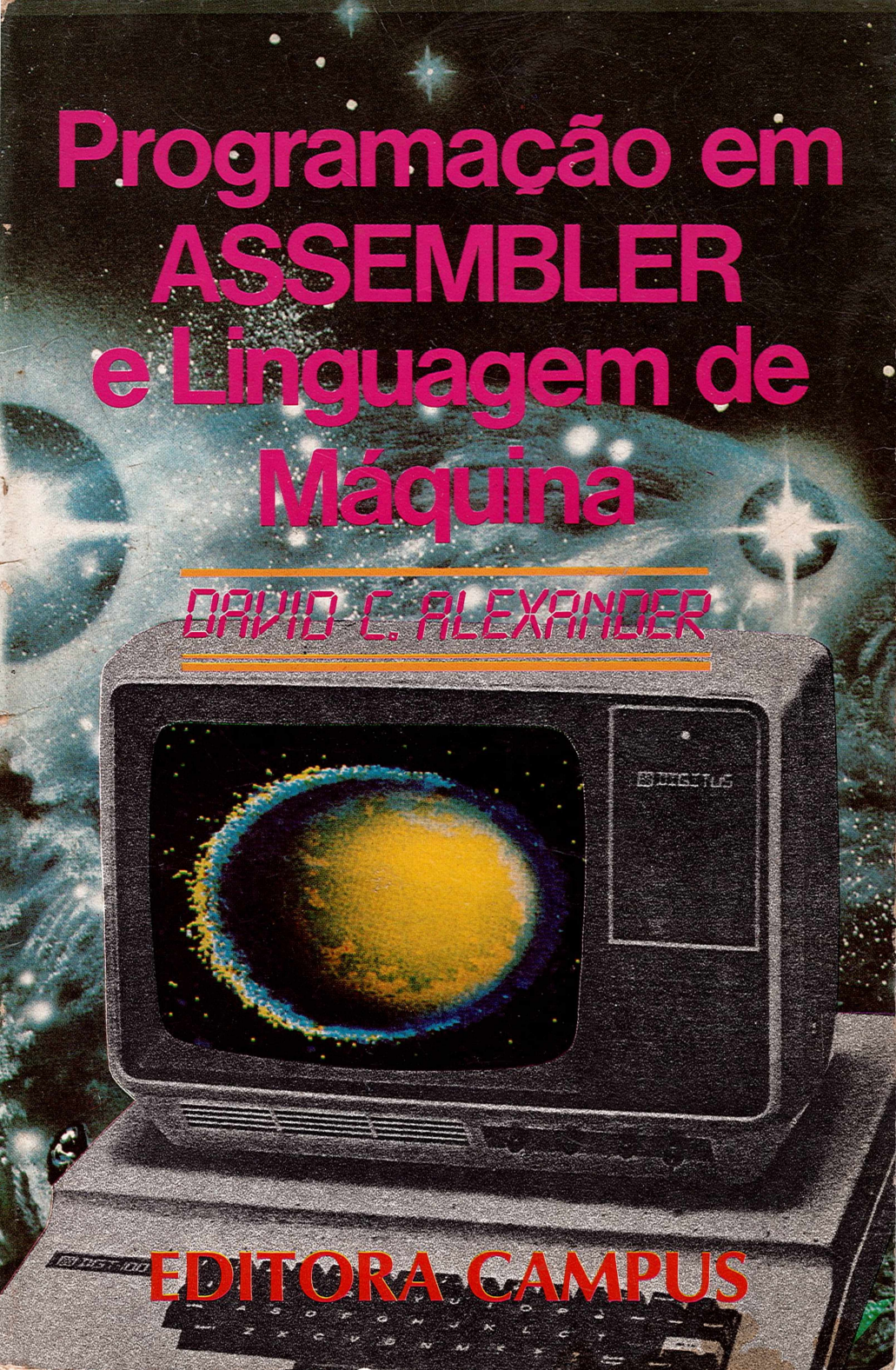
Programação em ASSEMBLER e Linguagem de Máquina

DAVID C. ALEXANDER



EDITORA CAMPUS

Títulos de Computação da Editora Campus

ADMINISTRAÇÃO DE SISTEMAS DE INFORMAÇÃO – P. Ein-Dor e E. Segev

BASIC BÁSICO (4ª Edição) – J.C. Pereira Filho

BASIC PARA MICROS PESSOAIS (2ª Edição) – J. C. Pereira Filho et al.

BRINCANDO COM O COMPUTADOR – J.A. Moreira

COBOL PARA ESTUDANTES (4ª Edição) – A. Parkin

COMO LIDAR COM O COMPUTADOR – H.C. Lucas Jr.

COMPUTADORES BRASILEIROS: INDÚSTRIA, TECNOLOGIA E DEPENDÊNCIA – P.B. Tigre

COMPUTADORES PARA USUÁRIOS – J.C. Pereira Filho
Vol. I – Aplicação e Uso dos Computadores
Vol. II – Equipamentos e Sistemas de Computação
Vol. III – Programas e Programação de Computadores
Vol. IV – Seleção de Sistemas de Computação

A CONSTRUÇÃO DE UM COMPILADOR – V.W. Setzer e I.S.H. Melo

CRIANÇA TAMBÉM FAZ PROGRAMAS – J.A. Moreira

DOCUMENTAÇÃO DE SOFTWARE – J.D. Lomax

ESPECIFICAÇÃO DE SISTEMAS – S.J. Waters

ESTRUTURAS DE DADOS – P.S. Veloso et al.

FUNDAMENTOS DE PROCESSAMENTO DE DADOS – W.T. Price.

GERÊNCIA DE BASES DE DADOS PARA MICROCOMPUTADORES – E.G. Brooner

GRAFOS E ALGORITMOS COMPUTACIONAIS – J.L. Szwarcfiter

GUIA DE LINGUAGENS DE COMPUTADORES – H.L. Helms Jr.

GUIA PARA PROGRAMADORES (2ª Edição) – M. Bohl.

IMPLANTAÇÃO DE MICROS E MINICOMPUTADORES COMERCIAIS – P.A. Knight.

INTRODUÇÃO À CIÊNCIA DA COMPUTAÇÃO COM WATFIV E FORTRAN – S.E.R. Carvalho

INTRODUÇÃO À PROGRAMAÇÃO COM PASCAL (2ª Edição) – S.E.R. Carvalho

INTRODUÇÃO À PROGRAMAÇÃO DE COMPUTADORES (2ª Edição) – H.V.R. Corrêa et al.

INTRODUÇÃO À PROGRAMAÇÃO FORTRAN – J.C. Pereira Filho

INTRODUÇÃO À SEGURANÇA DO COMPUTADOR – M.B. Wood

INTRODUÇÃO AO PROCESSAMENTO DE TEXTOS – G.L. Simons

INTRODUÇÃO AO VISICALC – E.A. Garbin

JCL SISTEMA/370 (2ª Edição) – G.D. Brown

LCP-LÓGICA DE CONSTRUÇÃO DE PROGRAMAS – J.D. Warnier

LINGUAGEM DE PROGRAMAÇÃO ALGOL – L.M. Segre

A LINGUAGEM PASCAL – V.L. Strube de Lima

LCS – LÓGICA DE CONSTRUÇÃO DE SISTEMAS – J. D. Warnier

MANUAL DE COBOL ESTRUTURADO – D.D. McCracken

MICROCOMPUTADORES PARA APLICAÇÕES COMERCIAIS – W. Barden Jr.

Programação em ASSEMBLER e Linguagem de Máquina

DAVID C. ALEXANDER

2ª EDIÇÃO

Tradução

Josemar Gomes Mendes
Engenheiro Eletrônico

EDITORA CAMPUS LTDA.
Rio de Janeiro

Do original:
Machine & Assembly Language Programming

1985, 2ª Edição

Capa
Otavio Studart

Diagramação, composição, paginação e revisão
Editora Campus Ltda.
Rua Barão de Itapagipe 55 Rio Comprido
Tel.: (021) 284 8443
20261 Rio de Janeiro RJ Brasil
Endereço telegráfico: CAMPUSRIO

ISBN 85-7001-237-3
(Edição original: ISBN 0-8306-1389-7 Tab Books Inc., USA)

Ficha Catalográfica
CIP-Brasil. Catalogação-na-fonte
Sindicato Nacional dos Editores de Livros, RJ.

Alexander, David C.
A367p Programação em ASSEMBLER e linguagem de máquina / David
2.ed. C. Alexander ; tradução de Josemar Gomes Mendes. – 2ª ed. – Rio de
Janeiro : Campus, 1985.

Tradução de: Machine & Assembly language programming.
Apêndices
Glossário
ISBN 85-7001-237-3

1. ASSEMBLER (Linguagem de programação para computado-
res) I. Mendes, Josemar Gomes II. Título.

CDD – 001.6424
85-0059 CDU – 800.92ASSEMBLER

Programação em ASSEMBLER e Linguagem de Máquina

Este livro é dedicado a

Minha querida esposa Andrea que, embora adoentada, dedicou seu tempo e paciência para me ajudar a escrevê-lo

Meus filhos – Annalece, Mike, Michelle, Billy e Frank – por sua ajuda e paciência

Dave, Fran, Steve, Ray, Don e Isla que leram o livro e me encorajaram com suas opiniões, que constituíram uma base para o desenvolvimento deste curso.

O membro da minha família, que dedicou uma semana do seu tempo para corrigir a minha terrível pontuação e ortografia (espero não ter acrescentado mais nada depois) e deseja permanecer incógnito

A todos, o meu MUITO OBRIGADO!

Sumário

Introdução, 3

1 Sistemas Numéricos, 13

2 Por que Usar Linguagem de Máquina e Assembler?, 23

3 Registradores, Pares de Registradores e Endereços, 25

4 Formato Assembler, 29

5 Códigos de Operação, 34

6 O Grupo de Loads de 8 Bits, 38

7 O Grupo de Loads de 16 Bits, 46

8 Instruções PUSH e POP, 52

9 Trocas, Transferências em Bloco e Grupo de Pesquisa, 58

10 O Grupo Aritmético de 8 Bits, 69

11 Aritmética de Âmbito Geral e o Grupo de Controle da UCP, 79

12 O Grupo Aritmético de 16 Bits, 84

13 O Grupo de Rotação (ROTATE) e de Deslocamento (SHIFT), 89

14 Ligar e Desligar Bits e o Grupo de Instruções de Teste, 96

15 O Grupo de Salto (JUMP), 101

16 O Grupo de Chamada (CALL) e Retorno (RETURN), 107

17 Pseudocódigos de Operação, 111

18 Programas de Exemplo, 118

19 Do BASIC para a Linguagem de Máquina: O Programa das Formas, 156

Apêndice A Códigos, 164
Apêndice B Sufixos JR, 166
Apêndice C Lista Numérica dos Códigos de Máquina, 168
Apêndice D Lista Alfabética dos Códigos de Máquina, 174
Apêndice E Palavras Reservadas, 181
Glossário, 182
Índice Analítico, 184

Introdução

Este livro foi escrito por um programador que tentou aprender a linguagem Assembler e de máquina, da maneira mais difícil, isto é, por conta própria, sem nenhuma ajuda.

Não salte os capítulos, a menos que você deseje ficar totalmente perdido e confuso. Este é um método de aprendizado passo a passo e *deve* ser seguido como tal. Todo o conteúdo está interligado entre si. Se não entender um capítulo, leia-o de novo até conseguir compreendê-lo.

A linguagem Assembler e a linguagem de máquina são como qualquer outra linguagem. As linguagens de computadores podem ser comparadas à linguagem falada. Todas as linguagens faladas realizam apenas uma coisa: elas passam um pensamento ou informação de uma pessoa para outra, independentemente do fato de se estar falando inglês, espanhol, alemão, russo, francês, ou qualquer outra língua. Por exemplo: *buenos dias, bonjour* e *hello* fazem a mesma coisa: todos expressam um cumprimento, embora sejam falados em três diferentes línguas.

Programação é programação, independente de que linguagem se use. Um programa faz uma só coisa: transfere a informação de um lugar para outro, seja ele em BASIC, FORTRAN, COBOL, PL/I, Assembler ou linguagem de máquina.

Este livro pressupõe que você não tem experiência prévia em programação e lhe ensinará a linguagem Assembler e de máquina. Se já tiver tido experiências com programação, o livro deverá tornar as linguagens Assembler e de máquina bem mais fáceis para você. A maioria dos proprietários de microcomputadores inicia na programação usando o BASIC. Se já estudou BASIC, saberá o quão fácil é esta linguagem.

Se lhe surgirem dúvidas ou perguntas com relação a este meu livro, pode escrever para o seguinte endereço: Dave Alexander, 220 West Second Street, Erie, PA 16507. Por favor, envie um envelope selado e endereçado.

CAPÍTULO 1

Sistemas Numéricos

Existem três tipos básicos de sistemas numéricos utilizados por programadores de computadores. Eles são: o decimal, o binário e o hexadecimal. O *sistema decimal* é baseado em potências de 10 e é aquele que você tem usado em toda a sua vida: 0 1 2 3 . . . 9

O *sistema binário* significa, simplesmente, potências de 2. Este sistema possui apenas dois algarismos: o zero e o um.

O computador utiliza o sistema binário, ou alguma versão desse sistema, em suas operações. Tal representação ocorre na memória, onde cada ponto se apresenta eletricamente positivo (um) ou negativo (zero); mais simplesmente, os diminutos dispositivos eletrônicos de chaveamento estão ligados ou desligados.

Como um programador de linguagem Assembler, você não escreverá programas em binário, mas é importante saber como funciona o sistema binário.

Vamos tomar um número em binário e convertê-lo para um número decimal. Seja o número 11001. Não, não se trata de onze mil e um. Antes que eu lhe diga o que realmente ele é, vamos examinar rapidamente o sistema decimal.

SISTEMA DECIMAL

Um quadro simples para se achar potências de 10 é mostrado na Tabela 1-1. Lembre-se de que qualquer número elevado a 0 é igual a 1.

Tabela 1-1. *Potências de 10.*

Número	Potência	Valor
10	0	1
10	1	10
10	2	100
10	3	1.000

10	4	10.000
10	5	100.000
10	6	1.000.000
10	7	10.000.000
10	8	100.000.000
10	9	1.000.000.000
10	10	10.000.000.000

Primeiramente, vamos considerar o número 25. Isto significa dizer 10 elevado a zero, ou seja, 1, multiplicado por 5, igual a 5, e 10 elevado à primeira potência, ou seja, 10 multiplicado por 2, igual a 20.

2 x 10 à primeira potência (segunda coluna) = 2 x 10 = 20
5 x 10 elevado a zero (primeira coluna) = 5 x 1 = 5

Some os dois resultados e obterá 25 como resposta.

```
 20   (coluna 2)
+5    (coluna 1)
------
 25
```

Vejamos isto em um tipo diferente de diagrama:

Dígito Número	2	1	
Número Decimal	2	5	
Multiplicado por	$(10)^1$	$(10)^0$	
Respostas	20	5	= 25

SISTEMA BINÁRIO

Voltemos à nossa conversão binária para decimal do número 11001. A tabela 1-2 vai nos ajudar a converter o número ao sistema decimal.

Primeiramente, conte o número de dígitos do número binário e subtraia 1. Isto lhe dará a potência mais elevada a ser usada. Existem 5 dígitos binários no número 11001, o que nos dá 4, como maior potência de 2 a ser usada.

A seguir, verifique o 4 na coluna de potência de 2 na tabela 1-2 e escreva o valor decimal correspondente na coluna de resposta. Isto mesmo, trata-se do número 16.

Em seqüência, olhe o número imediatamente inferior na tabela, o 3, escrevendo o correspondente decimal para ele. Fazendo isto, você encontra a resposta 8.

Note, agora, que o dígito binário que corresponde à potência 2 é 0 e, portanto, o resultado a somar é também 0.

O mesmo ocorre com a potência imediatamente inferior, ou seja, o 1, o resultado também é 0.

Finalmente, na potência de zero temos o resultado de 1, que deve ser anotado.

Temos então a soma das respostas: (16 + 8 + 0 + 0 + 1) = 25

Vamos ver isto de novo, passo a passo:

Passo 1. binário 11001 5 dígitos – 1 = 4 (a mais alta potência de 2 a ser usada).
Passo 2. da tabela 1-2, o valor decimal da quarta potência de 2 é 16.
Passo 3. da tabela 1-2, o valor decimal da terceira potência é 8.
Passo 4. da tabela 1-2, o valor decimal da segunda potência não é considerado, pois o algarismo correspondente no número é 0.
Passo 5. da tabela 1-2, o valor decimal da primeira potência não é considerado, pois o algarismo correspondente no número é 0.
Passo 6. da tabela 1-2, o valor decimal da potência zero é 1.
Passo 7. some os valores decimais (16 + 8 + 0 + 0 + 1) = 25

Eis, a seguir, um exemplo de conversão do sistema decimal para o binário:

Passo 1. Encontre o maior número decimal na tabela, que não exceda o número a converter.
Passo 2. Divida o seu número decimal por esse número. Se o resultado (parte inteira) não for 1, então você terá usado o número errado da tabela.

Tabela 1-2. *Conversão Binário para Decimal.*

Potência de 2	Valor Decimal
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024

Passo 3. Obtenha o resto da divisão anterior.
Passo 4. Encontre o próximo valor da tabela imediatamente inferior ao valor achado em 3 (a cada valor encontrado na tabela corresponde um algarismo 1 do número binário que está sendo formado, e a cada valor saltado, na procura do próximo inferior, corresponde um algarismo 0 no número binário).
Passo 5. Divida o resto achado no passo 3 pelo valor encontrado no passo 4 e repita os passos 3 – 4 – 5 até ter passado por todas as potências de 2 na tabela.

Experimente agora você mesmo encontrar o valor binário do decimal 25, desta forma. Refira-se então à tabela 1-3.

Aposto que você deve estar surpreso por verificar como a conversão é realmente fácil. Antes de prosseguirmos, vejamos alguns problemas de exemplo.

Converta os números binários para decimais.

Problemas de Conversão de Binário para Decimal

Nº do problema	Nº Binário	Nº Decimal
1	11101	_____
2	11000	_____
3	111011	_____
4	11111111	_____
5	00001111	_____

Verifique suas respostas contra aquelas fornecidas no final do capítulo. Se as respostas coincidirem, então prossiga para o próximo exercício. Se não, reexamine o seu trabalho e encontre os erros. As respostas fornecidas estão corretas. Se, durante a verificação, você não conseguir chegar à resposta correta, então releia este capítulo. *Não vá à frente até que você domine perfeitamente esta operação*!

Problemas de Conversão de Decimal para Binário

Nº do problema	Número Decimal	Nº Binário
6	256	_____
7	376	_____
8	1087	_____
9	65783	_____
10	15360	_____

Verifique suas respostas contra as fornecidas no final do capítulo. As mesmas considerações feitas anteriormente, são válidas também aqui. Não prossiga, a menos que você esteja bem firme neste tipo de operação!

Tabela 1-3. *Divisão Binária*

Etapa Nº	Valor Decimal	Número Decimal	Número Binário	Operação Aritmética
2	16	25	1	Divida por 16
		16		Subtraia

3	8	9	1	Divida por 8
		8		Subtraia

4	4	1	0	Não pode dividir 1 por 4
		0		Subtraia

5	2	1	0	Não pode dividir 1 por 2
		0		Subtraia

Etapa Nº	Valor Decimal	Número Decimal	Número Binário	Operação Aritmética
6	1	1	1	Divida 1 por 1
		1		Subtraia

		0		

7 Leia o valor binário de cima para baixo na coluna número binário

SISTEMA HEXADECIMAL

Antes de prosseguir, é possível que você deseje reler a parte correspondente ao sistema binário. Se o binário está bem claro para você, então está pronto para ver o sistema hexadecimal, ou, simplesmente, hexa, como às vezes o chamamos.

Esta é a parte mais importante do livro até aqui. Isto se deve a que, nas linguagens Assembler e de máquina, você usará constantemente o hexadecimal. Assegure-se que irá fixar bem esta seção.

Hexadecimal, ou hexa, significa de base 16. O sistema possui os algarismos de 0 a 15, o que pode parecer simples demais, porém há um detalhe.

