,dis	SRL B			
39	ADD HL,SP	SRL C		ADD IY, SP	
3A	LD A, (NN)	SRL D			
3B	DEC SP	SRL E			
3C	INC A	SRL H			
3D	DEC A	SRL L			
3E	LD A,N	SRL (HL)			SRL (IY +- dis)
3F	CCF	SRL A			
40	LD B,B	BIT 0,B	INB, (C)		
41	LD B,C	BIT 0, C	OUT (C),B		
42	LD B,D	BIT 0,D	SBC HL, BC		
43	LD B,E	BIT 0,E	LD(NN),BC		
44	LD B, H	BIT 0, H	NEG		
45	LD B,L	BIT 0,L	RET N		
46	LD B, (HL)	BIT 0, (HL)	IM 0	LD B, (IY +- dis)	BIT 0, (IY +- dis)
47	LD B,A	BIT 0, A	LD I,A		
48	LD C,B	BIT 1,B	IN C, (C)		
49	LD C,C	BIT1, C	OUT (C),C		
4A	LD C,D	BIT 1, D	ADC HL, BC		
4B	LD C,E	BIT 1, E	LD BC, (NN)		
4C	LD C, H	BIT 1, H			
4D	LD C,L	BIT 1, L	RET 2		
4E	LD C, (HL)	BIT 1, (HL)		LDC,(IY +- dis)	BIT 1, (IY +- dis)
4F	LD C,A	BIT 1, A	LD R,A		
50	LD D,B	BIT 2,B	IN D, (C)		
51	LD D,C	BIT 2,C	OUT (C),D		

		CB	ED	FD	FDCB dis
52	LD D,D	BIT 2,D	SBC HL,DE		
53	LD D,E	BIT 2,E	LD (NN),DE		
54	LD D,H	BIT 2,H			
55	LD D,L	BIT 2,L			
56	LD D,(HL)	BIT 2,(HL)	IM 1	LD D,(IY+dis)	BIT 2,(IY+dis)
57	LD D,A	BIT 2,A	LD A, I		
58	LD E,B	BIT 3,B	INE, (C)		
59	LD E,C	BIT 3,C	OUT (C),E		
5A	LD E,D	BIT 3,D	ADC HL,DE		
5B	LD E,E	BIT 3,E	LD DE,(NN)		
5C	LD E,H	BIT 3,H			
5D	LD E,L	BIT 3,L			
5E	LD E,(HL)	BIT 3,(HL)	IM 2	LD E,(IY+dis)	BIT 3, (IY+dis)
5F	LD E,A	BIT 3,A	LD A,R		
60	LD H,B	BIT 4,B	IN H,(C)		
61	LD H,C	BIT 4,C	OUT (C),H		
62	LD H,D	BIT 4,D	SBC HL,HL		
63	LD H,E	BIT 4,E	LD (NN),HL		
64	LD H,H	BIT 4,H			
65	LD H,L	BIT 4,L			BIT 4,(IY+dis)
66	LD H,(HL)	BIT 4,(HL)		LD H,(IY+dis)	
67	LD H,A	BIT 4,A	RRD		
68	LD L,B	BIT 5,B	IN L,(C)		
69	LD L,C	BIT 5,C	OUT (C),L		
6A	LD L,D	BIT 5,D	ADC HL,HL		
6B	LD L,E	BIT 5,E	LD DE,(NN)		
6C	LD L, H	BIT 5, H			
6D	LD L, L	BIT 5, L			
6E	LD L, (HL)	BIT 5,(HL)		LD L, (IY+dis)	BIT 5, (IY+dis)
6F	LD L, A	BIT 5, A			
70	LD(HL), B	BIT 6, B		LD(IY+dis), B	
71	LD(HL), C	BIT 6,C		LD(IY+dis), C	
72	LD(HL), D	BIT 6,D		LD(IY+dis), D	
73	LD(HL),E	BIT 6,E		LD(IY+dis), E	
74	LD(HL), H	BIT 6,H		LD(IY+dis), H	
75	LD(HL), L	BIT 6,L		LD(IY+dis), L	
76	HALT	BIT 6, (HL)			BIT 6, (IY+dis)
77	LD(HL), A	BIT 6,A		LD(IY+dis), A	
78	LD A, B	BIT 7,B			
79	LD A, C	BIT 7,C			
7A	LD A, D	BIT 7,D			
7B	LD A,E	BIT 7,E			