Como só possuímos algarismos numéricos de 0 a 9, o sistema hexa lança mão das letras do alfabeto a partir do dígito 9. Vejamos como o hexadecimal se compara com o decimal:

Decimal:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Hexa:	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

A tabela 1-4 serve como uma ajuda para a conversão decimal para hexa.

Agora, vamos trabalhar com alguns problemas sobre números hexadecimais. Primeiramente, transformemos um número decimal para hexa. Vamos usar o número 12.345 e convertê-lo para hexadecimal. Observe a tabela 1-4. Note que existem 4 colunas, numeradas, da esquerda para a direita, de 4 a 1. Cada coluna se abre em mais duas, com os números decimais à direita e os hexadecimais à esquerda.

Tabela 1-4. *Conversão Decimal para Hexadecimal*

Coluna 4		3		2		1	
Hex	Dec	Hex	Dec	Hex	Dec	Hex	Dec
0	0	0	0	0	0	0	0
1	4.096	1	256	1	16	1	1
2	8.192	2	512	2	32	2	2
3	12.288	3	768	3	48	3	3
4	16.384	4	1.024	4	64	4	4
5	20.480	5	1.280	5	80	5	5
6	24.576	6	1.536	6	96	6	6
7	28.672	7	1.792	7	112	7	7
8	32.768	8	2.048	8	128	8	8
9	36.864	9	2.304	9	144	9	9

Coluna 4		3		2		1	
Hex	Dec	Hex	Dec	Hex	Dec	Hex	Dec
A	40.960	A	2.560	A	160	A	10
B	45.056	B	2.816	B	176	B	11
C	49.152	C	3.072	C	192	C	12
D	53.248	D	3.328	D	208	D	13
E	57.344	E	3.584	E	224	E	14
F	61.440	F	3.840	F	240	F	15

Para converter o número 12.345, ou qualquer outro decimal, para hexa, siga os passos adiante:

1. Olhe a coluna 4 e encontre o número que mais se aproxima de 12.345, sem, contudo, excedê-lo. Escreva então o número hexadecimal a ele correspondente. Subtraia o número decimal de 12.345. Ao fazer este procedimento, descobrimos que o número decimal da coluna 4 é 12.288. Escreva o número hexadecimal a ele correspondente (3). Agora, subtraindo, temos: 12.345 – 12.288 = 57.

2. A seguir, procure na coluna 3 o número 57. O maior número, na coluna 3, que podemos subtrair de 57 é 0. Assim, escrevemos o número hexadecimal 0 e subtraímos: 57 – 0 = 57. O nosso número hexadecimal, até agora, está como 30.

3. Vamos à coluna 2 e procuremos o número mais próximo a 57 que não o exceda. Encontramos o 48. O número hexadecimal é 3. Escrevemos o 3, o que transforma o nosso número hexa em 303. Agora, subtraímos: 57 – 48 = 9.

4. O último passo é simples. Encontre o 9 na coluna 1, no lado decimal. O hexa correspondente é 9, que, então, é escrito, resultando o número hexadecimal convertido 3039.

Mas, espere um momento! Este é um número hexadecimal e, uma vez que ele não apresenta letras em nenhum de seus dígitos, parece-se com um número decimal. Temos que colocar uma identificação no número, para que não o confundamos com um número decimal. Para tal, simplesmente adicionamos a letra H. O H significa hexa e, já que o maior dígito hexadecimal é F, não há meios de confundirmos o H como um dígito. Agora o nosso número tem a forma 3039H.

Façamos agora uma conversão de hexadecimal para decimal, usando para exemplo o número 3C0FH.

Para transformar 3C0FH em decimal, iniciamos localizando o 3 na coluna 4, lado hexa. Escrevemos então o número decimal a ele correspondente (12.288). A seguir vamos à coluna 3 e, achando o C, escrevemos o decimal correspondente. Fazemos a mesma coisa com o restante dos dígitos, de forma que a nossa lista de números é: 12.288, 3.072, 0 e 15. Agora, somemos estes números. 12.288 + 3.072 + 0 + 15 = 15.375. Logo, 3C0FH = 15.375. É só isso.

Tente alguns números você mesmo. Você deve observar as disposições dos números decimais e hexadecimais e compará-los com o arranjo do sistema binário. Logo verá que os números hexadecimais ascendem em potências de 16. Se você não obtiver as respostas corretas para os problemas a seguir, releia esta seção.

Problemas de Conversão de Hexadecimal para Decimal

Nº do problema	Nº Hexa	Nº Decimal
1	3C00H	_____
2	7FFFH	_____
3	01C9H	_____
4	0F0H	_____
5	0FH	_____

Problemas de Conversão de Decimal para Hexa

Nº do problema	Nº Decimal	Nº Hexadecimal
6	32767	_____
7	16360	_____
8	0	_____
9	10000	_____
10	20480	_____

Compare suas respostas com as apresentadas no final do capítulo.

QUESTÕES

1. Indique que tipo de sistema numérico cada número abaixo está representando:

   A. 110001001 _____
   B. 3000H _____
   C. 3000 _____

2. Binário significa de base _____
3. Hexa significa de base _____
4. Decimal significa de base _____
5. Converta os números a seguir para os sistemas numéricos restantes:

	Decimal	Hexa	Binário
A.	121	_____	_____
B.	_____	39H	_____
C.	_____	_____	10110
D.	1000	_____	_____
E.	_____	FFH	_____

As respostas são dadas no final do capítulo.

RESPOSTAS

Conversões Binário para Decimal e Vice-versa.

Nº do problema	Resposta	Procedimento
1	31	11101 5 dígitos, começar na quarta potência (16) 16 + 8 + 4 + 2 + 1 = 31.
2	24	11000 5 dígitos, começar na quarta potência (16) 16 + 8 + 0 + 0 + 0 = 24.
3	59	111011 6 dígitos, começar na quinta potência (32) 32 + 16 + 8 + 0 + 2 + 1 = 59.
4	255	11111111 8 dígitos, começar na sétima potência (128) 128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 = 255.
5	15	00001111 – detalhe – apenas 4 dígitos efetivos. Os zeros à esquerda não são considerados, logo há apenas 4 dígitos e, assim, começamos na terceira potência (8) 8 + 4 + 2 + 1 = 15.
6		100000000
7		101111000
8		10000111111
9		1100011000110111
10		11110000000000

Conversões Hexadecimal para Decimal e Vice-versa

Nº do problema	Resposta	
1	15360	
2	32767	
3	457	
4	240	(Zeros à esquerda não contam)
5	15	
6	7FFFH	
7	3FF0H	
8	0000H	Ou 0H
9	2710H	
10	5000	

RESPOSTAS ÀS QUESTÕES

Questão Nº	Resposta	
1A.	Binário	
1B.	Hexadecimal	
1C.	Decimal	
2.	2	
3.	16	
4.	10	
5A.	79H	1111011
5B.	57	111011
5C.	22	16H
5D.	2E8H	1111101000
5E.	255	11111111

CAPÍTULO 2

Por que Usar Linguagem de Máquina e Assembler?

Você pode estar imaginando se valerá realmente a pena se preocupar em aprender a usar linguagem de máquina e Assembler. Deixe-me adiantar-lhe alguns fatos sobre estas linguagens, que você irá descobrir, à medida que for progredindo.

A linguagem de máquina dirige-se diretamente à UCP (Unidade Central de Processamento). Tal não faz o BASIC. Em conseqüência, leva-se cerca de 350 vezes mais tempo para se executar um programa, ou um comando, em BASIC do que em linguagem de máquina. Um bom exemplo disso é o "loop" de atraso de 1 segundo, que você, possivelmente, já deve ter usado em BASIC : FOR X = 1 TO 500 : NEXT. O mesmo "loop", em linguagem de máquina, não duraria um segundo. Em linguagem de máquina, o computador pode contar de 1 a 84.000 em um segundo. Vemos que 84.000/500 = 168, mas existe, em adição, o tempo gasto com uma sobrecarga maior das operações. Necessita-se de mais instruções em linguagem de máquina ou Assembler do que em BASIC, porém gasta-se muito menos tempo com trabalhos adicionais além de necessitar menor memória para se escrever os programas.

A principal diferença entre o BASIC e a linguagem de máquina é que um programa escrito em BASIC deve ser primeiramente convertido, uma instrução por vez, pelo computador, para linguagem de máquina, a fim de que o computador possa compreendê-lo. Em linguagem de máquina, estamos nos dirigindo diretamente ao computador, em sua própria linguagem, donde nenhuma conversão se torna necessária.

Uma outra característica da linguagem de máquina é que ela assegura que o programa não pode ser listado diretamente do teclado, a menos que você possua um programa como o T-BUG ou um Montador/Editor Assembler. Isto oferece uma maior segurança para arquivos confidenciais ou de negócios. As listagens também podem ser totalmente bloqueadas através de instruções especiais de programa, que impedem a carga de um programa para listar as instruções. Isto pode dificultar bastante as tentativas de cópias não-autorizadas de programas.

Além destes aspectos, existe a conveniência de se poder localizar sub-rotinas ou subprogramas em várias áreas do programa principal e, então, poder ir e voltar a essas rotinas inúmeras vezes. É possível que você já tenha visto isto feito com BASIC, no sentido de se tentar tornar mais difíceis as alterações do programa. Isto funciona ainda melhor com linguagem de máquina. Conforme você verá mais tarde, ou talvez já tenha visto, os programas em linguagem de máquina e em Assembler são difíceis de acompanhar, a menos que você realmente entenda da linguagem. Por esta razão, você tem uma vantagem, com relação à segurança, sobre as pessoas que não tenham um bom conhecimento da linguagem.

No próximo capítulo, começaremos a estudar as instruções e comandos de programação. Se, a qualquer momento, algo não estiver claro para você, releia o trecho até entendê-lo perfeitamente.

CAPÍTULO 3

Registradores, Pares de Registradores e Endereços

Com a linguagem de máquina, nós estaremos movimentando elementos de informação, de uma localização para outra, dentro do computador. Este processo endereça diretamente o computador e é cerca de 300 vezes mais rápido do que o processo equivalente em linguagem BASIC.

Vamos parar por um momento aqui e fazer uma visita à "agência dos correios" do TRS-80.

REGISTRADORES

À medida que entramos, nos deparamos com uma parede com várias caixas coletoras. Chamemos a estas caixas *registradores*. Obviamente, existirão muitas outras agências no TRS-80. Suponha que você se localiza em uma dessas. Este será o seu endereço ou localização. Da mesma forma eu também terei a minha. Assim sendo, na sua agência haverá também uma grande abertura para se colocar correspondência. Chamemo-la de um registrador. Quando você tem uma carta, ou alguma informação para mim, você a coloca na abertura coletora e o correio (a UCP) a apanha de sua localização e a entrega em meu endereço, colocando-a em meu registrador.

Acabamos de descrever a ação que transcorre dentro do computador.

ENDEREÇOS

Expliquemos o que vêm a ser *endereços*. Trata-se das localizações ou posições de memória no computador. Se você possui uma máquina de 16K, existem 32.768 posições de memória nela. Elas são numeradas de 000H a 7FFFH. Cerca de metade dessas posições é usada pelo computador em forma de ROM (Read-only memory – memória que só pode ser lida) para a guarda de rotinas e programas de utilidade e controle do sistema, não sendo acessíveis à programação. Estas localizações estendem-se, normalmente, de 0000H a 42E9H. Verifique o mapa da memória no seu manual do proprietário e identifique estas posições. O restante é a área de memória RAM (memória de

acesso aleatório disponível para programação do usuário). Seria uma boa idéia fazer uma cópia do mapa de memória e colocá-lo próximo ao seu terminal. Você irá constantemente referir-se a ele.

TIPOS DE REGISTRADORES

Agora, vejamos os registradores. Existem oito registradores. São eles denominados A, B, C, D, E, F, H e L. Estes registradores também possuem registradores *preparados,* correspondentes a eles. São chamados A', B', C', D', E', F', H' e L'. Eles serão discutidos mais adiante. O importante, no momento, é saber que eles existem. Existem ainda dois outros registradores, X e Y, com os quais não nos preocuparemos por enquanto. Eles também possuem correspondentes registradores prèparados X' e Y'.

Cada registrador isolado pode conter um valor até 255, ou seja, FFH. Para se usar um valor superior a 255, devem-se utilizar dois registradores em combinação, formando-se, assim, um *par de registradores.* Os registradores podem ser emparelhados da seguinte maneira: AF, BC, DE e HL. Estes pares podem também armazenar valores inferiores a 255. O valor máximo que pode ser armazenado em um par de registradores é 65.535, ou FFFFH.

Além dos quatro pares de registradores, existem dois *pares especiais de registradores,* IX e IY. Há, ainda, mais um par de registradores, chamado par de registradores sp ou "*stack pointer*" (ponteiro da pilha). Ele aponta para o topo da pilha ou para o próximo endereço onde a informação pode ser armazenada. Discutiremos o "stack pointer" mais tarde.

QUESTÕES

1. . . . . são localizações de memória no computador.
2. Os endereços de 0000H a 42E9H constituem a parte da memória conhecida como . . . . . .
3. Cite os oito registradores.
4. Cite os dois registradores especiais.
5. Quais são os registradores preparados?
6. Liste os pares de registradores.
7. Mencione os pares de registradores especiais
8. Qual o maior valor que pode ser armazenado em um registrador?
9. Qual o máximo valor que pode ser armazenado em um par de registradores?
10. O que é o par de registradores SP?

RESPOSTAS

1. Endereços
2. ROM
3. A, B, C, D, E, F, H e L
4. Y e X
5. A', B', C', D', E', F', H' e L'
6. AF, BC, DE e HL
7. IX e IY
8. 255 (FFH)
9. 65.535 (FFFFH)
10. "stack pointer"

CAPÍTULO 4

Formato Assembler

Uma vez que estamos apenas começando, é conveniente vermos o que estamos fazendo. Para tal, usaremos o Montador/Editor Assembler para escrever nossos programas.

O Montador/Editor é um programa que nos permite escrever programas em linguagem Assembler e, então, converte-os para linguagem de máquina. Quando iniciarmos, haverá cinco colunas, que serão do nosso interesse, no formato da linguagem Assembler. Elas estão listadas a seguir.

Coluna 1	**2**	**3**	**4**	**5**
Número da Linha	**Rótulo**	**Código de Op.**	**Operando(s)**	**Comentário**

Depois de *montarmos* (submeter ao processamento pelo Montador) o programa, haverá duas colunas adicionais à esquerda das cinco iniciais. A lista resultante será da forma:

Coluna 1	**2**	**3**	**4**	**5**	**6**	**7**
Posição de Memória	**Código de máquina**	**Número da Linha**	**Rótulo**	**Código de Op.**	**Operando**	**Comentário**

A diferença entre a linguagem Assembler e a linguagem de máquina é simples. Em linguagem Assembler nós entramos com as instruções de Assembler. O montador, então, traduz estas instruções para código de máquina. Se você está familiarizado com computadores de grande porte, você já terá ouvido falar que eles utilizam um componente de software chamado *compilador*. Os compiladores traduzem um programa inteiro, de uma só vez, de suas instruções originais para linguagem de máquina. Os *interpretadores* (comumente usados para o BASIC) traduzem o programa para linguagem de máquina, uma instrução por vez. Os compiladores são, geralmente, muito mais rápidos. O Montador Assembler trabalha da mesma forma que um compilador. (Os termos se distinguem apenas por trabalharem os compiladores sobre programas em linguagens de alto nível, como o COBOL, enquanto que o Montador opera sobre programas escritos em Assembler, que é uma linguagem de baixo nível, isto é, próxima à linguagem de máquina.

Em linguagem de máquina, você não entra com instruções codificadas em cada linha, como na linguagem Assembler. Em vez disso, você monta nas posições de memória os códigos de máquina (0's e 1's) que você deseja. Você pode gastar um tempo enorme procurando pelos códigos apropriados ao longo de um programa. Por esta razão, até que você se torne realmente "bom" no assunto, vamos usar a linguagem Assembler.

À medida que você for aprendendo a linguagem Assembler e de máquina, sugiro que vá prestando atenção aos códigos de máquina na segunda coluna da montagem de seus programas. Isto lhe ajudará a fixar alguns códigos e a aprender como se escreve um programa em linguagem de máquina.

Se você possui um programa T-BUG e não tem um Montador/Editor Assembler, eu recomendo que adquira um. Isto lhe facilitará escrever programas, pois os códigos de máquina estão listados na parte posterior do manual do programa Montador/Editor.

Vamos, agora, examinar as colunas a que já nos referimos. A primeira das sete colunas é denominada Posição (ou localização) de Memória. Ela nos diz onde a instrução está armazenada (seu endereço). Este número é sempre dado em hexadecimal.

A próxima coluna é o Código de Máquina. Este é o código traduzido que instrui o computador sobre o que fazer. Por exemplo: CDC901. Veremos, mais tarde, o que significa esta instrução mas, no momento, estamos só interessados em ver com que se parece um código de máquina.

A terceira coluna é o Número da Linha. Assim como no BASIC, cada linha do programa original tem um número. Os números das linhas são úteis para se localizar áreas do programa, quando você tiver escrito alguma seção específica. Se você usa sempre certos números de linha para certas funções, quando necessitar aumentar o programa ou encontrar aquela área rapidamente, você pode mandar listar os números correspondentes àquelas linhas. Um bom hábito é reservar sempre certas faixas de números para certas funções, como por exemplo:

100-200	IMPRIMIR O MENU
200-300	PESQUISAR A FUNÇÃO SELECIONADA
300-400	ÁREA DE ENVIO DE MENSAGENS

A quarta coluna é a coluna de Rótulo. Esta coluna define um título para uma certa função ou área do programa a qual é referenciada em outra parte do mesmo. *Os rótulos devem começar com uma letra* e podem conter uma combinação qualquer de caracteres alfanuméricos (letras e números) de até 6 caracteres. O rótulo não deve conter o símbolo $ e os mnemônicos ADD, SUB, INC, EQU e DEC. Estes rótulos são encontrados no manual do Montador ou no apêndice deste livro.

A coluna número cinco refere-se ao código de operação. Sua função é instruir ao computador que operação ele deve realizar.

Existem vários códigos de operação e podemos citar dentre eles o LOAD, ADD, SUBTRACT, INCREMENT, DECREMENT, EXCHANGE, CALL e

JUMP. Discutiremos em detalhe estes códigos mais tarde. Desta forma, não se preocupe em tentar entendê-los agora, pois foram mencionados, aqui, apenas para dar uma idéia do que são.

Existe ainda um outro tipo de código de operação, o chamado Pseudo-código. Trata-se de instruções especiais que orientam o computador para a execução de procedimentos especiais. Mais tarde examinaremos as funções desses pseudocódigos. Por enquanto, é suficiente saber que existem nove pseudocódigos: ORG, EQU, DEFL, END, DEFBN, DEFB S, DEFW, DEFS e DEFM.

A sexta coluna é a coluna do(s) operando(s). O operando é o registrador, ou o par de registradores ou a posição de memória sobre a qual a operação deve ser realizada. Por exemplo, se eu lhe digo para somar, isto é o código de operação. Você ainda não sabe o que deve ser somado a que. Mas se eu lhe digo para somar o conteúdo do registrador "A" ao conteúdo do par de registradores"HL", aí, então, você sabe o que fazer. Aqui, o registrador A e o par HL são os operandos.

A sétima coluna é a coluna dos comentários. O comentário serve para explicar o que estamos pretendendo realizar com aquela instrução. Todas as entradas nesta coluna devem ser precedidas por um *ponto e vírgula.* O ponto e vírgula separa o comentário dos operandos e informa ao computador para ignorar o texto adiante, que é o comentário.

Você pode perguntar "Por que precisaríamos disso?" Deixe-me narrar-lhe uma história verdadeira que sucedeu comigo.

Eu vinha trabalhando em um programa, desenvolvendo algumas rotinas para controle de estoque. Depois de uma hora colocando as linhas de comentários, eu me cansei de datilografar aquela inutilidade e, desta forma, parei de escrevê-los. Três horas mais tarde, já na linha 4.000, contando de 10 em 10, eu precisei encontrar uma sub-rotina que eu havia usado em alguma linha anterior. Já que eu não dispunha mais de comentários para me guiar, gastei, simplesmente, duas horas para localizar o que eu desejava. Depois disso eu uso religiosamente os comentários, como meio de documentação do trabalho. É sempre possível removê-los, após o término do desenvolvimento do programa, mas dispor dessa informação pode poupar um bocado de tempo e aborrecimento. (Ressalte-se ainda a conveniência dos comentários nos casos em que a programação deva ou possa vir a ser analisada e/ou modificada por outra pessoa.)

QUESTÕES

1. . . . . são grupos de letras e números que podem ser usados como rótulo.
2. A localização de memória é sempre mostrada em . . . . . .
3. O . . . . . diz ao computador o que fazer.
4. . . . . são usados em todas as linhas e podem ser usados para dividir o programa em seções.
5. . . . . são usados para identificar uma sub-rotina.
6. O . . . . . diz ao computador que tipo de função ele deve realizar.
7. O . . . . . é um código de operação especial.
8. O . . . . . informa sobre o que deve ser realizada a operação.
9. . . . . .devem ser usados em cada linha, para lhe esclarecer o que você está procurando fazer naquela instrução.

RESPOSTAS

1. caracteres alfanuméricos
2. hexadecimal
3. código de máquina
4. números de linha
5. rótulos
6. código de operação
7. pseudocódigo
8. operando
9. comentários

CAPÍTULO 5

Códigos de Operação

O último capítulo abordou rapidamente a função dos códigos de operação e o que vêm a ser eles. Agora, vamos aprofundar a discussão sobre os vários códigos e o que eles executam. É bom frisar que este é um livro para principiantes e, sendo assim, não desceremos a todos os detalhes possíveis. Pretendo, basicamente, transmitir ao leitor um conhecimento fundamental do que vem a ser cada coisa. Você poderá aprimorar sua educação valendo-se de outros livros disponíveis no mercado.

Talvez seja conveniente aprender um pouco sobre a memória do computador. Para facilitar a explicação, você poderia idealizar uma pilha de caixas. As caixas representariam as posições de memória. Haverá uma caixa para cada posição. A pilha de caixas representaria a *pilha da memória*, ou "stack". Uma pilha nada mais é do que um grupo de células de memória *consideradas* enfileiradas ou empilhadas uma sobre a outra.

O conceito de pilha, ou fila, usado no tratamento das posições de memória nos computadores segue uma disciplina de operação LIFO (last in first out – último a entrar, primeiro a sair). Imagine um homem subindo uma pilha de caixas. À medida que ele sobe, vai colocando uma folha de papel em cada uma das caixas. Ao descer, ele deverá recolher as folhas de papel de cada caixa. Ao fazer isto, o último papel que ele pôs nas caixas será o primeiro a ser retirado. O primeiro que foi colocado será o último que ele retirará.

A pilha é usualmente definida como a memória disponível, acima da última instrução do programa. Existem, todavia, duas exceções.

A primeira exceção é que um programa pode conter o que se denomina uma *pilha interna*. Esta é uma área, dentro do programa, que é usada para armazenar, temporariamente, informações. A segunda exceção ocorre quando um programa está localizado no meio da memória e você pode, então, usar o espaço disponível abaixo do programa, como uma pilha. Esta área é chamada de "buffer", que vem a ser uma memória temporária.

A partir deste ponto, você começará a aprender a respeito dos diferentes códigos de operação e um pouco sobre de que forma são eles usados.

Agora, é importante que você se familiarize com as seguintes abreviaturas:

N	Número. Qualquer decimal ou hexadecimal de 0 a 32767.
R	Qualquer registrador
R'	Qualquer registrador preparado.
RR	Qualquer par de registradores
RR'	Qualquer par de registradores preparado

O primeiro código de operação, LOAD, é tratado nos dois próximos capítulos. A partir daí, abordaremos os códigos PUSH e POP, que são usados para salvar e recuperar informações.

Outros códigos de operação incluem *exchanges* (trocas), *block transfers* (transferências de bloco), *searches* (pesquisas), *loads with decrements and increments* (cargas com decrementos e incrementos), *adds* (adições), *subtracts* (subtrações), *logical ands* e *ors* (e's e ou's lógicos), *decrements* (decrementos), *increments* (incrementos), *rotates* (rotações), *shifts* (deslocamentos), *bit sets* (posicionamentos de bit), *bit resets* (reposicionamentos de bit), *jumps* (saltos ou desvios), *calls* (chamadas) e *returns* (retornos). Eles serão cobertos em capítulos posteriores.

QUESTÕES

1. O que significa LIFO?
2. A . . . . . é a memória disponível acima da última instrução do programa.
3. Um . . . . . é, às vezes, denominado uma pilha interna.
4. Se o topo da pilha é 7000H e o extremo inferior da mesma é 6000H, qual será o endereço da próxima posição disponível na pilha?

RESPOSTAS

1. last in first out (último a entrar, primeiro a sair)
2. pilha
3. "buffer"
4. 7001H

CAPÍTULO 6

O Grupo de Loads de 8 Bits

Vamos examinar, agora, o primeiro código de operação, o *LD*. LD significa LOAD (carregar). Este comando diz ao computador para carregar um registrador, par de registradores, ou posição de memória com um valor. Tal valor pode ser fornecido como entrada pelo teclado, de uma área dentro do programa, de um outro registrador, ou de uma posição de memória. Como uma definição geral, carregar significa *armazenar* informação.

Existem dois tipos de LOAD. Um LOAD de 8 bits e um de 16 bits, o qual será tratado no próximo capítulo.

LOADs de 8 BITS

Um *bit* (contração de binary digit – dígito binário) é a menor unidade de informação dentro de uma posição de memória. Existem 8 bits em um *byte*. Um byte pode representar uma letra, um número, um espacejamento ou um elemento de pontuação, por exemplo. Em cada posição de memória, há lugar para um byte.

Para estudar os códigos de operação, nós incluiremos os operandos, para dar uma idéia dos vários tipos de funções que podem ser realizadas pelos códigos de operação.

Uma boa regra geral, a usar para determinar que tipo de LOAD você está analisando, é que os LOADS de 8 bits usam registradores isolados. Vamos formatar as operações da seguinte maneira:

código de operação operando

Agora, vejamos os LOADs de 8 bits. O primeiro deles nos diz para carregar R,R'.

LD R,R'

Nesta instrução, o registrador R é carregado com o conteúdo do seu correspondente preparado R'. Por exemplo, se o registrador H contém 05H e

o registrador H' contém 10H, após LD H,H' ser executada, ambos os registradores conterão 10H.

Antes de LD R,R'

Registrador	Conteúdo	Registrador	Conteúdo
H	05H	H'	10H

Depois de LD R,R'

Registrador	Conteúdo	Registrador	Conteúdo
H	10H	H'	10H

Agora, suponha que desejemos carregar um registrador com o valor de um outro registrador.

LD R,R

Vamos supor que o registrador A contenha 0FH e que o registrador H contenha 2FH. Após a execução de LD A,H, os registradores A e H conterão ambos o valor 2FH. Isto também pode ser feito com registradores preparados. A seguir, temos dois exemplos de como fazer isto.

LD A,H
LD A',H'

Antes de LD R,R

Registrador	Conteúdo	Registrador	Conteúdo
A	05H	H	10H
A'	15H	H'	25H

Depois de LD R,R

Registrador	Conteúdo	Registrador	Conteúdo
A	10H	H	10H
A'	25H	H'	25H

O próximo comando acionará a carga de um registrador com o valor de um dado número. Os valores numéricos para registradores isolados não podem exceder 255 ou FFH. Todavia, você pode efetuar a carga em números decimais ou hexadecimais. Se o sufixo H não for adicionado ao número, o computador o tratará como um número decimal. Deve-se ter muito cuidado ao executar essas operações, pois há uma grande diferença entre 10 decimal e 10H, por exemplo. Além disso, se você tentar fazer a carga do número 8A e esquecer o H, o computador imprimirá uma mensagem de erro. Este comando de LOAD tem o formato:

LD R,N

Neste caso, se o registrador A contiver 04H e você desejar que ele passe a ter o valor 0FH, você executaria o seguinte comando: LD A,0FH. Depois da execução desta instrução, o registrador A terá como conteúdo 0FH.

Antes de LD R,N

Registrador	Conteúdo
A	04H

Depois de LD R,N

Registrador	Conteúdo
A	0FH

Agora, suponha que você deseje carregar um registrador com o conteúdo de uma localização de memória. Isto pode se tornar um tanto complicado, razão por que é importante prestar atenção aos detalhes. Você se lembra de que um registrador só pode armazenar valores até 255 e que, para números maiores que 255, você necessita usar um par de registradores.

Digamos que o par de registradores HL contenha o número 3C00H. Tal valor é, em nosso exemplo, o endereço de uma posição de memória. Todavia, o que queremos que seja carregado no registrador A é o conteúdo da posição de memória 3C00H e não o valor 3C00H, que é o conteúdo do par de registradores HL. Podemos fazer isto colocando o operando HL entre parênteses, como (HL). Suponhamos que na posição de memória 3C00H tenhamos o valor 45H, que, por acaso, é o código ASCII da letra "E". (Examinaremos os códigos de representação de caracteres ASCII mais tarde.) O comando para realizar a pretendida operação tem a forma:

```
LD      A, (HL)
```

Antes de LD A, (HL)

HL	Localização 3C00H	Registrador A
3C00H	45H	10H

Após LD A, (HL)

HL	Localização 3C00H	Registrador a
3C00H	45H	45H

Digamos que, inicialmente, o registrador A contenha o valor 10H. Depois de executarmos a instrução LD A, (HL), o registrador A conterá o valor 45H, o par HL continuará com o valor 3C00H e a posição de memória 3C00H permanecerá com o seu valor 45H.

O comando visto pode ser usado para qualquer registrador. No exemplo acima, nós evitaríamos utilizar os registradores H ou L para a operação, uma vez que isto destruiria o valor armazenado no par de registradores HL.

Vejamos, agora, os registradores especiais IX e IY. A mesma regra aplica-se a estes registradores, como foi feito no exemplo anterior. Existe, porém, uma diferença. Estes registradores podem trabalhar com um deslocamento, dentro do comando, sobre o valor do seu conteúdo. Digamos que o par de registradores IX contenha 3C00H e que o conteúdo da posição de memória 3CFFH seja 20H (o código ASCII do carácter espaço, branco).

O comando para os registradores IX e IY têm a forma geral:

```
LD    R, (IX + D)
LD    R, (IY + D)
```

O D, neste exemplo, denota o deslocamento. O delocamento é o número de bytes além do endereço apontado pelo conteúdo do par de registradores

IX ou IY. Contudo, se não há deslocamento, a sua indicação pode ser omitida na instrução. Desta forma, depois da execução da instrução LD A, (IX + FFH), o conteúdo do registrador A será 20H, sendo que o par IX conterá 3C00H e a posição 3CFFH continuará com seu valor 20H. Você também pode expressar o deslocamento em forma decimal.

Agora, vamos carregar um valor em uma posição de memória. O valor pode ser carregado em uma pilha ("stack"), em um "buffer" ou em uma posição correspondente a uma localização da tela do vídeo, por exemplo. Vamos fazer a carga do carácter "E" na posição correspondente à primeira posição da tela do vídeo, que é 3C00H. Suponhamos que, no momento, 3C00H contenha 20H, um espaço. Digamos, ainda, que o registrador A (o acumulador) esteja com o valor 45H, o "D" e que o par de registradores DE contenha 3C00H. O comando tem a forma geral:

```
LD      (RR),R
```

Antes de LD (RR),R:

Par de registradores	Conteúdo	Posição	Conteúdo	Conteúdo do Registrador A
DE	3C00H	3C00H	20H	45H

Após a execução de LD (RR),R:

Par de registradores	Conteúdo	Posição	Conteúdo	Conteúdo do Registrador A
DE	3C00H	3C00H	45H	45H

Substituindo os R's pelos respectivos registradores, teremos: LD (DE),A. Após a execução desta instrução, o registrador A ainda conterá 45H. O par DE ainda conterá 3C00H, mas a localização de memória 3C00H conterá, agora, o valor 45H. O resultado é que a letra "E" passará a ocupar a posição de memória que corresponde à primeira posição que é exibida em uma tela de vídeo. Mais à frente, trabalharemos com isto e usaremos estes procedimentos para exibir várias coisas na tela.

A próxima coisa que faremos é carregar a posição indicada pelo conteúdo dos dois pares de registradores especiais, IX e IY, com o conteúdo de um outro registrador. Faremos isto de forma análoga à que fizemos no último exemplo, com os comandos tendo a forma:

```
LD      (IX + D),R
LD      (IY + D),R
```

Os resultados são os mesmos da instrução LD (DE),A e o (IX + D) é tratado da mesma forma que em LD R,(IX + D), que já discutimos. A seguir, um exemplo:

Antes de LD (IX + D),R
LD (IY + D),R:

Par de Registradores	Conteúdo	Posição	Conteúdo
IX	3C00H	3C00H	20H
IY	3C50H	3C50H	41H

Conteúdo do Registrador A	Deslocamento
45H	10H

Posição	Conteúdo	Posição	Conteúdo
3C10H	50H	3C60H	49H

Depois da execução de LD (IX + D),R
LD (IY + D),R:

Par de Registradores	Conteúdo	Posição	Conteúdo
IX	3C00H	3C00H	20H
IY	3C50H	3C50H	41H

Conteúdo do Registrador A	Deslocamento
45H	10H

Posição	Conteúdo	Posição	Conteúdo
3C10H	45H	3C60H	45H

Existem duas outras possíveis maneiras de se carregar o registrador acumulador (A) com o conteúdo de uma localização de memória. Você já viu que podemos carregar o registrador A com o conteúdo do par HL: LD A,(HL). Agora vejamos as outras formas. Uma delas:

```
LD      A,(DE)
LD      A,(BC)
```

que nada mais é que o mesmo formato de LD A,(HL), executado com os pares DE e BC.

A outra forma é carregar um valor diretamente de uma posição de memória para o acumulador. Usa-se, para tal, a instrução da forma:

```
LD      A,(NN)
```

No exemplo a seguir, digamos que o registrador A contenha 50H, a localização 3C00H contenha 45H e que pretendemos que o valor armazenado em 3C00H seja carregado em A. O comando seria:

```
LD      A,(3C00H)
```

Localização	Conteúdo	Registrador	Conteúdo
3C00H	45H	A	50H

Depois de exècutarmos o comando, o conteúdo de 3C00H permanecerá sendo 45H e o conteúdo do acumulador se tornará 45H.

Localização	Conteúdo	Registrador	Conteúdo
3C00H	45H	A	45H

Este procedimento seria útil, por exemplo, para se apanhar a informação exibida na tela e enviá-la para a impressora.

Praticamente, tudo pode ser carregado no acumulador. Você se lembra de que, mais atrás, carregamos o conteúdo do par de registradores DE, ou melhor, a posição de memória indicada pelo conteúdo do par DE, com o conteúdo do registrador A. Podemos fazer o mesmo também com os pares de registradores HL e BC. Os comandos seriam:

```
LD      (HL),A
LD      (BC),A
```

Eis aqui exemplos desses comandos, assumindo que o registrador A contém o valor 52H. Os exemplos obedecem à ordem que vimos adotando até aqui.

Antes:

Registrador	Conteúdo	Localização	Conteúdo
HL	5000H	5000H	FFH
BC	5A00H	5A00H	23H

Depois:

Registrador	Conteúdo	Localização	Conteúdo
HL	5000H	5000H	52H
BC	5A00H	5A00H	52H

Nós também podemos carregar uma posição, ou seja, endereço, de memória com o conteúdo do registrador A. Fazemos assim:

```
LD      (NN),A
```

OU:

```
LD      (42EAH),A
```

Como você já deve ter concluído, no grupo de LOADs de 8 bits, o conteúdo do registrador, par de registradores, posição de memória, ou *o número à direita da vírgula* é carregado na localização de memória, registrador, ou par de registradores *à esquerda da vírgula.*

Existem mais quatro LOADs no grupo de 8 bits. Nós não os explicaremos neste livro mas os mostraremos a seguir. Considero que estas quatro instruções estão além do estágio do principiante e além do mais você não as usará tão cedo.

LD	A,I	Carrega o acumulador com o conteúdo do vetor de interrupção.
LD	A,R	Onde R = conteúdo do "refresh" da memória.
LD	I,A	Carrega o vetor de controle de interrupções com o conteúdo do acumulador.
LD	R,A	Onde R = registrador do "refresh" da memória, que passa a receber o conteúdo do acumulador.

Não se preocupe com interrupções e "refresh" neste livro, mas note que o acumulador é o registrador A.

QUESTÕES

1. O que significa LD?
2. . . . . significa armazenar informação.
3. Um. . . . . é a menor unidade de informação em uma posição de memória.
4. Existem 8 . . . . . em um . . . . . .
5. Mostre como se carrega o registrador A com o conteúdo do registrador A'.
6. Mostre como carregar o registrador A com o valor FFH.
7. Mostre como carregar o registrador A com o conteúdo da posição endereçada pelo conteúdo do par de registradores HL.
8. Diga qual o conteúdo do par de registradores HL, o conteúdo da localização endereçada pelo par de registradores HL e o conteúdo do registrador A, após a execução da instrução LD A,(HL), dadas as condições abaixo:

   A. O par de registradores HL contém 3C00H
   B. O conteúdo de 3C00H é 41H
   C. O registrador A contém 5EH

RESPOSTAS

1. LOAD, carregar
2. Carregar
3. bit
4. bits byte
5. LD A,A'
6. LD A,FFH
7. LD A,(HL)
8. HL conterá 3C00H
   3C00H conterá 41H
   O registrador A conterá 41H

CAPÍTULO 7

O Grupo de Loads de 16 Bits

Agora, conhecendo o grupo de LOADs de 8 bits, você está pronto para entrar no grupo de 16 bits. Não pense que se trata de algo mais difícil que o grupo de 8 bits. Realmente, não é.

O grupo de LOADs de 16 bits lida com pares de registradores, em lugar de registradores isolados. Com este grupo, veremos um novo par de registradores ser usado, o *par de registradores SP.* Este é o par de registradores que, continuamente, aponta para o topo da pilha. Por esta razão, ele é chamado de par de registradores SP, ou seja "stack pointer" (ponteiro da pilha).

Neste capítulo, você notará a ausência do par de registradores AF dos comandos de LOAD. Isto é devido a que o registrador A será usado para acumular e transferir informação para outros pares de registradores, enquanto que o registrador F é usado como sinalizador de diversas condições. Mais tarde estudaremos as condições que são assinaladas. A propósito, há ainda uma conveniência a mais: o registrador F armazenando a condição assinalada é facilmente lembrado como tal, pois o assinalamento também é conhecido como "FLAG" (bandeira), que inicia por F.

O primeiro LOAD é aquele que faz a carga de um número inteiro de 2 bytes em um par de registradores. Este número pode ser de zero a 65535 (FFFFH).

A codificação desta instrução é:

```
LD      RR,NN
```

Onde RR é qualquer dos pares de registradores BC, DE, HL, ou SP e NN é o número inteiro.

Vejamos como funciona esta instrução, tendo, como exemplo, NN = 3C00H e RR sendo o par DE.

```
LD      DE,3C00H
```

Depois da execução, o conteúdo do par de registradores DE será 3C00H e o conteúdo da localização de memória 3C00H (caso este número represente uma posição de memória) permanecerá inalterado. Detalhes do exemplo:

Antes

Par de reg.	Conteúdo	Localização	Conteúdo
DE	4000H	3C00H	45H

Depois

Par de reg.	Conteúdo	Localização	Conteúdo
DE	3C00H	3C00H	45H

A próxima instrução é:

```
LD      IX,NN
```

Esta instrução comanda a carga do par de registradores IX com um número inteiro. Por exemplo:

```
LD      IX,7FFFH
```

fará com que o par de registradores IX contenha o valor 7FFFH. O mesmo é válido para o par de registradores IY e se comportam da mesma forma que a instrução LOAD que examinamos anteriormente.