		CB	ED	FD	FDCB dis
7C	LD A,H	BIT 7,H			
7D	LD A,L	BIT 7,L			
7E	LD A, (HL)	BIT 7, (HL)		LD A, (IY+dis)	BIT 7, (IY+dis)
7F	LD A, A	BIT 7,A			
80	ADD A,B	RES 0,B			
81	ADD A,C	RES 0,C			
82	ADD A,D	RES 0,D			
83	ADD A,E	RES 0,E			
84	ADD A,H	RES0,H			
85	ADD A,L	RES 0,L			
86	ADD A, (HL)	RES 0, (HL)		ADD A,(IY+dis)	RES 0, (IY+dis)
87	ADD A,A	RES 0,A			
88	ADC A,B	RES 1,B			
89	ADC A,C	RES 1,C			
8A	ADC A,D	RES 1,D			
8B	ADC A,E	RES 1,E			
8C	ADC A,H	RES 1,H			
8D	ADC A,L	RES 1,L			
8E	ADC A,(HL)	RES 1,(HL)		ADC A,(IY+dis)	RES 1, (IY+dis)
8F	ADC A,A	RES 1,A			
90	SUB B	RES 2,B			
91	SUB C	RES 2,C			
92	SUB D	RES 2,D			
93	SUB E	RES 2,E			
94	SUB H	RES 2,H			
95	SUB L	RES 2,L			
96	SUB (HL)	RES 2,(HL)		SUB (IY+dis)	RES 2, (IY+dis)
97	SUB A	RES 2,A			
98	SBC A,B	RES 3,B			
99	SBC A,C	RES 3,C			
9A	SBC A, D	RES 3,D			
9B	SBC A,E	RES 3,E			
9C	SBC A, H	REB S 3,H			
9D	SBC A,L	RES D,L			
9E	SBC A, (HL)	RES 3, (HL)		SBC A, (IY+dis)	RES 3, (IY+dis)
9F	SBC A,A	RES 3,A			
A0	AND B	RES 4,B	LDI		
A1	AND C	RES 4,C	CPI		
A2	AND D	RES 4,D	INI		
A3	AND E	RES 4,E	OUTI		
A4	AND H	RES 4,H			
A5	AND L	RES 4,L			

		CB	ED	FD	FDCB dis
A6	AND (HL)	RES 4,(HL)		AND (IY+dis)	RES 4, (IY+dis)
A7	AND A	RES 4,A			
A8	XOR B	RES 5,B	LDD		
A9	XOR C	RES 5,C	CPD		
AA	XOR D	RES 5,D	IND		
AB	XOR E	RES 5,E	OUT D		
AC	XOR H	RES 5,H			
AD	XOR L	RES 5,L			
AE	XOR (HL)	RES 5,(HL)		XOR (IY+dis)	RES 5,(IY+dis)
AF	XOR A	RES 5,A			
B0	OR B	RES 6,B	LDIR		
B1	OR C	RES 6,C	CPIR		
B2	OR D	RES 6,D	INIR		
B3	OR E	RES 6,E	OTIR		
B4	OR H	RES 6,H			
B5	OR L	RES 6,L			
B6	OR (HL)	RES 6,(HL)		OR(IY+dis)	RES 6,(IY+dis)
B7	OR A	RES 6,A			
B8	CP B	RES 7,B	LDDR		
B9	CP C	RES 7,C	CPDR		
BA	CP D	RES 7,D	INDR		
BB	CP E	RES 7,E	OTDR		
BC	CP H	RES 7,H			
BD	CP L	RES 7,L			
BE	CP(HL)	RES 7,(HL)		CP(IY+dis)	RES 7,(IY+dis)
BF	CP A	RES 7,A			
C0	RET NZ	SET 0 B			
C1	POP BC	SET 0,C			
C2	JP NZ, NN	SET0, D			
C3	JP NN	SET 0, E			
C4	CALL NZ,NN	SET 0, H			
C5	PUSH BC	SET 0, L			SET 0,(IY+dis)
C6	ADD A. N	SET 0, (HL)			
C7	RST 0A	SET 0A			
C8	RET Z	SET 1,B			
C9	RET	SET 1,C			
CA	JP Z, NN	SET 1,D			
CB		SET 1,E			
CC	CALL Z, NN	SET 1,H			
CD	CALL NN	SET 1,L			SET 1, (IY+dis)
CE	ADC A,N	SET 1,(HL)			
CF	RST 8	SET 1,A			