Vamos, agora, supor que desejemos carregar o conteúdo de uma posição de memória em um par de registradores. Isto pode ser feito usando-se qualquer dos pares BC, DE, HL ou SP. Carreguemos o par HL com o conteúdo da posição de memória 4000H. O comando seria:

```
LD      HL,(4000H)
```

ou

```
LD      RR,(NN)
```

Vamos fazer uma pausa e analisar melhor esta operação. Sabemos que existem 8 bits em um byte. Sabemos, também, que um byte representa uma posição de memória e, desta forma, existem 8 bits em cada posição de memória. Contudo, acabamos de realizar um LOAD de 16 bits. Isto significa que fizemos a carga do conteúdo de duas posições de memória em um par de registradores. Diga-me o que foi armazenado onde. (Dica: usei o par de registradores HL [H, em inglês é a inicial de alto, "high", e L a de baixo, "low"] para facilitar a memorização.)

Já que esta foi uma instrução de LOAD de 16 bits, os conteúdos foram buscados na pilha dentro da regra de último a entrar primeiro a sair. Portanto, o registrador H conterá o byte de ordem mais alta do LOAD, enquanto que o registrador L o de ordem mais baixa.

Vamos fazer de novo a operação, desta vez, olhando os conteúdos do par de registradores HL, após a execução da instrução. Digamos que a posição 4000H contenha o valor 20H e que a posição 4001H contenha o valor 45H. Depois da execução, o registrador H conterá 45H, obtido da posição 4001H e o registrador L,20H, buscado na posição 4000H. Um outro exemplo, para fixar a idéia:

Antes

Registrador	Conteúdo	Localização	Conteúdo
H	30H	4000H	20H
L	41H	4001H	52H

Depois

Registrador	Conteúdo	Localização	Conteúdo
H	52H	4000H	20H
L	20H	4001H	52H

Não fique com a idéia de que o par de registradores HL significa "high" "low" (alto/baixo), nem que o par HL é o único par que pode ser usado para este tipo de operação. Meramente, usamos o par HL neste exemplo, na esperança de que ele contribua para a sua memorização do exemplo, o que lhe facilitará eliminar dúvidas futuras, quando estiver lidando com outros pares de registradores em situações similares. É sempre uma boa idéia aprender estas pequenas dicas de memorização de procedimentos. Freqüentemente, quando você estiver escrevendo programas longos e complexos, ficará confuso ou um tanto cansado de interpretar as leis da linguagem de máquina. Nestas situações, estas dicas ajudarão a desanuviar a sua mente.

Agora que você já sabe como carregar o conteúdo de uma posição de memória em um par de registradores, como é que imagina que se possa fazer o contrário, isto é, carregar o conteúdo de um par em uma localização de memória? Pense um minuto sobre isto.

```
LD      (NN),HL
```

Novamente, para ajudar ao exemplo, usamos o par de registradores HL. Temos, em primeiro lugar, o comando LD, seguido pela posição de memória colocada entre parênteses, separada, a seguir, do par de registradores por uma vírgula.

O conteúdo do registrador L será carregado na posição de memória especificada. Para onde vai o conteúdo do registrador H? Vai para a posição + 1, ou seja, para a posição imediatamente superior à posição especificada, como mostrado:

Antes de LD (4000H),HL

Registrador	Conteúdo	Localização	Conteúdo
H	30H	4000H	20H
L	41H	4001H	52H

Depois

Registrador	Conteúdo	Localização	Conteúdo
H	30H	4000H	41H
L	41H	4001H	30H

Mais uma vez, procure lembrar-se do H, para alto, ("high") e do L, para baixo, ("low"). O par de registradores HL não é o único que pode ser usado para transferir informação para a memória. Podemos, também, usar, de forma

idêntica, os pares de registradores BC, DE, SP, IX e IY. Nós também podemos carregar pares de registradores diretamente a partir de outros pares de registradores.

Este método é limitado a três pares, a partir dos quais, a carga será feita em um determinado e específico par de registradores. Os três pares de registradores são o HL, o IX e o IY. Eles podem, unicamente, carregar seu conteúdo no par de registradores SP.

```
LD      SP,HL
LD      SP,IX
LD      SP,IY
```

Nestas instruções, o par de registradores SP é carregado com o conteúdo do outro par. Teremos, então, idênticos conteúdos em ambos os pares de registradores.

QUESTÕES

1. Mostre como carregar o par de registradores HL com o conteúdo da posição de memória endereçada pelo par de registradores DE.
2. Mostre como carregar o valor 3C00H no par de registradores HL.
3. Escreva a instrução que carrega o par de registradores DE com o conteúdo da posição 3C00H.
4. Indique o conteúdo do par de registradores HL, o conteúdo da localização apontada pelo par de registradores HL e o conteúdo do par de registradores DE, depois da execução da instrução LD DE,(HL), dadas as seguintes condições iniciais:
   A. O par de registradores HL contém 3C00H
   B. O conteúdo de 3C00H é 41H
   C. O conteúdo de 3C01H é 8EH
   D. O par de registradores DE contém 92FFH

RESPOSTAS

1. LD HL,(DE)
2. LD HL,3C00H
3. LD DE,(3C00H)
4. O par HL conterá 3C00H
   3C00H conterá 41H
   O par DE conterá 8E41H

CAPÍTULO 8

Instruções PUSH e POP

No último capítulo falamos um pouco sobre o par de registradores SP e dissemos que ele era o ponteiro da pilha ("stack pointer"). Vejamos, agora, uma das funções do "stack pointer".

O PONTEIRO DA PILHA

Digamos que temos um valor armazenado no par de registradores HL e que desejemos salvar este valor, para que possamos utilizar o par HL em outra operação. Nós poderíamos armazenar o valor que está no par HL em uma posição de memória ou em outro par de registradores. Contudo, digamos que todos os demais pares de registradores estão sendo usados e, mais, que não dispomos de espaço de memória interna em nosso programa (internal buffer). Suponhamos, ainda, que não tenhamos disponibilidade de "buffer" externo e que não saibamos onde se encontra o topo da pilha.

Primeiramente, vamos aprofundar o conceito de pilha. O topo da pilha é a *primeira posição de memória disponível que não é usada pelo programa.* Isto não significa que, se você tem uma máquina de 16K e a última posição usada pelo programa for 5000H, o topo da pilha seja a posição 5001H. Neste caso, o topo da pilha estará em 7FFFH, já que a pilha ainda não foi usada. Uma vez que seu uso obedece à regra de último a entrar, primeiro a sair, ela será construída de cima para baixo e não de baixo para cima (e será utilizada, isto é, lida, de baixo para cima). O fim da pilha estará em 7FFFH e o topo em 5001H se e somente se nós carregarmos dados em toda a pilha.

Contudo, se movermos alguma coisa para o topo da pilha, inicialmente, ela irá para 7FFFH e a seguir para 7FFEH. A próxima localização disponível seria 7FFDH, sendo que o par de registradores SP estaria apontando para 7FFEH. Outra coisa muito importante de ser compreendida neste ponto é que o par SP está sempre apontando para a última posição usada. Toda vez que ele é chamado para a execução de alguma operação, é decrementado de uma unidade, isto é, seu conteúdo é diminuído de 1.

INSTRUÇÃO PUSH

Voltemos agora ao problema inicial. Suponhamos que o par de registradores HL contenha o valor 1050H. O registrador H contém 10H e o registrador L o valor 50H. Podemos salvar este valor através do comando PUSH. O formato é:

PUSH HL

Digamos que, antes deste comando ser executado, o par de registradores SP esteja apontando para 700AH. Depois da execução, a localização 7009H conterá 10H enquanto que a localização 7008H conterá 50H. O par SP estará apontando para 7008H. Estude o seguinte:

Antes de PUSH HL (supondo que HL contenha 1050H)

Localização	Conteúdo	Localização	Conteúdo
7009H	00H	7008H	00H

Depois

Localização	Conteúdo	Localização	Conteúdo
7009H	10H	7008H	50H

Analisemos isto mais uma vez. Ao iniciarmos, o par de registradores SP estava apontando para 700AH. O comando PUSH decrementou o "stack pointer" de 1, de forma que ele passou a apontar para 7009H. A seguir, o conteúdo do registrador H foi carregado na posição 7009H. O par SP foi, então, decrementado mais uma vez, passando a endereçar a posição 7008H. Neste momento, o conteúdo do registrador L foi carregado na posição 7008H. Lembre-se de que esta foi uma operação LIFO. A propósito, todos os pares de registradores BC, DE, HL, AF, IX e IY podem ser salvos na pilha.

INSTRUÇÃO POP

Agora que temos o conteúdo do par HL armazenado na pilha, como podemos fazer para obtê-lo de volta de lá? Usamos o comando POP, que tem a forma:

POP HL

Após a recuperação do par, através da instrução POP o seu conteúdo será 1050H, novamente. Quando informamos ao computador para realizar um POP do par HL, o conteúdo da posição endereçada pelo "stack pointer" é carregado no registrador de mais baixa ordem do par (L). O SP é, então, incrementado e o conteúdo de posição agora endereçada por SP é carregado na ordem mais alta do par de registradores (H). O ponteiro da pilha (SP) é, então, incrementado mais uma vez. Eis o que aconteceu:

Antes de POP HL (admitamos que HL contenha 4000H)

Localização	Conteúdo	Localização	Conteúdo
7009H	10H	7008H	50H

Registrador	Conteúdo	Registrador	Conteúdo
H	40H	L	00H

O par SP contém 7008H

Depois

Localização	Conteúdo	Localização	Conteúdo
7009H	10H	7008H	50H
Registrador	Conteúdo	Registrador	Conteúdo
H	10H	L	50H

O par SP contém 700AH

Replay compacto. O SP está apontando para 7008H. Aparece a instrução POP HL. O conteúdo de 7008H é 50H. O registrador L é carregado com 50H. O SP move-se então para 7009H, que contém 10H. O registrador H é carregado com 10H. O conteúdo do par HL passou, então, a ser 1050H. (o SP é incrementado, então, para 700AH).

Quando o computador executou o PUSH do par de registradores HL, ele *não* tomou nenhuma nota do que, nem onde, ele estava carregando na memória. Isto é, ele não identifica que "esta é a área, na qual eu fiz o PUSH do HL". Isto faz com que seja necessário que você mantenha um controle da ordem em que executa PUSHs e POPs, dentro do programa. Se você não cuidar deste detalhe, pode, acidentalmente, não ter a informação carregada de volta no lugar que desejava.

Por exemplo, é possível fazer o PUSH de mais de um par de registradores, sem fazer o POP de outros da pilha. Se você executa PUSH HL e depois PUSH DE e o par SP está apontando para 700AH antes do primeiro PUSH, ocorreria o seguinte: O conteúdo do registrador H será armazenado em 7009H, L será posto em 7008H, D em 7007H e E em 7006H. O par SP estará, então, apontando para 7006H. Agora, quando você fizer o POP de qualquer registrador, será feita a carga dos conteúdos de 7006H e 7007H no par em questão. Isto poderá trazer problemas. Por exemplo:

	PUSH	HL	
	PUSH	DE	
Registrador	Conteúdo	Localização	Conteúdo
H	C9H	7009H	09H
L	01H	7008H	01H
D	3CH	7007H	3CH
E	00H	7006H	00H

Então, se você executa

	POP	HL	
	POP	DE	
Registrador	Conteúdo	Localização	Conteúdo
L	00H	7006H	00H
H	3CH	7007H	3CH

E	01H	7008H	01H
D	C9H	7009H	C9H

O conteúdo do par DE é, agora, aquele que estava originalmente armazenado no par HL e vice-versa. Se você quiser evitar estas inversões, a maneira correta de programar seria:

```
PUSH    HL
PUSH    DE
```

e, então,

```
POP     DE
POP     HL
```

Desta forma, você recupera corretamente os dados. Não é necessário fazer o POP dos pares de registradores subseqüentemente, isto é, você pode executar um POP de um par, realizar algumas funções e, depois, fazer o POP do próximo par. (Não se esqueça, último a entrar, primeiro a sair – LIFO.*)

*N.T. – Em outros tipos de microprocessadores, a lógica de execução das instruções PUSH e POP pode, freqüentemente, adotar convenções diferentes das do Z-80, que aqui foram analisadas. Isto pode ser, eventualmente, uma fonte de confusão para o programador.

QUESTÕES

1. Fazer um . . . . . significa salvar o valor contido em um par de registradores na posição do topo da pilha.
2. Fazer um . . . . . significa recuperar, para um par de registradores, a informação localizada na posição do topo da pilha.
3. Dê a ordem correta para se recuperar as informações dos pares de registradores DE e HL, após a execução de PUSH HL PUSH DE.
4. Mostre como colocar o valor do par de registradores HL no par DE e vice-versa, usando as instruções PUSH e POP.

RESPOSTAS

1. PUSH
2. POP
3. POP DE POP HL
4. PUSH DE PUSH HL
   POP DE POP HL

CAPÍTULO 9

Trocas Transferências em Bloco e Grupo de Pesquisa

No último capítulo, nós discutimos o uso do PUSH e do POP para permutar os conteúdos de dois pares de registradores. Poderíamos também realizar isto, através de várias combinações de comandos de LOAD. Veremos, agora, uma maneira mais fácil de conseguir isto, usando o comando EXCHANGE (Troca).

COMANDO EXCHANGE

O comando EXCHANGE que estaremos utilizando só é válido em certas circunstâncias. Vejamos o primeiro caso.

EX DE,HL

Neste comando, os conteúdos dos pares de registradores DE e HL são permutados, ou seja, trocados entre si. Eis um exemplo: Se o par HL contém o número 1000H e o par DE contém o número 5025H, depois da instrução EX DE,HL, o par de registradores DE conterá 1000H e o par HL conterá 5025H. É importante notar que o conteúdo original do registrador H era 10H, o do registrador L era 00H, o do registrador D era 50H e o do registrador E era 25H.

Registrador	Conteúdo	Par de Registradores	Conteúdo
H	10H	HL	1000H
L	00H		
D	50H	DE	5025H
E	25H		

Depois do comando EXCHANGE, o conteúdo do registrador H será 50H, o do registrador L 25H, o do registrador D 10H e o do registrador E 00H.

Registrador	Conteúdo	Par de Registradores	Conteúdo
D	10H	DE	1000H
E	00H		

H	50H	HL	5025H
L	25H		

O que estou tentando mostrar é que os registradores H e D trocam de conteúdo entre si, enquanto os registradores L e E também o fazem.

A próxima instrução de EXCHANGE que abordaremos é a instrução EX AF,AF'. Neste comando, o conteúdo do par de registradores AF é trocado com o conteúdo do seu correspondente par preparado AF'. O conteúdo do registrador A é permutado com o conteúdo do registrador A' e o conteúdo do registrador F é permutado com o conteúdo do registrador F'. Por exemplo, se o conteúdo do par AF for 4350H e o do par AF' 5025H, teremos, depois do comando

```
EX      AF,AF'
```

que o conteúdo do par AF será 5025H, enquanto que o do par AF' será 4350H. Estas trocas são elementares, porém você deve ter em mente duas coisas a respeito delas. A primeira é que elas só podem ser feitas entre determinados pares de registradores. Tais pares estão listados neste capítulo e nos manuais dos montadores Assembler. A segunda é que o comando EX só afeta os valores contidos nos registradores e não o conteúdo da memória endereçada por aqueles valores. Se o par de registradores HL contém 3C00H e esta localização contém o valor 20H e ainda, se o par DE contém 4000H e a posição 4000H contém 2EH, então, após o comando

```
EX      DE,HL
```

o par HL conterá 4000H e a localização 4000H continuará contendo 2EH. O par DE conterá, agora, 3C00H e esta localização 3C00H continuará armazenando o valor 20H, como observamos.

O próximo comando EXCHANGE a ser visto é o comando EXX. Ele é, geralmente, usado apenas em aplicações muito especiais. O comando tem a forma:

```
EXX
```

Observe que não há qualquer operando em seguida à instrução. Este comando diz para o computador que ele faça a troca dos conteúdos dos pares BC, DE e HL com os conteúdos dos seus respectivos correspondentes pares de registradores preparados BC', DE' e HL'.

Vamos exemplificar com uma tabela com valores para os pares BC, DE, HL, BC', DE' e HL':

BC	BC'	DE	DE'	HL	HL'
1000H	1010H	2000H	2020H	3000H	3030H

Após o comando EXX:

BC	BC'	DE	DE'	HL	HL'
1010H	1000H	2020H	2000H	3030H	3000H

Como você pode constatar, os conteúdos dos pares de registradores são intercambiados apenas com os dos seus correspondentes preparados. Não há troca de conteúdo entre os pares HL, DE e BC.

O próximo comando também tem um uso especial e limitado. Este comando troca o conteúdo da posição de memória apontada pelo "Stack Pointer" (par SP) pelo conteúdo dos pares HL, IX ou IY.

```
EX      (SP),HL
EX      (SP),IX
EX      (SP),IY
```

Novamente, estamos realizando uma simples permuta, porém sob certas condições. No primeiro exemplo, o conteúdo do registrador L é trocado com o conteúdo da posição de memória apontada pelo SP, enquanto que o conteúdo do registrador H é permutado com o conteúdo da posição imediatamente superior na pilha. Examinemos este EXCHANGE usando os seguintes valores:

O conteúdo do par HL é 3C00H
O par SP contém o valor 7000H
A localização 7000H contém 67H
A localização 7001H contém 32H

Dadas estas condições, ao executarmos a instrução EX (SP),HL, resultará o seguinte:

O par de registradores HL conterá 3267H
A posição de memória 7000H conterá 00H
A posição de memória 7001H conterá 3CH
O par de registradores SP conterá 7000H.

O mesmo seria válido se usássemos os pares IX ou IY, em lugar do par HL.

TRANSFERÊNCIAS EM BLOCO

Vejamos, agora, uma forma rápida de fazer a troca e a carga de conteúdos de pares de registradores. Isto é conseguido através do uso de transferências em bloco. Numa transferência em bloco, a informação é passada de um par de registradores para outro, sendo que, então, certos pares são incrementados (avançados de uma posição de memória apontada pelo seu conteúdo), enquanto que o conteúdo de outros são decrementados, em operação oposta à anterior.

A primeira que investigaremos é a instrução LDI. Ela aparece nos programas na forma:

```
LDI
```

Observe que não existe operando para o código de operação LDI. Isto é verdadeiro para todas as transferências em bloco. A razão é que a transferência em bloco já diz, sinteticamente, tudo que deve ser feito pelo computador. Isto também implica que você está limitado àquilo que pode ser transferido

com o uso da instrução TRANSFER (transferência). No caso da instrução LDI, um byte de dados é transferido da posição de memória apontada pelo par de registradores HL para o endereço apontado pelo par DE. Então, os conteúdos dos pares HL e DE são incrementados de uma unidade. Ao mesmo tempo, o conteúdo do par de registradores BC (byte counter – contador de bytes) é decrementado de uma unidade.

Vamos repetir a seqüência.

Estabeleçamos as seguintes condições. O par de registradores DE contém 3C00H, o endereço da área de memória correspondente à primeira posição na tela. Imagine que a tela esteja limpa, ou que ela esteja preenchida com espaços (caracteres que representam *em branco*). O código hexadecimal ASCII do espaço é 20H.

Desta forma, as localizações de memória 3C00H, 3C01H, 3C02H etc. conterão todas elas o valor 20H. Agora, desejamos exibir na tela os conteúdos das localizações de memória 5000H, 5001H e 5002H, que contêm as três primeiras letras do alfabeto, A, B e C. Os códigos destas letras são, respectivamente, 41H, 42H e 43H. Assim, a posição 5000H contém 41H, a posição 5001H contém 42H e a posição 5002 contém 43H. Agora, desejamos exibir as três letras. Fazemos isto informando ao contador de bytes (par BC) que desejamos que a instrução seja repetida três vezes. Para tal, deveremos já ter carregado nesse par de registradores o valor 03H.

Executamos, então, o comando LDI. Tal comando fará as seguintes coisas acontecerem:

O conteúdo da posição 5000H é colocado na posição 3C00H.
O par DE é incrementado para 3C01H.
O par HL é incrementado para 5001H.
O contador de bytes (par BC) é decrementado para 02H.

Agora, o conteúdo de 3C00H é 41H, o de 3C01H é 20H e o de 3C02H é 20H. O conteúdo da posição 5000H é 41H, o de 5001H é 42H e o de 5002H é 43H. O contador de bytes BC foi decrementado de um, passando para 02H.

Depois da execução de toda a instrução, teremos o quadro descrito anteriormente, porém note que a instrução não foi repetida. O conteúdo do par BC é, ainda, 02H e não 00H. As próximas execuções da instrução repetirão a seqüência descrita.

Vejamos a instrução LDIR. Esta instrução também não possui operandos. O operando é subentendido como sendo o mesmo da instrução LDI, porém com uma significativa adição: ela será repetida até que o conteúdo do par BC seja zero. Usando a mesma informação que usamos para o comando LDI, vamos examinar o que sucede com o comando LDIR.

Já sabemos que o processo será repetido três vezes, já que o conteúdo original do par BC era três.

Localização	Conteúdo	Localização	Conteúdo
3C00H	41H	5000H	41H
3C01H	42H	5001H	42H

3C02H	43H	5002H	43H
3C03H	20H		

O par BC contém 00H
O par DE contém 3C03H
O par HL contém 5003H

Assim como no comando LDI, o conteúdo das posições apontadas pelo par HL permanece inalterado.

Agora, a próxima instrução, a LDD. Esta instrução é similar à LDI, à exceção de que com a LDD os pares de registradores são todos decrementados. Usando os mesmos valores do exemplo anterior, vejamos o que produz a instrução LDD. Depois da primeira instrução teremos:

Localização	Conteúdo	Localização	Conteúdo
3C00H	41H	5000H	41H
3C01H	20H	5001H	42H
3C02H	20H	5002H	43H

O conteúdo do par HL será 4FFFH
O conteúdo do par DE será 3BFFH
O conteúdo do par BC será 02H

Observe que os conteúdos de todos os pares de registradores foram decrementados, embora tivesse sido produzido o mesmo resultado. Vamos repetir o exercício, mudando algumas coisas.

Primeiramente, alteremos o conteúdo de DE de 3C00H para 3C02H. A seguir, mudemos o conteúdo de HL de 5000H para 5002H. Deixaremos o conteúdo do par BC com 03H e os conteúdos das localizações de 3C00H a 3C03H e de 5000H a 5002H com os mesmos valores usados no primeiro exemplo. A última coisa a mudar será o código de operação. Vamos, agora, usar a instrução LDDR. Ela repetirá a ação da instrução LDD até que o conteúdo do BC seja 00H.

Eis os resultados:

Localização	Conteúdo	Localização	Conteúdo
3C00H	41H	5000H	41H
3C01H	42H	5001H	42H
3C02H	43H	5002H	43H
3C03H	20H		

O conteúdo de BC será 00H
O conteúdo de HL será 4FFFH
O conteúdo de DE será 3BFFH

Observe que se chegou ao mesmo resultado a que levou a execução da instrução LDIR, porém os conteúdos dos pares de registradores foram decrementados.

PESQUISAS

Já que nos familiarizamos com as duas primeiras partes deste capítulo, veremos, agora, o grupo de SEARCH (Pesquisa). Em linguagem de máquina, nós pesquisamos coisas através da comparação entre duas entidades. Isto é feito pela comparação de um determinado padrão com alguma outra coisa em questão. Esta outra coisa poderá ser um dado ou um valor que tenhamos inserido em algum lugar, dentro do programa. Como resultado, conforme ocorra uma identidade na comparação ou não, certas ações são executadas.

Por exemplo, se eu lhe der uma cesta contendo todos os tipos de castanhas do mundo e lhe pedir para localizar uma castanha-do-pará, você poderá, naturalmente, perguntar-me: – como é uma castanha-do-pará?.

Para abreviar toda a trabalheira de descrever as características de uma castanha-do-pará, eu ponho logo em sua mão uma dessas castanhas e peço para você encontrar uma igual a ela na cesta. Peço que você pesquise toda a cesta e, caso não encontre, levante o braço e, encontrando, fique de pé.

Ao realizar esta tarefa, você comparará todas as castanhas da cesta com aquela na sua mão e responderá de acordo com o estabelecido para os casos de encontrar ou não um casamento adequado.

Isto é denominado pesquisa por comparação e é desta forma que opera o computador.

Sabemos que o computador não pode levantar o braço ou ficar de pé. Por esta razão, nós lhe instruímos para que assinale um "FLAG" (sinalizador) de condição – CONDITION FLAG.

Posicionar um FLAG de condição significa colocar um valor zero ou um, em um byte, dependente de certa condição.

Não vamos nos aprofundar tecnicamente sobre isto, apenas diremos que a condição ficará armazenada no registrador F e que a "castanha", ou seja, o valor-padrão que estamos querendo pesquisar, é colocado no registrador A.

Por esta razão, não podemos alterar os conteúdos dos registradores A e F ou do par AF, durante uma operação de pesquisa.

FLAGS (Sinalizadores)

A tabela 9-1 mostra a lista dos flags e o seu uso. Explicarei cada um dos flags, de forma que saberemos o que estamos fazendo, quando estivermos realizando o procedimento de pesquisa.

O flag C, de "vai um", ou transporte, informa ao computador que o valor da coluna em questão, em uma soma, é grande demais para a coluna e que, portanto, ocorre um "vai um", ou, então, que tivemos que pedir emprestado, no caso de uma subtração. Este é o uso mais freqüente para o flag C, isto é, em uma soma ou subtração.

O flag N, de soma/subtração, é usado para indicar se o comando está requisitando uma soma ou subtração. Nós vimos um exemplo disto nas instruções LDI e LDD. Naqueles casos, tivemos que somar ou subtrair do conteúdo

dos pares de registradores em questão. Só para assentar idéias, o flag N é posto em 0 em todas as funções de soma e em 1 nas funções de subtração.

O flag P/V, de paridade /"overflow" (estouro) é posicionado sempre que ocorre uma condição de overflow em uma rotina de soma ou subtração. Um overflow (estouro) jamais ocorre quando se somam valores de sinais opostos (exemplo: + 100 somado a − 500). O estouro só ocorre em certos casos de soma de valores de sinais iguais, (ex: + 100 somado a + 500, ou − 100 somado a − 500). O oposto é verdadeiro no caso da função de subtração. O overflow pode ocorrer se e somente se subtraímos valores de sinais opostos.

O flag de transporte parcial (half carry), ou flag H, é usado primordialmente para ajustar o ponto decimal, durante as funções de soma ou subtração.

Tabela 9-1. *Introdução aos Flags.*

FLAG	USO
C	flag de "vai um", ou transporte
N	flag de soma/subtração
P/V	flag de paridade/overflow
H	flag de transporte parcial
Z	flag de zero
S	flag de sinal

O flag de zero, ou flag Z, indica qual é o estado do acumulador. Se o valor do acumulador for zero, o flag Z é posto com o valor 1. Se o valor do acumulador for diferente de zero, o valor de Z será 0.*

O último flag é o flag S, de sinal, que informa ao computador se o valor do acumulador é positivo ou negativo. Se o valor for positivo, o valor do flag será zero, se o valor do acumulador for negativo o flag terá o valor 1.**

COMANDOS DE COMPARAÇÃO (CP)

Vejamos, agora, alguns comandos de comparação.

O primeiro deles, neste grupo de pesquisa, é o comando CPI. Ele tem a forma:

CPI

Apenas pelo código de operação, o computador entende o que deve ser feito, sem necessidade de operandos. O comando instrui ao computador para que COMPARE o valor armazenado na posição de memória, endereçada pelo par de registradores HL, com o valor do acumulador (o registrador A). Se a comparação resultar em uma condição verdadeira, um flag é posicionado no registrador F e o conteúdo do par HL é incrementado de uma unidade. Ao mesmo tempo, o conteúdo do par BC é decrementado de uma unidade.

*N.T. – Isto após ser feito um teste de comparação com zero. Além desta operação, o flag Z pode ser, também, posicionado por outras operações.)

**N.T. – Outras operações também afetam o flag S.

Sigamos este processo com um exemplo. Digamos que o conteúdo do par HL seja 3C01H e que o conteúdo da posição 3C01H seja 20H. Vamos, também, assumir que o conteúdo do registrador A seja 20H. Finalmente, imaginemos que o conteúdo do par BC seja 05H.
Agora, após o comando

CPI ,

o computador terá verificado o conteúdo da posição 3C01H (posição endereçada pelo par HL) em comparação com o conteúdo do registrador A. Ele terá descoberto que os valores coincidem, porque, quando subtraiu o conteúdo da posição 3C01H do conteúdo do registrador A, o resultado da subtração foi zero, já que ambos os valores eram 20H. Portanto, o flag S terá sido reposicionado em zero, o flag Z terá sido feito igual a 1 e o par HL terá sido incrementado. O par BC terá, ainda, sido decrementado.

Agora, existe o seguinte quadro:

O conteúdo do par BC é 04H.
O conteúdo do par HL é 3C02H.
O conteúdo da localização 3C01H é, ainda, 20H.
O conteúdo do registrador A é, ainda, 20H.
O flag Z contém 1.
O flag S contém 0.

Podemos, agora, informar ao computador o que fazer, no caso do flag Z estar ligado (igual a 1). Porém suponha, agora, que o valor do registrador A fosse 21H, em vez de 20H, e que tudo o mais fosse o mesmo do exemplo. Ocorreria o seguinte:

O conteúdo do par BC é 05H.
O conteúdo do par HL é 3C01H.
O conteúdo da posição 3C01H é 20H.
O conteúdo do registrador A é 21H.
O flag Z não foi posicionado como 1.
O flag S não é posicionado em 1, mas o teria sido se o registrador A contivesse inicialmente o valor 1FH, pois a subtração teria deixado um valor negativo em A.

Este processo poderia ser repetido automaticamente, porém não com esta instrução. Para conseguirmos tal procedimento, usamos a instrução CPIR.

Usemos a mesma informação do último exemplo, alterando o código de operação para CPIR. O computador irá testar posições de memória, até que uma das seguintes condições ocorra:

1. Uma comparação verdadeira ocorra.
2. O conteúdo do par BC tenha sido decrementado até atingir o valor zero.

Primeiramente, se uma comparação verdadeira for encontrada, o computador agirá da mesma forma como agiu nesse caso com a instrução CPI.

Contudo, se não se encontra uma comparação coincidente, o par HL será incrementado e o par BC decrementado.

Uma outra coisa interessante também ocorre. O contador de instruções (Program Counter – PC) será decrementado de dois, uma vez que a instrução CPIR é uma instrução de dois bytes de tamanho. Isto fará o comando ser repetido e assim o processo se repetirá até que o conteúdo do par BC tenha sido decrementado até zero.

Todavia, o que aconteceria se o conteúdo do par BC já fosse zero, logo no início da instrução? Neste caso, a máquina iria executar um ciclo através de 64K bytes da memória, a menos que, antes disso, encontrasse uma condição verdadeira de comparação. Se ele não encontrar tal condição, prosseguirá para a próxima instrução e o flag Z não será colocado igual a 1 (o que indicaria o conteúdo de A ser zero após a operação de subtração, implicando, assim o encontro da condição verdadeira).*

Os próximos dois comandos são semelhantes aos dois últimos que vimos, porém com uma pequena diferença. Em lugar do conteúdo do par HL ser incrementado, ele é decrementado.

Vejamos o comando CPD e comparemo-lo com o comando CPI. Para tal, usemos as mesmas condições que usamos para exemplificar o comando CPI. Os resultados serão os mesmos, à exceção de que o conteúdo do par HL será decrementado, em lugar de incrementado. Os pontos importantes a guardar são:

1. O conteúdo do par HL é decrementado.
2. A função COMPARE é a mesma.

O último código de operação deste grupo, o CPDR, é bastante similar ao CPIR. De novo, a diferença reside em que o conteúdo do par HL é decrementado, em lugar de incrementado. A função e as regras são as mesmas nos comandos CPIR e CPDR. Existe, contudo, um aspecto comum aos dois comandos, que não abordamos até agora. Após a realização de cada comparação de dados, o computador reconhecerá uma condição de interrupção. *As interrupções fazem com que o computador suspenda a execução de certas funções, devido a determinadas condições.*

O que significa uma interrupção? Não examinaremos este assunto neste livro, já que ele destina-se a um uso avançado da linguagem. Apenas tenha em mente que o computador reconhecerá interrupções nos casos mencionados.

*N.T. – Existe, ainda, um dos comandos mais comumente usados, que é o

CP s

onde o operando s pode ser um número N menor que 256, ou o conteúdo de uma posição de memória endereçada pelos pares HL ou IX + D ou IY + D.
Entre outros flags, a instrução CP liga o flag Z quando a comparação do operando com o conteúdo do acumulador for positiva, isto é, o operando for igual ao conteúdo do registrador A.

QUESTÕES

1. . . . . informa ao computador para permutar o conteúdo do par BC com o seu correspondente par preparado e, ao mesmo tempo, permutar os conteúdos dos pares HL e DE com os correspondentes conteúdos dos respectivos pares preparados.
2. . . . . é a instrução que promove a permuta do conteúdo do par AF com o do seu correspondente par preparado AF'.
3. A instrução . . . . . transfere um byte de dados do endereço apontado pelo par HL para o endereço apontado pelo par DE. Depois disso, ela incrementa os pares HL e DE.
4. A instrução . . . . . executa a mesma ação do comando da questão 3, todavia com a repetição do comando.
5. A instrução . . . . . é, basicamente, a mesma instrução da questão 3, porém os pares HL e DE são decrementados.
6. A instrução . . . . . é a mesma da questão 5, porém a instrução é repetida.
7. CP significa . . . . . .
8. A instrução . . . . . manda o computador comparar o valor contido no registrador A com o valor armazenado na posição de memória endereçada pelo par HL. No caso de resultado positivo na comparação, um flag é posicionado (= 1) no registrador F e o conteúdo do par HL é incrementado. Ao mesmo tempo, o par de registradores BC é decrementado.
9. A instrução . . . . . é a mesma instrução da questão 8, à exceção de que o conteúdo do par HL é decrementado.
10. A instrução . . . . . é a mesma da questão 8, porém o comando é repetido.
11. A instrução . . . . . é a mesma da questão 9, porém o comando é repetido.
12. Liste os 6 flags e explique os seus usos.

RESPOSTAS

1. EXX
2. EX
3. LDI
4. LDIR
5. LDD
6. LDDR
7. COMPARE
8. CPI
9. CPD
10. CPIR
11. CPDR
12. C vai um (ou transporte)
    N soma/subtração
    P/V paridade/overflow (estouro)
    H transporte parcial
    Z zero
    S sinal

CAPÍTULO 10

O Grupo Aritmético de 8 Bits

Em linguagem de máquina, somente duas operações aritméticas são permitidas, a adição e a subtração. A multiplicação e a divisão são realizadas através de repetidas adições e subtrações, respectivamente.

ADIÇÃO E SUBTRAÇÃO REPETITIVAS

De que forma você pode multiplicar, através de sucessivas adições, e dividir, por meio de seguidas subtrações? Tome um pedaço de papel e escreva o número 3 dez vezes, em uma coluna. Some a coluna. A resposta terá sido 30. Tente de novo, usando outro número. A resposta continuará sendo a mesma que você obteria se tivesse multiplicado o número pelo número de vezes que ele aparece na coluna. Você acabou de realizar a multiplicação através de sucessiva adição.

Vamos, agora, realizar a divisão, usando a subtração. Desta vez, escreva o número 30 e subtraia 3 desse número. Você obtém 27. Subtraia 3 de 27 e continue o processo, até ter executado cinco subtrações.

$30-3=27 \quad 27-3=24 \quad 24-3=21 \quad 21-3=18 \quad 18-3=15$

Entendeu o processo? Cinco é a metade de 10. Dez vezes 3 é 30. A metade de 30 é 15. Aqui, mais uma vez, você usou uma operação sucessivamente, a subtração, para obter a divisão.

Uma coisa importante a ter em mente é que o registrador A (o acumulador) contém o valor a ser somado ou subtraído.

COMANDOS DE ADIÇÃO

Existem cinco distintos comandos de adição. O primeiro instrui ao computador para somar o conteúdo de um registrador ao conteúdo do registrador A. O registrador em questão pode ser qualquer, dentre o A, B, C, D, E, H ou L. Observe a ausência do F nesta lista. Ele irá conter o flag de condição e, desta forma, não poderá ser usado.

Vejamos o que sucede quando somamos o conteúdo do registrador H ao registrador A. Suponha que H contenha 10H e que A contenha 05H. Depois do comando

```
ADD     A,H
```

O registrador H continuará com seu valor 10H mas o registrador A terá, agora, o valor 15H, a soma dos valores dos dois registradores. O que terá acontecido com os flags? Vamos examiná-los.

O FLAG S será ZERADO
O FLAG Z será ZERADO
O FLAG H será ZERADO
O FLAG P/V será ZERADO
O FLAG N será ZERADO
O FLAG C será ZERADO

Examine os flags e descubra por que as condições resultantes foram estas. Dica: seria diferente, se os números fossem 19H e 19H.

Resposta – O flag H seria ligado, devido ao transporte.

Agora, um comando para somar, diretamente, um número ao registrador A:

```
ADD     A,N
```

Nesta instrução, o número, aqui representado por N, é somado ao conteúdo do acumulador. Já que esta é uma operação de 8 bits, o número N não pode exceder 255 ou FFH.

Vejamos um exemplo. Se o conteúdo do acumulador for 10H e o número em questão, a ser somado, 15H, teremos após a instrução

```
ADD  A,15H
```

Registrador	Conteúdo	Valor Somado	Novo conteúdo
A	10H	15H	25H

O acumulador conterá 25H. Poderíamos também somar um número decimal ao acumulador. Substituamos o 15H por seu equivalente, em notação decimal 1,21.

```
ADD     A,21
```

Observe que o 21 não é seguido da letra H. O computador tratará este número como um número decimal e o somará ao conteúdo do acumulador, após ter feito sua conversão para o correspondente hexadecimal 15H.

O que ocorre com os flags? Eles resultam no mesmo estado que observamos no exemplo anterior. Aliás, eles seguem as mesmas regras para todo o grupo de aritmética de 8 bits.

Vamos, agora, somar o conteúdo de uma posição de memória ao registrador A. O comando para tal é:

```
ADD     A,(HL)
```

Se o conteúdo do registrador A for 10H, o do par HL for 4000H e o da localização 4000H for 15H, então, após a execução da instrução, os resultados serão os seguintes:

Registrador	Par de Registradores	Localização
A	HL	4000H
25H	4000H	15H

O par de registradores HL continuará com 4000H.
O conteúdo da localização 4000H continuará sendo 15H.
O registrador A passará a conter o valor 25H.

As mesmas regras são aplicáveis aos dois comandos de adição restantes:

```
ADD     A,(IX + D)
ADD     A,(IY + D)
```

Lembre-se de que D, aqui, representa um deslocamento. A propósito, se não for informado nenhum deslocamento, o computador lidará com o endereço contido no par IX ou IY. Exemplos:

```
ADD     A,(IX)
ADD     A,(IX + 5)
```

Se o conteúdo do par IX for 4000H, então, na primeira instrução, o conteúdo da localização 4000H é adicionado ao conteúdo do registrador A. Já na segunda instrução, o conteúdo da posição 4005H, isto é, 4000H + 5, será somado ao registrador A.

ADIÇÃO ESPECIAL

Temos, agora, um tipo de adição especial. Trata-se da instrução ADC.

A instrução ADC significa adicionar (ADD) com transporte (CARRY). Em outras palavras, some o conteúdo do registrador, ou de posição de memória, ou um número N qualquer ao registrador A e leve junto o excesso, acaso indicado no flag C (CARRY – transporte, "vai um").

Se o registrador A contém 16H e o flag C estiver ligado (= 1) e executarmos uma soma do valor 10H ao acumulador, com a ADC, eis o que acontece:

```
ADC     A,10H     (O flag C está ligado)
```

teremos:

Registrador	Conteúdo	Valor adicionado	Novo Conteúdo
A	16H	10H	27H

O registrador A conterá 27H. Sabemos que 10H somado a 16H dá 26H, porém observe que esta soma foi uma adição com transporte e o flag de transporte, ou vai um, estava ligado (= 1). Desta forma, pudemos capturar este 1 do flag, que estava lá perdido no registrador F.

COMANDOS DE SUBTRAÇÃO

O formato da subtração é o mesmo da adição, ou seja

```
SUB     A,H
```

Se o registrador A contiver 10H e o registrador H contiver 05H, então, após a execução da instrução, o conteúdo do registrador A será 5H.

As mesmas variações de comandos existentes para o ADD (soma), existem para o comando de subtração SUB. Contudo, é interessante prestar atenção ao flag de sinal, já que o resultado pode ser negativo.