		CB	ED	FD	FDCB
D0	RET NC	SET 2,B			
D1	POP DE	SET 2,C			
D2	JP NC, NN	SET 2,D			
D3	OUT N,A	SET 2,E			
D4	CALL NC, NN	SET 2,H			
D5	PUSH DE	SET 2,L			
D6	SUB N	SET 2, (HL)			SET 2, (IY+dis)
D7	RST 16	SET 2,A			
D8	RET C	SET 3,B			
D9	EXX	SET 3,C			
DA	JP C, NN	SET 3,D			
DB	IN A,N	SET 3,E			
DC	CALL C, NN	SET 3,H			
DD		SET 3,L			
DE	SBC A,N	SET 3,(HL)			SET 3 (IY+dis)
DF	RST 24	SET 3,A			
E0	RET PO	SET 4,B			
E1	POP HL	SET 4,C	POP IY		
E2	JP PO, NN	SET 4,D			
E3	EX(SP), HL	SET 4,E	EX(SP), IY		
E4	CALL PO, NN	SET 4,H			
E5	PUSH HL	SET 4,L	PUSH IY		
E6	AND N	SET 4, (HL)			SET 4, (IY+dis)
E7	RST 32	SET 4,A			
E8	RET PE	SET 5,B			
E9	JP (HL)	SET 5,C	JP(IY)		
EA	JP PE, NN	SET 5,D			
EB	EX DE, HL	SET 5,E			
EC	CALL PE, NN	SET 5,H			
ED		SET 5,L			
EE	XOR N	SET 5,(HL)		SET 5,(IY+dis)	
EF	RST 40	SET 5,A			
F0	RET P	SET 6,B			
F1	POP AF	SET 6,C			
F2	JP P,NN	SET 6,D			
F3	DI	SET 6,E			
F4	CALL P,NN	SET 6,H			
F5	PUSH AF	SET 6,L			
F6	OR N	SET 6,(HL)		SET 6,(IY+dis)	
F7	RST 48	SET 6,A			
F8	RET M	SET 7,B			
F9	LD SP,HL	SET 7,C		LD SP,IY	

		CB	ED	FD	FDCBdis
FA	JP M, NN	SET 7,D			
FB	EI	SET 7,E			
FC	CALL M,NN	SET 7,H			
FD		SET 7,L			
FE	CP N	SET 7,(HL)		SET 7,(IY+dis)	
FF	RST 56	SET 7,A			

REGRAS PARA UTILIZAÇÃO DA TABELA

a) Para obter o código das instruções que utilizam o registrador indexado IX, basta trocar todas as ocorrências de FD por DD.

b) A tabela deve ser lida da coluna para a linha. Por exemplo, o código CBFA indica e instrução que se encontra na intersecção entre a coluna CB e a linha FA, ou seja, SET 7,D.

c) As instruções que se iniciam com FDCBdis têm quatro BYTEs - o quarto BYTE é encontrado na linha, e o terceiro depende do deslocamento desejado. Por exemplo, BIT 6,(IY+00) tem o código FDCB0076, onde 00 indica um deslocamento nulo na representação complemento de dois (oito BITs), e 76 indica que se trata do BIT 6.

d) A tabela abaixo dá uma indicação aproximada (isto é, a maior dessas instruções vai estar no intervalo dado) de como encontrar o código de uma instrução que não utiliza registrador indexado.