Eis um pequeno exercício para você. Escreva as condições dos flags para o exemplo acima e então reescreva-as para as seguintes condições:

O conteúdo do registrador A é 05H.
O conteúdo do registrador H é 10H.

Resposta: O flag N será ligado.
O flag S será ligado.

Observe a diferença, se você subtrai 16 de 5, obtém 11 negativo, ou −11.

O último comando de subtração, que analisaremos em separado, é o comando SBC, ou seja, subtraia também o transporte (neste caso trata-se do "tomar emprestado"). Este comando é similar ao ADC, à exceção do fato de que a operação envolvida é a subtração. Ele também segue as regras básicas da subtração mencionadas atrás.

Vamos supor que o conteúdo do registrador A seja 16H e que o flag de transporte esteja ligado. Digamos que desejemos subtrair 05H do registrador A. O comando seria:

```
SBC     A,05H
```

O resultado seria que o registrador A passaria a conter o valor 10H. Mais uma vez observe que o flag de transporte estava ligado e, por isso, causou a ocorrência de uma condição de "tomar emprestado", ou seja, subtraiu-se 1 do registrador que contém o resultado.

OPERAÇÃO LÓGICA "E"

Vejamos, agora, uma instrução mais difícil, o comando AND (E). Esta é uma operação lógica E, na qual se realiza uma soma bit a bit. "Soma bit a bit do quê?", você pode perguntar. Existem onze possíveis diferentes combinações para este comando. São elas:

```
AND     R (onde R é qualquer dos registradores B, C, D, E, H, L, ou
        A)
AND     N (onde N é qualquer número até FFH)
AND     (HL) (onde HL endereça uma posição de memória)
AND     (IX + D)
AND     (IY + D)
```

Quando o computador encontra uma instrução AND, ele executa o que se poderia chamar de uma adição, onde há analogia entre os correspondentes bits. O resultado ficará armazenado no registrador A. Um exemplo explica melhor:

Se o conteúdo do registrador A for C3H, isto significa que o valor binário ali armazenado é 11000011. Se o registrador B contiver 7BH, isto significa que contém o valor binário 01111011. Quando o computador encontrar

```
AND     B
```

ele fará a soma dos registradores desta maneira:

```
B       01111011
A       11000011
        --------
A       01000011   , ou seja 43H.
```

Devemos ressaltar que, em uma adição lógica binária:

$0 + 0 = 0$
$1 + 1 = 1$
$1 + 0 = 0$

MASCARAMENTO

O uso mais comum para a instrução AND é a possibilidade de fornecer uma máscara. Uma máscara é usada para cobrir alguma coisa, sobre a qual você não deseja agir. Neste caso, não desejamos que determinada coisa seja armazenada no registrador A.

Vejamos um exemplo. Os códigos hexadecimais para os números do teclado de 0 a 9 são, respectivamente, 30H a 39H. Suponha que você queira guardar um dígito no registrador A mas não o seu valor hexa. Você poderia fazer isto executando uma subtração do valor 30H do número em questão, porém seria mais fácil fazer um mascaramento do valor contido em A. Se o registrador A contiver 30H e nós o mascararmos usando a instrução

```
AND     OFH
```

apenas os últimos quatro bits do byte permaneceriam em seu estado original durante a soma lógica. Isto é porque OFH é 00001111 em binário. 30H é 00110000 em binário. Desta forma, quando estes dois valores são logicamente somados, produz-se, no registrador A, o valor binário 00000000 que desejávamos.

OPERAÇÃO LÓGICA "OU"

O próximo comando é o comando lógico OR (OU). Este comando usa uma outra tabela de soma lógica para executar a soma bit a bit dos valores em questão. Diferentemente da última função que vimos, o E LÓGICO, onde o único caso em que o resultado era 1, ocorria quando os dois bits, sobre os quais se estava operando, eram 1, a função OU opera da seguinte forma:

1 + 1 = 1
1 + 0 = 1
0 + 0 = 0

Com este tipo de operação, também conhecido por adição binária real, se, por exemplo, tivermos o valor 48H (01001000) no registrador A e o registrador H contiver 12H (00010010), obteremos o valor 5AH (01011010), quando efetuarmos a operação OR. Vejamos como se sucedem as coisas:

OR	H	
A	48H	01001000
H	12H	00010010
A	5AH	01011010

Observe que, neste caso, obteríamos o mesmo resultado se tivéssemos somado (instrução ADD) os dois valores 12H e 48H (mas isto foi apenas um caso particular, em que, no ADD, não teríamos nenhum transporte, ou "vai um"). O comando OR pode ser executado em uma das seguintes maneiras:

1. Pode ser usado com os seguintes registradores: B, C, D, E, H, L e A.
2. Pode ser usado para somar logicamente (segundo a tabela) números até FFH.
3. Pode ser usado para somar logicamente (tabela) o conteúdo de posição de memória endereçada pelo par HL (OR (HL)).
4. Pode ser usado com os pares de registradores IX e IY (OR (IX + D) OR (IY + D)).

OPERAÇÃO LÓGICA "OU EXCLUSIVO" (EXCLUSIVE OR, ou XOR)

Este comando OR especial, chamado de EXCLUSIVE OR ou XOR, isto é, OU EXCLUSIVO, é também destinado ao mesmo tipo de uso que os comandos AND e OR, no que diz respeito aos operandos. Contudo, sendo ele um tipo especial de comando OR, tem, de fato, um uso especial, qual seja o de zerar o acumulador. Se executarmos o comando

XOR A

o conteúdo do acumulador será zero, como resultado.*
Agora que você passou por esta parte mais difícil, vamos fazer uma pequena recapitulação e aprender algumas coisas mais fáceis.

*N.T. – A tabela lógica do XOR é a seguinte:

1 + 1 = 0
1 + 0 = 1
0 + 0 = 0
isto é, o resultado é 1, se e somente se, apenas um dos elementos da operação for 1.

COMPARAÇÃO DE 8 BITS

No último capítulo, aprendemos alguns comandos de comparação (COMPARE), que utilizamos no grupo de pesquisa (SEARCH). Vejamos, agora, uma comparação muito simples e comum. Nesta comparação nós apenas comparamos 8 bits. A propósito, este é o tipo de comparação que nós, provavelmente, mais usaremos. Eis o comando e as maneiras pelas quais ele pode ser usado:

```
CP      B
```

O conteúdo do registrador B, ou qualquer dos registradores C, D, E, H, L ou A, é comparado com o conteúdo do registrador A. Como resultado, um flag é posicionado no registrador F e o programa continua com a próxima instrução. Normalmente, a próxima instrução diz ao computador o que fazer no caso do registrador F atender a determinadas condições especificadas na instrução.

Outros possíveis usos são:

```
CP      N
```

Onde N é um número não maior que FFH.

```
CP      (HL)
```

Compara o conteúdo da posição endereçada pelo par HL.

```
CP      (IX + D)
```

Compara o conteúdo da posição endereçada pelo conteúdo do par IX adicionado do deslocamento.

```
CP   (IY + D)
```

o mesmo que (IX + D).

O COMANDO DE INCREMENTO

Foi fácil? Então, eis aqui um ainda mais fácil.

```
INC     R
```

Este comando incrementa um registrador R, que pode ser qualquer, dentre os registradores A, B, C, E, H ou L. Por exemplo, se o registrador H contém 05H, depois do comando

```
INC     H
```

este registrador conterá 06H. Além disso, o registrador F irá posicionar um flag para assinalar a condição resultante da execução da instrução INC.

O comando INC também pode ser usado para incrementar o conteúdo de posições de memória, endereçadas pelos pares HL, IX + D ou IY + D. A forma seria:

```
INC     (HL)
INC     (IX + D)
INC     (IY + D)
```

aqui também os flags apropriados são posicionados.

O COMANDO DE DECREMENTO

O último comando neste grupo é o comando DECREMENT, ou DEC. Você deve lembrar-se de que discutimos este comando no último capítulo. Ele implica subtração de uma unidade.

O comando DEC pode ser executado sobre qualquer dos registradores B, C, E, H, L ou A. Também pode ser efetuado sobre as posições endereçadas pelos pares HL, IX + D e IY + D.

Quando o computador encontra o comando DEC, o valor armazenado no registrador ou na posição de memória é decrementado de 1. Por exemplo, se o registrador B contém 05H, depois do comando

```
DEC     B
```

ele passará a conter 04H.

Assim como no comando INC, o registrador F apresentará uma configuração de flags posicionados em função das condições causadas pela execução deste comando. Estas condições de estado podem variar entre o ligamento do flag S e/ou do flag P/V.

QUESTÕES

1. Mostre como se soma o número 15H ao conteúdo do registrador A.
2. Qual seria o conteúdo resultante do registrador A, na questão 1, caso seu conteúdo, antes da soma, fosse 0AH?
3. Se o registrador A contém FFH e o registrador H contém 10H, escreva o comando para subtrair o conteúdo do registrador H do registrador A.
4. Usando a informação da questão 3, mostre os conteúdos de antes e depois da operação nos registradores A e H.
5. O que significa ADC?
6. O que significa SBC?
7. Quando o computador encontra a instrução . . . . . , ele executará uma adição lógica binária, segundo a analogia entre os bits.
8. A instrução . . . . . utiliza uma adição real binária, para obter o resultado da operação lógica.
9. Mostre como comparar o valor do registrador A com o conteúdo da posição endereçada pelo par de registradores HL.
10. Mostre como adicionar 1 ao conteúdo da posição de memória endereçada pelo par HL, usando o comando INC.
11. Usando a instrução DEC, subtraia 1 do conteúdo da posição endereçada pelo par HL.

RESPOSTAS

1. ADD A,H
2. 1FH
3. SUB A,H
4.

Registrador	A	H
Antes	FFH	10H
Depois	EFH	10H

5. ADD com transporte (CARRY)
6. SUB com transporte (CARRY)
7. AND
8. OR
9. CP (HL)
10. INC (HL)
11. DEC (HL)

CAPÍTULO 11

Aritmética de Âmbito Geral e o Grupo de Controle da UCP

Vejamos o primeiro comando deste grupo, o comando DAA.

O COMANDO DAA

Este comando ajusta, ou seja, modifica o acumulador para a execução das operações de adição e subtração em código BCD. BCD significa Binary Coded Decimal – Decimal codificado em binário.

Suponha que queiramos somar o decimal 15 ao decimal 27. Desejamos que a resposta fique em representação hexadecimal, já que o computador trabalha com hexa. Vejamos o que acontece, quando somamos os dois números.

$$15 + 27 = 42$$

Veja os números em binário:

Decimal	Binário	Decimal	Binário
1	0001	5	0101
2	0010	7	0111
3	0011	C	1100

Isto nos forneceu a resposta 3C, porém nós queríamos 42. O que aconteceu?

O computador está trabalhando com hexa e nós em decimal. Contudo, o computador pode reconhecer representações binárias por dois sistemas, o hexadecimal e o BCD. Para modificar o resultado para que ele seja representado na forma decimal codificada em binário, nós usamos a instrução DAA, a qual não utiliza operandos.

A instrução DAA, automaticamente, adiciona um certo valor ao número para modificá-lo para a representação BCD. Desta forma:

Hexa	Binário	Hexa	Binário
3	0011	C	1100
	+ 0001		+ 0110
4	0100	2	0010

Não se preocupe com os valores que são somados. Usamos estes como um exemplo. O que é importante ter em mente é que a instrução DAA faz o ajuste automaticamente.

O COMANDO CPL

Suponhamos que você deseje inverter o conteúdo do registrador A. Temos um comando para realizar isto:

CPL

Quando o computador encontra esta instrução, ele inverte o conteúdo do registrador A como um complemento de 1. Eis um exemplo:

Bit nº	1	2	3	4	5	6	7	8
Registrador A antes	0	0	1	1	1	1	1	0
Registrador A depois	1	1	1	0	0	0	1	1

Os valores dos primeiros quatro bits do registrador A estão na mesma ordem, porém localizados nos últimos quatro bits do registrador, e os últimos quatro estão, agora, na posição dos primeiros quatro. Os valores dos bits 1 e 5 foram permutados, assim como o foram os dos bits 2 e 6, 3 e 7, 4 e 8.

O COMANDO NEG

Um outro comando, do qual você não terá, provavelmente, necessidade durante um certo tempo, é o comando NEG. Quando esta instrução é executada, o conteúdo do registrador A é trocado como um complemento de 2. Isto é o mesmo que subtrair o conteúdo do acumulador de zero. Se o registrador contiver + 10, a instrução faria com que passasse a conter – 10. Vejamos um exemplo.

Bit Nº	1	2	3	4	5	6	7	8
Registrador A antes	1	0	0	1	1	0	0	0
Registrador A depois	0	1	1	0	1	0	0	0

Note que os valores binários dos primeiros quatro bits são trocados, em relação ao que eram, antes da execução da instrução NEG.

INSTRUÇÕES CCF E SCF

Até o momento, não temos falado muito acerca do registrador F. As duas instruções a seguir, a CCF e a SCF, lidam, especificamente, com aquele registrador.

A instrução CCF não possui operando e inverte o FLAG de CARRY (transporte) do registrador F.

A instrução SCF liga o FLAG de CARRY no registrador F, além de desligar (= 0) os FLAGS H e N.

O COMANDO NOP

Esta instrução é assim chamada por significar que nenhuma operação deva ser feita, isto é "no operation", ou no-op, ou NOP. É comumente usada em linguagem de máquina, embora possa também ser usada em Assembler. Quando o computador encontra um NOP, ele nada executa e, simplesmente, passa para a próxima instrução.

Se isto é verdade, então, por que seria tal instrução usada em linguagem de máquina? Em linguagem de máquina, você, provavelmente, estaria usando um programa manipulador dos seus dados de entrada (instruções), tipo T-BUG, para construir o seu programa. Se você desejasse reservar uma área no programa, um "buffer", encheria essa área com NOPs. O código de máquina do NOP é 00 e, desta forma, você teria uma área cheia de zeros e isto garantiria que você não teria nenhuma "sujeira", rejeitável pelo computador, naquela área.

As próximas seis instruções que compõem este grupo são usadas apenas por programadores muito avançados. São elas o HALT, DI, EI, IM 0, IM 1 e IM 2. Não vamos entrar na discussão destas instruções, porque não desejamos que você se confunda, nem que lhe tragamos alguma desorientação. Todavia, depois de dominar esta linguagem e a tiver utilizado por algum tempo, sugiro que você procure ler alguns livros mais avançados, que abordem estas instruções.

QUESTÕES

1. O que é a UCP?
2. O que significa BCD?
3. Que comando ajusta o valor no acumulador para o código BCD?
4. Que instrução transforma decimal em hexadecimal (isto é, uma representação hexadecimal, cuja configuração binária representa o número decimal, através de um código)?
5. Qual a instrução que inverte o conteúdo do registrador A?
6. A instrução . . . . . subtrai o conteúdo do registrador A de zero.
7. A instrução . . . . . inverte o flag de carry (transporte) do registrador F.
8. A instrução . . . . . pode ser usada para reservar espaço dentro do programa.

RESPOSTAS

1. Unidade Central de Processamento
2. Binary Coded Decimal – Decimal codificado em binário
3. DAA
4. DAA
5. CPL
6. NEG
7. CCF
8. NOP

CAPÍTULO 12

O Grupo Aritmético de 16 Bits

Neste capítulo, discutiremos as instruções ADD (Soma), SUB (Subtração), INC (incremento) e DEC (decremento) de 16 bits. Elas são bastante parecidas com as correspondentes instruções de 8 bits. A principal diferença é que elas usam pares de registradores em lugar de simples registradores.

SOMAS (ADDS) DE 16 BITS

A primeira delas é

ADD HL,SS

onde SS é qualquer dos pares de registradores BC, DE, HL ou SP. Por exemplo, se o par HL contiver o número 2121H e o par DE o valor 4534H, então, depois do comando

ADD HL,DE

Você terá os seguintes resultados:

Antes

Par de Registr.	Conteúdo	Par de Registr.	Conteúdo
HL	2121H	DE	4534H

Depois

Par de Registr.	Conteúdo	Par de Registr.	Conteúdo
HL	6655H	DE	4534H

O conteúdo do par HL conterá o resultado da soma, isto é, 6655H. Você deve ser cuidadoso com este comando, ou poderá fazer alguma coisa desastrosa. Você pode somar algo que, no fundo, não desejava.

Não se esqueça de que os conteúdos dos pares de registradores são somados. Se você tiver uma posição de memória (endereço) armazenada em um destes pares e se esquecer de movê-la (salvá-la) em outro local, tal como em um "buffer", ou fazer o seu PUSH para o topo da pilha, ou armazená-la em um outro par de registradores e, então, carregar no par em questão o valor

que você quer ver somado ao outro par, você acabará somando ao conteúdo do par um valor que representa um endereço, em vez do número que pretendia originalmente. Vejamos um exemplo.

Digamos que o par DE esteja apontando para a posição 3C00H, o par HL contenha o valor 4000H e a localização 3C00H contenha o valor 41H. Se você se esquecer de fazer, por exemplo, um PUSH DE e um LOAD do valor desejado neste par, antes da execução da instrução.

```
ADD  HL,DE
```

isto fará com que o conteúdo de HL seja 7C00H. O conteúdo do par DE permanecerá o mesmo, assim como o conteúdo da posição 3C00H. O problema é que o valor resultante em HL não terá nada a ver com o que você, eventualmente, queria.

Um erro destes poderia levar horas para ser detectado, portanto, seja atencioso.

O próximo é o comando ADC. A definição de ADC é ADD com CARRY (transporte). Este comando é executado da mesma maneira que o comando ADD, mas, neste caso, o flag de transporte (carry) é somado ao conteúdo do par HL, e o resultado de tal soma fica armazenado no próprio par HL.

Eis um exemplo. Sendo o conteúdo do par HL 5437H, o conteúdo do par DE 2222H e o flag de transporte no registrador F estando ligado (= 1), o comando

```
ADC  HL,DE
```

resultará no conteúdo do par HL igual a 765AH. Se somarmos 5437H a 2222H, obtemos 7659H, mas, já que o flag de carry do registrador F estava ligado, ao usarmos o comando ADC, nós somamos o valor 1H a mais no resultado 7659H, obtendo 765AH.

SBC

O mesmo é verdadeiro no caso do comando SBC (Subtração com transporte), no que diz respeito às operações nos pares de registradores. No comando SBC HL,RR, o par RR seria qualquer dos pares BC, DE, HL, ou SP. Um bom exemplo deste comando seria:

Consideremos o conteúdo do par HL como sendo 6666H, o conteúdo do par DE como 2222H e o flag C do registrador F ligado. Com tais condições, a execução de

```
SBC  HL,DE
```

produzirá o seguinte: O conteúdo do par DE continuará a ser 2222H. O conteúdo do par HL será 4443H. Isto se deve ao fato de que o flag C, estando ligado, causou a subtração de 1 do resultado final no par HL.

OUTROS ADDs (Somas)

A próxima instrução é o ADD IX,PP. Nesta instrução, PP seria qualquer dos pares:

BC, DE, IX, ou SP.

Embora esta instrução seja processada da mesma forma que a instrução ADD HL, RR e produza, exatamente, os mesmos resultados, através da mesma rotina da instrução ADD, o par de registradores HL não pode ser utilizado nesta instrução. A dica para se lembrar desta característica do comando é que qualquer coisa que seja somada ao par IX não pode estar no par HL.

A última instrução de ADD é a ADD IY,RR. Nesta instrução, RR pode ser qualquer dos pares BC, DE, IY, ou SP. Mais uma vez, observe a ausência do par HL na lista. A instrução ADD IY,RR funciona identicamente à instrução ADD HL,RR, produz os mesmos resultados através da mesma rotina seguida pela instrução ADD, com a única limitação de excluir o par HL como operando. Novamente, a maneira de memorizar este aspecto é atentar para o fato de que qualquer coisa que se queira somar ao par IY não poderá estar no par HL, ou no par IX.

INC E DEC

Os últimos dois comandos deste grupo são o INC e o DEC. Estas funções podem ser executadas em qualquer dos pares HL, DE, BC, SP, IX, ou IY. Assim como nas correspondentes instruções de incremento e decremento do grupo de 8 bits, o conteúdo do par operando é incrementado ou decrementado de 1.