TIPO DE INSTRUÇÃO	LOCAL APROXIMADO EM HEX
ROTAÇÃO	00 - 1F (CB)
DESLOCAMENTO	20 - 3F (CB)
LOAD registrador, registrador	40 - 7F
ARITMÉTICAS	80 - 9F
LÓGICAS	A0 - B7
MANIPULAÇÃO DE BIT	40 - FF (CB)
SALTO, CHAMADAS, RETORNO	C0 - FF

APÊNDICE 5 — TABELA DE COMPLEMENTO DE DOIS (OITO BITs)

-1 = FF	-33 = DF	-65 = BF	-97 = 9F
-2 = FE	-34 = DE	-66 = BE	-98 = 9E
-3 = FD	-35 = DD	-67 = BD	-99 = 9D
-4 = FC	-36 = DC	-68 = BC	-100 = 9C
-5 = FB	-37 = DB	-69 = BB	-101 = 9B
-6 = FA	-38 = DA	-70 = BA	-102 = 9A
-7 = F9	-39 = D9	-71 = B9	-103 = 99
-8 = F8	-40 = D8	-72 = B8	-104 = 98
-9 = F7	-41 = D7	-73 = B7	-105 = 97
-10 = F6	-42 = D6	-74 = B6	-106 = 96
-11 = F5	-43 = D5	-75 = B5	-107 = 95
-12 = F4	-44 = D4	-76 = B4	-108 = 94
-13 = F3	-45 = D3	-77 = B3	-109 = 93
-14 = F2	-46 = D2	-78 = B2	-110 = 92
-15 = F1	-47 = D1	-79 = B1	-111 = 91
-16 = F0	-48 = D0	-80 = B0	-112 = 90
-17 = EF	-49 = CF	-81 = AF	-113 = 8F
-18 = EE	-50 = CE	-82 = AE	-114 = 8E
-19 = ED	-51 = CD	-83 = AD	-115 = 8D
-20 = EC	-52 = CC	-84 = AC	-116 = 8C
-21 = EB	-53 = CB	-85 = AB	-117 = 8B
-22 = EA	-54 = CA	-86 = AA	-118 = 8A
-23 = E9	-55 = C9	-87 = A9	-119 = 89
-24 = E8	-56 = C8	-88 = A8	-120 = 88
-25 = E7	-57 = C7	-89 = A7	-121 = 87
-26 = E6	-58 = C6	-90 = A6	-122 = 86
-27 = E5	-59 = C5	-91 = A5	-123 = 85
-28 = E4	-60 = C4	-92 = A4	-124 = 84
-29 = E3	-61 = C3	-93 = A3	-125 = 83
-30 = E2	-62 = C2	-94 = A2	-126 = 82
-31 = E1	-63 = C1	-95 = A1	-127 = 81
-32 = E0	-64 = C0	-96 = A0	-128 = 80

Apêndice 6

Você só deve ler o que se segue se realmente procurou descobrir como funciona a rotina dada no epílogo. As observações abaixo irão ajudá-lo a compreendê-la.

Observações:

a) os endereços 30000 e 30001 guardam o endereço do arquivo de imagem;
b) o endereço 30098 guarda o escore do jogo — cada vez que a rotina passa por este ponto, há um incremento de um, ao escore anterior;
c) as instruções presentes entre 30013 e 30024 servem para colocar o caractere 0 na tela;
d) no endereço 30104 é guardado o número gerado pela variável do sistema que conta os quadros colocados na tela — é um número randômico;
e) as instruções presentes entre 30025 e 30035 definem um "atraso" (DELAY) — quanto menor ele for, mais rápido será o jogo — portanto, seria possível definir um grau de dificuldade para jogo, através da variação do valor carregado no registrador B (endereço 30028);
f) entre o endereço 30036 e 30048 é controlado o teclado — o valor do caractere correspondente à tecla acionada é guardado no acumulador (instrução LD A,C: endereço 30048);
g) se uma das teclas entre 5 e 8 for acionada, então a posição na tela, onde estava o caractere 0, será limpada e definido o novo local de impressão deste carectere;
h) se este novo local já estiver ocupado, então haverá um retorno para o BASIC, e o valor presente no endereço 30098 definirá o escore;
i) as instruções seguintes detectam se o caractere 0 já atingiu a borda do tabuleiro, assim como imprimem aleatoriamente, numa das posições contíguas à localidade anteriormente ocupada por 0, o caractere 151; note como o lugar a ser impresso é definido através de instruções lógicas de rotação e de comparação.