Por exemplo, se o conteúdo do par DE é 3C01H, executando-se

```
INC  DE
```

teríamos o novo conteúdo de DE como sendo 3C02H. Da mesma forma, executando

```
DEC  DE
```

resultaria em um conteúdo de 3C00H em DE.

QUESTÕES

1. A principal diferença entre o grupo aritmético de 8 bits e o grupo de 16 bits é que aquele utiliza . . . . . , enquanto que este trabalha com. . . . . .
2. Indique as condições anteriores e posteriores à instrução ADD HL,DE dado que HL contenha 1315H e DE 3F8EH.
3. Mencione qual a diferença entre as seguintes instruções:

   A. LD A,(HL)
   B. LD DE,(HL)
   C. LD DE,HL

RESPOSTAS

1. registradores pares de registradores

2.

Pares de registradores	HL	DE
antes	1315H	3F8EH
depois	52A3H	3F8EH

3. A. Carregar o registrador A com o conteúdo da posição endereçada pelo par HL. É uma carga de 8 bits.
   B. Carregar o par DE com o conteúdo da posição endereçada pelo par HL. É uma instrução de carga de 16 bits.
   C. Carregar o par de registradores DE com o conteúdo do par HL. Trata-se de uma carga em 16 bits.

CAPÍTULO 13

O Grupo de Rotação (ROTATE) e de Deslocamento (SHIFT)

Este capítulo poderá ser um tanto difícil, porém é de capital importância. Como um programador em linguagem de máquina, você irá querer executar uma série de rotinas. O conteúdo deste capítulo o ajudará a realizar, com facilidade, a implantação de complexas rotinas.

ROTATE (Rotação)

A primeira parte deste grupo é a seção sobre o ROTATE. Você vai logo ver o que realiza um ROTATE e em que implica a sua execução. Porém, primeiramente, vou-lhe dar uma regra primária geral.

Geralmente, quando você roda para a esquerda (ROTATE left), você está multiplicando por 2. Quando você roda para a direita (ROTATE right), você está executando uma rotina de divisão.

Em alguma parte atrás, mencionei que, em lingugem de máquina, não podemos multiplicar e dividir. Para tal, temos que realizar sucessivas somas e subtrações. Isto era, é e será sempre verdade. Um ROTATE é, apenas, uma outra maneira de realizar tudo isto.

Vejamos o primeiro ROTATE. Trata-se do RLCA (Rotate Left with Carry Arithmetic). Esta instrução não possui operando, uma vez que ele é subentendido – trata-se do registrador A, o acumulador.

Quando o computador encontra esta instrução, automaticamente move o conteúdo de cada bit, a começar pelo bit 0, para o próximo bit à esquerda. O conteúdo do último bit é movido para o flag de transporte (carry) do registrador F e para o primeiro bit do registrador A.
Eis um exemplo:

Condição do acumulador antes do RLCA								
Bit Nº	7	6	5	4	3	2	1	0
Conteúdo	1	0	0	0	1	0	0	0

Condição do acumulador após o RLCA

Bit Nº	C	7	6	5	4	3	2	1	0
Conteúdo	1	0	0	0	1	0	0	0	1

onde C é o flag de carry (transporte) do registrador F. Observe de que forma foi realizado o ROTATE. Agora examine o valor decimal do número, antes e depois da operação. Deixo isto com você, mas acredito que não lhe será difícil verificar o que aconteceu e o que aconteceria se repetíssemos a instrução diversas vezes.

Vejamos agora a instrução RLA (Rotate Left Arithmetic). Nesta instrução, como na anterior, o conteúdo de cada bit é movido para o seu vizinho à esquerda, porém com uma diferença: O conteúdo do último bit é movido para o flag de transporte (carry), mas não é movido para o primeiro bit. Em vez disso, o conteúdo anterior do flag de carry é movido para o primeiro bit.

Antes

Bit Nº	C	7	6	5	4	3	2	1	0
Conteúdo	1	0	1	1	1	0	1	1	0

Depois

Bit Nº	C	7	6	5	4	3	2	1	0
Conteúdo	0	1	1	1	0	1	1	0	1

As próximas duas instruções são semelhantes às duas anteriores, com a diferença única da rotação ocorrer para a direita. A primeira é a Rotate Right with Carry Arithmetic, RRCA.

No exemplo a seguir, você constatará que ocorrerá o mesmo fato com o acumulador, como na RLCA, com a exceção da rotação ser para a direita.

Antes do RRCA

Bit Nº	7	6	5	4	3	2	1	0
Conteúdo	0	0	0	1	0	0	1	0

Depois

Bit Nº	7	6	5	4	3	2	1	0
Conteúdo	0	0	0	0	1	0	0	1	0

Antes de prosseguirmos com a próxima instrução, escreva estes dois exemplos em binário e converta-os a decimal:

valor decimal antes do RRCA	18
valor decimal depois	9

Observe a mudança ocorrida, em função da execução do comando RRCA.

A próxima instrução é a Rotate Right Arithmetic, RRA. Este comando é idêntico ao RLA com a exceção da rotação ser executada para a direita. O exemplo a seguir mostra, com clareza, o que acontece:

Antes do RRA

Bit Nº	7	6	5	4	3	2	1	0	C
Condição	1	1	1	0	0	0	0	1	0

Depois

Bit Nº	7	6	5	4	3	2	1	0	C
Condição	0	1	1	1	0	0	0	0	1

Observe que o conteúdo prévio do flag de carry é movido para o bit 7 e que o conteúdo anterior do bit 0 é movido para o flag de carry.

Mais uma vez, quero que você copie os números binários do exemplo anterior e converta-os a decimal, para se familiarizar com o que sucede nestas operações.

As quatro instruções a seguir lidam com a operação Rotate Left with Carry – giro à esquerda com transporte, ou seja RLC.

Neste comando, o conteúdo de cada bit do byte é deslocado para o próximo bit à esquerda. O conteúdo do bit 7 é movido para o flag de transporte (carry) e também para o bit 0.

Estas são as quatro possíveis combinações do comando RLC:

RLC R
RLC (HL)
RLC (IX + D)
RLC (IY + D)

onde R representa qualquer dos registradores B, C, D, E, H, L, ou A, e o D das expressões IX + D e IY + D representa um deslocamento, conforme já abordado em capítulos anteriores.

Estas instruções são idênticas à RLCA, com as mesmas coisas ocorrendo com o conteúdo do registrador representado por R, ou com a posição de memória endereçada por HL, ou por IX + D, ou por IY + D.

A próxima instrução é a RL, isto é, Rotate Left. Esta instrução, da mesma forma que a instrução RLA, desloca o conteúdo de cada bit para o seu vizinho à esquerda. O bit 7 é levado para o flag de carry e o conteúdo deste é movido para o bit 0. A seguir, as várias combinações possíveis com este comando, ou seja, as mesmas do anterior:

RL R
RL (HL)
RL (IX + D)
RL (IY + D)

Um exemplo, para fixar as idéias:

Antes do RL

Bit Nº	C	7	6	5	4	3	2	1	0
Conteúdo	0	1	0	0	0	1	1	1	1

Após o comando

Bit Nº	C	7	6	5	4	3	2	1	0
Conteúdo	1	0	0	0	1	1	1	1	0

Lembrando que o byte pode ser qualquer dos registradores ou a posição endereçada pelo par HL ou os pares IX + D e IY + D.

A próxima instrução, o Rotate Right with Carry, giro à direita com transporte, ou seja, RRC, executa o mesmo que a instrução RLC, à única exceção do giro ser feito para a direita, entre os bits do byte. As formas possíveis de operandos para esta instrução são as mesmas das duas instruções anteriores.

O último Rotate é o Rotate Right, ou rotação à direita, instrução RR. Faz o mesmo que a instrução RL, sobre os mesmos operandos, com a distinção única de a rotação entre os bits ser para a direita.

INSTRUÇÃO SHIFT – Deslocamento

O próximo conjunto de instruções são as instruções de SHIFT, isto é, de deslocamento. Diferentemente do Rotate, o SHIFT faz com que o primeiro bit do lado por onde se inicia o deslocamento seja zerado e o valor do último bit seja perdido.

O primeiro SHIFT que examinaremos é o Shift Left Arithmetic, deslocamento aritmético para a esquerda, ou seja, SLA. Este comando pode ser executado sobre qualquer dos registradores B, C, D, E, H, L, ou A. Também pode operar sobre a posição de memória apontada pelos pares HL, ou IX + D, ou IY + D. Vejamos, com um exemplo, o que acontece quando esta instrução é executada:

Se o conteúdo do registrador A for:

Bit Nº	7	6	5	4	3	2	1	0
Conteúdo	0	0	0	0	1	1	1	1

então, após a execução de

```
SLA     A
```

o conteúdo de A será:

Bit Nº	C	7	6	5	4	3	2	1	0
Conteúdo	0	0	0	0	1	1	1	1	0

Observe que o conteúdo do bit 7 foi deslocado para o flag C do registrador F e que o bit 0 foi zerado. Escreva os valores decimais derivados dos números binários do exemplo acima e constate qual a relação entre eles

valor decimal anterior	15
valor decimal posterior	30

A próxima instrução é a Shift Right Arithmetic, ou SRA, isto é deslocamento aritmético para a direita, que faz, exatamente, o mesmo da instrução SLA, com a diferença de que o giro é para a direita e que o bit 7 permanece inalterado. Eis um exemplo:

Antes do SRA

Bit Nº	7	6	5	4	3	2	1	0
Conteúdo	1	0	1	1	1	0	0	0

Após o SRA

Bit Nº	7	6	5	4	3	2	1	0	C
Conteúdo	1	1	0	1	1	1	0	0	0

valor decimal anterior 184
valor decimal posterior 220

QUESTÕES

1. Quando usamos um . . . . . estamos multiplicando por 2.
2. Quando usamos um . . . . . estamos dividindo por 2.
3. Explique a denominação das instruções abaixo:

A.	RLC	E.	RRCA
B.	RL	F.	RRA
C.	RLCA	G.	SLA
D.	RLA	H.	SRA

RESPOSTAS

1. ROTATE LEFT
2. ROTATE RIGHT
3. A. Rotate Left with Carry
   B. Rotate Left
   C. Rotate Left with Carry Arithmetic
   D. Rotate Left Arithmetic
   E. Rotate Right with Carry Arithmetic
   F. Rotate Right Arithmetic
   G. Shift Left Arithmetic
   H. Shift Right Arithmetic

CAPÍTULO 14

Ligar e Desligar Bits e o Grupo de Instruções de Teste

Neste capítulo, aprenderemos alguns detalhes muito interessantes acerca dos bits. Por exemplo, você pode testar qualquer bit de qualquer registrador, para saber se ele é zero ou 1. "Para que eu precisaria saber disto?", você pode perguntar. Você verá, mais tarde, como desviar a execução para diferentes locais, dentro de um programa, e seria muito bom se pudéssemos testar a condição de um determinado bit e, dependendo desta condição, dirigir a execução para esta ou aquela rotina, dentro do programa.

Uma outra coisa que você aprenderá neste capítulo é como mudar o valor de um bit isolado de um byte. Isto é particularmente útil, se você precisa ligar um bit de teste nos casos em que certa condição ocorra. Posteriormente, ao testar este bit, você tomará a ação para o caso detectado. Quando faz este posicionamento do bit, tal condição irá perdurar, até que você a altere novamente. Exemplos da utilidade deste aspecto nós encontramos nos programas de jogos e de aplicações comerciais.

Nos programas de jogos, você poderia criar uma condição que indicaria se determinado movimento foi ou não legal, ou se uma carta jogada é maior ou não que outra, por exemplo. Você também pode se valer dessas funções para manter um placar.

Em aplicações comerciais, por exemplo, se você deseja assinalar se uma conta está ou não aberta, pode utilizar esta função. Ela também poderia ser usada para indicar o estado de certa peça em estoque.

INSTRUÇÃO BIT

Vejamos a primeira instrução deste grupo, a instrução BIT.

BIT B,R

onde B é qualquer bit, de 0 a 7, e R é qualquer dos registradores B, C, D, E, H, L, ou A. O R também pode representar o conteúdo de um endereço apontado pelos pares HL, IX + D ou IY + D.

Quando o comando BIT é executado, o flag Z do registrador F conterá o complemento do bit que foi testado.

Se o bit testado era 1, o flag Z será 0 e vice-versa.

Vejamos um exemplo de uso da instrução. Supondo que o bit 5 do registrador H seja zero, ao executarmos

BIT 5,H

o flag Z do registrador F será 1, enquanto que o bit 5 do registrador H continuará sendo zero:

Antes de BIT 5,H									
Bit Nº	7	6	5	4	3	2	1	0	Z
Condição	1	1	0	0	0	0	0	1	0
Após BIT 5,H									
Bit Nº	7	6	5	4	3	2	1	0	Z
Condição	1	1	0	0	0	0	0	1	1

Mais tarde veremos de que forma o posicionamento do flag Z pode nos ser útil. No momento, o importante é entender de que forma isto é feito.

INSTRUÇÃO SET

A próxima instrução é a instrução SET, que tem a forma:

SET B,R

Da mesma forma como descrito na instrução BIT, o B significa o número do bit e R um dos registradores ou posição apontado pelos pares de registradores. O bit pode ser qualquer um de 0 até 7 e R representa o mesmo conjunto de possibilidades descrito na instrução anterior.

A instrução SET faz com que o bit especificado seja ligado (= 1) no registrador ou posição em questão.

Por exemplo, se o bit 3 do registrador L for 0, então, após a execução de

SET 3,L

o conteúdo do bit 3 daquele registrador passará a ser 1:

Antes de SET 3,L									
Bit Nº	7	6	5	4	3	2	1	0	Z
Condição	1	1	1	0	0	0	0	1	0
Após SET 3,L									
Bit Nº	7	6	5	4	3	2	1	0	Z
Condição	1	1	1	0	1	0	0	1	1

Isto será útil para colocarmos um valor-padrão para teste pela instrução BIT. Embora não seja uma boa prática de programação ficar saltando para frente e para trás, vejamos o que aconteceria se fizéssemos uma instrução BIT 3,L no exemplo que usamos para mostrar a instrução SET:

Antes de BIT 3,L

Bit Nº	7	6	5	4	3	2	1	0	Z
Condição	1	1	1	0	0	0	0	1	0

Se nós executássemos BIT 3,L antes de executarmos a instrução SET, o flag Z seria ligado. Se tal fosse feito após a execução da instrução SET, o flag Z seria desligado da seguinte maneira:

Após o comando SET

Bit Nº	7	6	5	4	3	2	1	0	Z
Condição	1	1	1	0	1	0	0	1	1

Se você não percebeu como nem por que tal acontece, leia novamente as instruções BIT e SET.

Existe, também, um modo de se desligar (= 0) um bit do registrador ou posição de memória. Utilizamos, para tal, o comando RES, que é escrito assim:

```
RES  B,R
```

onde, novamente, B e R representam as mesmas entidades descritas nas instruções BIT ou SET.
Executando o comando:

```
RES  3,L
```

faríamos com que o bit 3 do registrador L fosse colocado em zero. Se, então, executássemos

```
BIT  3,L
```

o flag Z do registrador F seria ligado.

QUESTÕES

1. Mostre de que maneira podemos testar o quinto bit do registrador L.
2. Mostre como ligar o primeiro bit do registrador A.
3. De que maneira você conseguiria trocar o bit nº 7 do registrador D, caso ele fosse 1?

RESPOSTAS

1. BIT 5,L
2. SET 0,A
3. RES 7,D

CAPÍTULO 15

O Grupo de Salto (JUMP)

Este é o grupo de instruções que você devia estar ansioso para conhecer. Ele lhe permitirá dirigir a execução do programa para qualquer área, incondicionalmente, isto é, à sua vontade, ou dependendo de certas condições.

A INSTRUÇÃO JUMP (Salto)

Esta primeira instrução, que examinaremos, nos permitirá desviar para um ponto específico do programa que determinemos. Seu formato é:

JP NN

onde NN é a posição de memória que desejamos. Se você tem alguma experiência prévia com BASIC, isto é o mesmo que a instrução GOTO.

Existem duas maneiras de se saltar para uma posição em linguagem Assembler, porém apenas uma em linguagem de máquina. Antes de explicar tal detalhe, quero salientar que a única maneira de voltar atrás será através de um outro JUMP para trás, dentro do programa. Quando você usa um JUMP, não pode usar um RETURN para voltar à instrução subseqüente ao JUMP inicial. (Discutiremos o comando RETURN mais à frente.)

A primeira maneira de executar um JUMP é espeficiar a posição em hexadecimal. Este é o método que você deve usar em linguagem de máquina. Também é permitido em linguagem Assembler, porém existe uma maneira mais fácil de fazê-lo (e mais recomendável).

Esta é uma das razões pelas quais usamos rótulos nas instruções. Você pode executar um JUMP, fazendo um desvio, ou salto, para um rótulo, também conhecido pelo termo inglês *"LABEL"*. Quando o Assembler encontra um JUMP para um rótulo, ele, automaticamente, calcula o endereço da primeira posição de memória do rótulo. A partir daí, ele converte o rótulo em endereço. Desta maneira, a coisa é bem mais fácil, em linguagem Assembler, já que não precisamos calcular cada posição de memória para inseri-la na instrução. (Uma outra vantagem é que, caso você modifique o programa, inserindo

instruções entre os dois pontos em questão, ou seja, o ponto da instrução JUMP e o ponto para o qual o salto deva ocorrer, você teria que recalcular os endereços, se usar a forma de endereçamento direto em lugar de rótulo). A seguir, exemplos dos dois tipos de JUMPS:

Se o contador de instruções se encontra com o valor 5000H e a instrução

```
JP  4000H
```

for executada em seguida, então o programa suspenderá automaticamente a execução na posição 5000H e deslocará a execução para a posição 4000H. (O contador de instruções também será atualizado com o valor 4000H.)

Uma outra maneira de fazermos isto seria através do uso de um rótulo. Suponha que tenhamos uma rotina, dentro do programa, à qual demos o Rótulo ROTINA e que a posição de memória da primeira instrução em ROTINA esteja no endereço 653FH. Digamos que o contador de instruções esteja apontando para a posição 45A4H. Se a instrução localizada em 45A4H for

```
JP  ROTINA
```

o contador de instruções será automaticamente colocado em 653FH e a instrução em 653FH será executada.

A próxima instrução do grupo de JUMP é denominada JUMP condicional. Quando o computador encontra um salto condicional, ele só fará o desvio se determinada condição for satisfeita.

As condições que são reconhecidas estão mostradas na tabela 15-1.

No capítulo anterior, dissemos que o flag Z do registrador F apresentaria o complemento da condição do bit testado. Se a condição do bit for 1, o flag Z será zerado. Poderíamos executar o comando JP NZ,ROTINA. Vejamos de que forma estas duas instruções trabalhariam em conjunto:

```
BIT   3,H          ; teste o bit 3 do registrador H
JP    NZ,ROTINA    ; desvie para ROTINA se o bit 3 do reg. H for 1.
```

TABELA 15-1. JUMPs Condicionais

CONDIÇÃO	FLAG RELEVANTE
NZ (Não zero)	Z
Z (Zero)	Z
NC (Não transporte)	C
C (Transporte)	C
PO (Paridade ímpar)	P/V
PE (Paridade par)	P/V
P (Sinal positivo)	S
M (Sinal negativo)	S

JUMP RELATIVO

Eis um comando um pouco mais difícil. Esta instrução informa ao computador para executar um JUMP RELATIVO, ou JR. Ela é seguida de um deslocamento, o qual o computador deve saltar. Este deslocamento se inicia na posição seguinte à instrução e a execução é retomada na próxima localiza-

ção após o deslocamento. Isto, em suma, significa que o programa saltará por sobre um certo número de posições de memória, especificado pelo deslocamento.

Por exemplo, esteja o contador de instruções na localização 5000H e a instrução encontrada em 5000H seja JR 5. (Esta instrução tomará duas posições: em 5000H teremos a instrução JR e em 5001H o deslocamento 5.) O contador de instruções começaria na localização 5002H e saltaria 5002H, 5003H, 5004H, 5005H e 5006H. O programa, então, recomeçaria a execução em 5007H.

Existem dois cuidados, aqui, a serem ressaltados. Eu não recomendo o uso da instrução JR para principiantes, já que, por descuido, se você saltar para o meio de uma instrução, pode danificar todo o programa. Isto ocorre porque tanto números quanto instruções são codificados em hexa. Existe uma instrução codificada que corresponderá a qualquer número armazenado na memória. Portanto, se você saltar para um número – digamos que seja para o deslocamento de um outro JR, por exemplo –, o computador tratará aquele deslocamento como um código de operação e executará o que quer que tal código mande fazer. Isto mudaria todo o programa.

O segundo ponto refere-se ao fato de que você só consegue saltar, com o JUMP RELATIVO, 129 posições para frente e 126 para trás.

Em linguagem de máquina, se você cometer qualquer descuido, poderá causar um erro fatal no programa (um erro que impossibilite a execução do mesmo, ou cause seu encerramento inesperadamente). Se você cometer o mesmo erro em linguagem Assembler e, por exemplo, comandar um JR ao contador de instruções para um rótulo que esteja fora da área possível de acesso, o Assembler lhe alertará quanto a este fato, o que lhe permitirá, facilmente, o acerto da instrução, trocando o JR por JP.

No final deste livro, há uma lista de deslocamentos válidos para o JUMP RELATIVO. Ela não só lhe ajudará a programar em linguagem de máquina, quanto aos deslocamentos, como será útil para seguir um rastreamento do programa, caso você tenha que depurá-lo usando um programa do tipo T-BUG.

Existem outros quatro possíveis comandos de JUMP RELATIVO. São eles:

```
JR  C,D
JR  NC,D
JR  Z,D
JR  NZ,D
```

Nestas instruções, como na anterior, D representa o deslocamento do salto relativo. À exceção de que se trata de um JUMP RELATIVO, as condições que devem ser obedecidas, para que o desvio seja executado, são as mesmas do comando JP. (Trabalhando em Assembler, você pode especificar um rótulo do programa no lugar do deslocamento D.)

Existe, ainda, mais um JUMP de interesse para nós. Nesse JUMP, nós desviaremos para um endereço que esteja contido em um par de registradores.

Somente três pares podem ser utilizados para armazenar endereços com a finalidade de uso por tal JUMP. São os pares HL, IX e IY. As instruções seriam:

```
JP   (HL)
JP   (IX)
JP   (IY)
```

Quando o computador encontra uma destas instruções, o desvio é feito para a localização armazenada no par de registradores em questão.

A última instrução é a de um salto de deslocamento sob uma condição não-zero (DISPLACEMENT JUMP ON NON ZERO – DJNZ). Como o próprio nome indica, trata-se de uma instrução onde há um desvio, caso uma condição de não-zero seja obedecida.

Para testar se há zero ou não, o computador verifica o conteúdo do registrador B. Se o conteúdo não for zero, ele será decrementado de 1 e o desvio é executado. Contudo, se o conteúdo de B for zero, a próxima instrução, em seqüência, será executada. Eis um exemplo:

```
        LD    A,0H    ; coloca zero em A
        LD    H,05H   ; armazena o valor 5 no registrador H
        LD    B,03H   ; carrega o número de "loops" (voltas)
LOOP    ADD   A,H     ; A agora contém mais 5
        DJNZ  LOOP    ; se B não for zero, desvie para a soma
        JP    NEXT    ; vá para a próxima rotina
```

Na rotina acima, o registrador A é, inicialmente, colocado em zero. A seguir, carregamos o registrador H com o valor 5. O registrador B é, então, carregado com o valor 3. Na linha denominada (rótulo) LOOP, o conteúdo do registrador H é somado ao conteúdo do registrador A. Na primeira passagem, o registrador A conterá o valor 5H, após a execução do ADD. A seguir, a instrução DJNZ é executada. Isto faz decrementar o conteúdo de B, após o computador ter verificado que ele não era zero, ocorrendo, então, o desvio para LOOP. O conteúdo de H (5H) é, novamente, somado ao conteúdo de A. Esta soma ocorrerá quatro vezes, antes que a instrução DJNZ detecte que o registrador B contém zero. Neste ponto, o programa passará à instrução seguinte e executará o JUMP para o rótulo especificado como NEXT. Neste exemplo, o valor final de A será 14H. Esta é uma das maneiras de se executar multiplicação através de sucessivas adições.

N.T. – Uma das razões pelas quais o programador optaria por usar um JR em lugar de um JP, quando isto for possível, é o fato de que a instrução JR é mais curta, menor 1 byte, que a JP e facilita a relocação do programa.

QUESTÕES

1. Mostre como saltar para o rótulo HERE, se um teste realizado no registrador A resultar em uma condição de zero. Mostre o mesmo para o caso do teste resultar em não-zero.
2. Mostre como executar um JUMP RELATIVO, na questão anterior.

RESPOSTAS

1. JP	Z,HERE	JP	NZ,HERE
2. JR	Z,HERE	JR	NZ,HERE

CAPÍTULO 16

O Grupo de Chamada (CALL) e Retorno (RETURN)

Este é o último grupo de instruções que cobriremos neste livro. Existem ainda dois outros grupos, o de INPUT (entrada) e o de OUTPUT (saída), porém são grupos de instruções que só devem ser utilizados por programadores mais experientes.

INSTRUÇÃO CALL (Chamada)

O CALL é bastante parecido com o JUMP. Há, porém, uma diferença básica. O CALL deve ter um RETURN (retorno), para ser uma instrução válida. Assim como no JUMP incondicional, existem duas maneiras de se executar um CALL incondicional em linguagem Assembler. Já em linguagem de máquina, há somente uma forma. A instrução CALL incondicional é da forma:

```
CALL    NN
```

onde NN é a posição de memória da sub-rotina que deve ser executada. Em linguagem Assembler, podemos aqui fazer um CALL também para um rótulo (label), em vez de um endereço de memória.

Quando o computador encontra a instrução CALL, o conteúdo do contador de instruções é colocado (PUSH) no topo da pilha. A seguir, o contador de instruções é carregado com o endereço de memória de dois bytes, para onde deve ser desviada a execução. Quando se chegar a um RETURN, o ponteiro da pilha (Stack Pointer) é incrementado de 3 (uma instrução CALL tem o comprimento de 3 bytes, 1 para o código de operação e 2 para o endereço) e o conteúdo da nova posição, por ele apontada, é restaurado (POP) no contador de instruções. Isto faz com que a execução seja retomada na linha seguinte à da instrução CALL.

Existe um outro tipo de CALL, o CALL condicional. O CALL condicional é executado se o conteúdo do registrador F (flags) atende à condição especificada no programa. A instrução tem o formato:

CALL CC,NN

onde CC é a condição e NN a localização da chamada.

Se a condição for satisfeita, a instrução CALL é executada da mesma maneira explicada no CALL incondicional; caso contrário, o programa continuará a execução na instrução da linha seguinte.

INSTRUÇÃO RETURN

Assim como existem dois tipos de CALL, também existem dois tipos de RETURNs (retornos). O primeiro é o RETURN incondicional:

RET

Quando o computador encontra este tipo de instrução, o Stack Pointer (ponteiro da pilha) é incrementado de 3 (porque a instrução CALL ocupa 3 bytes) e o conteúdo da posição agora apontada por ele é transferido (POP) para o contador de instruções. Desta forma, a execução continua na linha seguinte à da instrução CALL.

O segundo é o RETURN condicional:

RET CC

onde CC é a condição imposta. Este RETURN é executado somente se a condição estabelecida na instrução for satisfeita. As condições são as mesmas possíveis para a instrução CALL, isto é, os flags do registrador F são testados para se ver se obedecem ao especificado.

QUESTÕES

1. Mostre como chamar o rótulo (label) HERE, caso o teste que tenha sido feito no registrador A tenha dado resultado zero. Mostre como fazer a chamada no caso contrário, isto é, de resultado não-zero.
2. Mostre como realizar um retorno, na questão acima.

RESPOSTAS

1. CALL	Z,HERE	CALL	NZ,HERE
2. RET	Z	RET	NZ

CAPÍTULO 17

Pseudocódigos de Operação

Um pseudocódigo é uma instrução especial usada apenas em linguagem Assembler. Esta instrução não afeta diretamente o programa mas fornece instruções especiais ao programa montador Assembler sobre o que desejamos que seja feito.

A primeira que examinaremos é a instrução de pseudocódigo ORG. ORG é um símbolo mnemônico para ORIGINATE (dar origem a) e informa ao computador em que posição de memória desejamos que o programa seja iniciado. É necessário ter uma instrução ORG para todo programa em linguagem Assembler que se escreva. Em linguagem de máquina, isto não é necessário, já que você estaria utilizando um programa do tipo T-BUG, para ajudá-lo a codificar o seu programa. Você começaria a escrever o programa diretamente na posição de memória que desejasse.

Os pseudocódigos são colocados na coluna dos códigos normais de operação, na estrutura da instrução e todos eles são palavras reservadas. Isto significa que você não os poderá usar como rótulos em seu programa.

Se você fosse iniciar seu programa na posição 5000H, eis como deveria proceder, para comandar tal ação:

```
100     ORG     5000H
```

Quando o computador encontrar esta instrução, a qual tem que ser a primeira do programa, ele fará o posicionamento do programa na posição 5000H. Isto significa que a primeira instrução do seu programa estará na posição 5000H.

O pseudocódigo ORG também pode ter rótulo. Normalmente o programador utiliza o rótulo INÍCIO ou START ou algo do gênero para tal.

O segundo pseudocódigo é a instrução END. Ela deve ser a última instrução do programa. A instrução END é usada pelo montador Assembler para localizar a última posição de memória ocupada pelas instruções de um programa. Quando um programa é gravado em uma fita ou diskete, o primeiro registro da fita ou diskete, chamado de "header", contém os endereços de início e fim do programa, seguidos do nome do programa.

Exemplos do comando END:

```
END     5000H
```

ou

```
END     START
```

Quando você escreve um programa usando uma programação de suporte do tipo T-BUG e chega ao final, estando pronto para gravá-lo em uma fita, você usaria um comando P, ou seja, PUNCH (grave). Este seria um comando para o programa T-BUG e não para o seu programa em linguagem de máquina. O comando que você entraria pelo teclado seria da forma:

```
P   Endereço de Início    Endereço de Fim    Endereço de Início
    Nome do Programa
```

Se o endereço de início do programa é 5000H e o endereço da última posição de memória ocupada pelas instruções mais um for 5C00H e o nome do programa for SPORTS, o comando P seria escrito desta maneira:

```
P   5000    5C00    5000    SPORTS
```

Quando você entrar tal comando pelo teclado, o computador irá gravar o "header" da fita com o Endereço de Início, o Endereço de Fim, o Endereço de Início e o nome do arquivo, que, no caso, é SPORTS. Ele, então, copiará todo o conteúdo das posições de memória desde o endereço 5000H até 5C00H para a fita.

Quando se usa Assembler, a coisa é mais simples. Tudo o que você teria que fazer seria entrar o comando A ou ASSEMBLE, seguido do nome do arquivo. A operação de gravação seria realizada automaticamente pelo Assembler.

Quando o programa gravado é, de novo, carregado para a memória do computador, sob o comando SYSTEM, o computador lê o Endereço de Início, o Endereço de Fim, o Endereço de Início e, por último, o nome do arquivo.

Eis o que ele faz com estas informações: Primeiramente, ele verifica se o nome do arquivo, gravado na fita, bate com o nome que você forneceu pelo teclado. Se coincidir, ele começa a carga da informação contida na fita na posição de início ali especificada. Quando termina a carga, ele verifica se o endereço onde ele carregou a última informação da fita coincide com o endereço de fim especificado na fita. Se coincidir, ele, então, posiciona o ponto de início de execução no endereço de início gravado na fita.

O terceiro pseudocódigo é o EQUATE ou EQU. Esta instrução só pode ser usada uma única vez no programa, para um dado rótulo. Seu objetivo é colocar um valor associado a um rótulo. Por exemplo:

```
RÓTULO      PSEUDOCÓDIGO      VALOR
VÍDEO       EQU               3C00H
```

3C00H é o endereço inicial da área de memória, correspondente à primeira posição da tela do vídeo. Tendo a instrução acima colocado o valor 3C00H associado ao rótulo VÍDEO, de agora em diante, tudo que devemos fazer, a cada vez que queiramos nos referir à posição 3C00H, é usar a palavra VÍDEO.

Poderíamos, também, para facilitar as coisas ainda mais, ter usado o rótulo V, simplesmente, em lugar de VÍDEO. (Talvez não devesse ter chamado sua atenção para estas simplificações, pois temo que você comece a usar uma série de abreviações e acabe se confundindo ao longo do programa.)

A seguir veremos o pseudocódigo DEFL, que significa DEFINE LABEL (defina rótulo). Esta instrução também associa um valor a um rótulo. Ela pode ser repetida diversas vezes no programa, usando diferentes rótulos. Seu formato é o do seguinte exemplo:

```
LAB     DEFL    'D'
```

Deste ponto em diante, a cada vez que o computador encontrar o rótulo LAB ele o substituirá pela letra D e fará o que quer que especifique para o caso: exibir a letra na tela, armazená-la em algum outro local, ou imprimi-la em papel através de uma impressora.

Observe que, neste caso, D representa uma letra e não um valor numérico.

O quinto pseudocódigo é a instrução DEFB, ou seja DEFINE BYTE (defina byte). Ela pode ser usada tantas vezes quantas se deseje ao longo do programa. Esta instrução pode ser usada para, por exemplo, gerar um caracter de teclado sem que o usuário se preocupe com a configuração hexadecimal correspondente ao caracter. Isto é feito, por exemplo, usando-se a instrução desta forma:

```
ONE     DEFB    31H
```

De agora em diante, toda vez que o computador encontrar a palavra ONE, ela a tratará como o caracter 1 (31H é o hexadecimal correspondente ao dígito 1).

O próximo é o pseudocódigo DEFW, que significa DEFINE WORD (defina palavra). Esta instrução é útil para trabalhar com grupos de 16 bits e pode associar um valor hexadecimal a um rótulo. Por exemplo:

```
TEN     DEFW    10H
```

Neste exemplo, a instrução DEFW associa o valor 10H ao rótulo TEN. A partir daí, quando o computador encontrar a palavra TEN ele a tratará como 10H. Isto faria com que o computador, dada a instrução

```
ADD     A,TEN
```

somasse 10H ao conteúdo do registrador A.

O próximo pseudocódigo é o DEFS, ou DEFINE SPACE (defina espaço). Esta instrução diz ao computador para reservar certo espaço dentro do programa, para algum uso especial. Se você se lembra do que abordamos no capítulo sobre operações com relação à pilha ("Stack"), recordará que fi-

zemos menção quanto ao uso de "buffers", ou seja áreas de memória reservadas para certo uso. Eis uma maneira de criar um buffer:

```
BUFFER    DEFS    10H
```

Após esta instrução, 16 posições de memória serão reservadas para o buffer. A partir daí, quando comandarmos qualquer coisa a ser realizada pelo programa com relação a BUFFER, ela será realizada na área reservada denominada BUFFER.

Se nós inseríssemos DEFS 10H na posição 5000H e a rotulássemos de BUFFER, a área de 5000H a 500FH estaria protegida, reservada para um buffer denominado BUFFER.

É importante ressaltar aqui que o rótulo BUFFER é apenas isto mesmo, um rótulo e, assim sendo, em seu lugar poderiam ser utilizadas quaisquer outras palavras, tais como ÁREA, GATO, ou qualquer outra coisa.

O último pseudocódigo é a instrução DEFM, ou seja, DEFINE MESSAGE (defina mensagem). Ela é usada com um rótulo e pode ser repetida o quanto se desejar, desde que os rótulos tenham distintas identificações.

Seu objetivo é associar uma mensagem a um rótulo, de modo que, em lugar de lidar com toda a mensagem, você possa fazer menção, simplesmente, ao rótulo. Por exemplo, se você quer imprimir o nome John Doe a cada vez que o rótulo NAME seja encontrado no programa, como parte do operando, você faria o seguinte:

```
NAME    DEFM    'John Doe'
```

Observe que a mensagem deve ser colocada entre aspas.

Você deve atentar para o fato de que a instrução DEFM carrega o código hexadecimal da primeira letra da mensagem no endereço em que esta instrução se inicia. A seguir ela se dirige para o endereço seguinte e carrega aí a próxima letra da mensagem. Este processo é repetido até que a última letra da mensagem tenha sido carregada. Por exemplo, se o DEFM se inicia em 6000H e a mensagem tem um comprimento de 10 caracteres, o primeiro carácter será armazenado em 6000H, o segundo em 6001H e assim por diante, até que todos os 10 caracteres tenham sido armazenados.

Quando o programa manda carregar um registrador com uma mensagem, o registrador em questão é automaticamente apontado para o endereço onde se localiza o início da referida mensagem.

Tenho encontrado um grande uso para o DEFM na criação de listas de endereços e de listas de menu para programas. Na tabela 17-1 mostro alguns exemplos de como se valer deste recurso.

Agora, quando eu desejo imprimir um menu, ou emitir uma instrução que me permita fazer uma entrada a partir do teclado, tudo o que preciso fazer é carregar um par de registradores com um rótulo.

No próximo capítulo do livro, forneço uma sub-rotina para este fim.

Tabela 17-1. *Utilizando a instrução DEFM*

```
NAME        DEFM    'NAME'
ADR         DEFM    'ADDRESS'
CTY         DEFM    'CITY'
STA         DEFM    'STATE'
ZIP         DEFM    'ZIP CODE'
MEN         DEFM    'MENU'
F1          DEFM    'FIND ALL PERSONS
                    WITH THE
SAME LAST NAME'
F2          DEFM    'FIND ALL ZIP CODES
                    IN AN
AREA'
LIST        DEFM    'LIST ALL NAMES IN
                    FILE'
```

Existe, ainda, um outro ponto importante que deve ser ressaltado. Nós podemos fazer um EQUATE do comprimento da mensagem e usá-lo para fazer a carga de um outro par de registradores com o comprimento da mensagem. Nós sabemos que não podemos usar o EQUATE mais de uma vez com o mesmo rótulo. Os rótulos nas instruções de pseudocódigo DEFM seguem a mesma regra. Tomemos o exemplo do rótulo ZIP. A mensagem era 'ZIP CODE'. O comprimento da mensagem, incluindo o espaço, é 8. Poderíamos usar o DEFB 8, ou poderíamos carregar o contador de bytes (um par de registradores) com 8. Mas há uma maneira mais fácil de realizar isto.

Por que não fazer um EQUATE do comprimento de ZIP? Eis como fazemos isto:

```
ZIPL    EQU     $-ZIP
```

Quando o computador recebe a instrução para carregar o contador de bytes (par BC) com o comprimento da mensagem ZIP,

```
LD      BC,ZIPL
```

ele o encontrará associado (através do EQUATE) ao rótulo ZIPL e o carregará no par de registradores BC.

Para esclarecer os detalhes desta instrução EQUATE, é necessário explicar que o símbolo $ significa um encadeamento ou mensagem (que poderia se constituir de letras, números, ou ambos). O carácter indica comprimento, e ZIP é, obviamente, o rótulo a cujo tamanho queremos realizar a associação de um outro rótulo.

Esta é uma boa maneira de se lembrar como escrever a instrução: Na verdade, o caracter não significa comprimento, ele é uma pontuação necessária ao texto da instrução. Todavia, se você se lembrar deste traço como comprimento, isto lhe ajudará a memorizar o formato da instrução.

QUESTÕES

1. O pseudocódigo . . . . . informa ao computador onde iniciar.
2. O pseudocódigo . . . . . informa ao computador onde termina o programa.
3. O pseudocódigo . . . . . informa ao computador que associe um valor a um rótulo.
4. O pseudocódigo . . . . . informa ao computador que valor atribuir (caracter) a um rótulo.
5. O pseudocódigo . . . . . informa ao computador para dar um valor numérico a um rótulo.
6. O pseudocódigo . . . . . informa ao computador para dar um valor hexadecimal a um rótulo.
7. O pseudocódigo . . . . . informa ao computador que reserve espaço.
8. O pseudocódigo . . . . . informa ao computador para que associe uma mensagem a um rótulo.

RESPOSTAS

1. ORG
2. END
3. EQU
4. DEFL
5. DEFB
6. DEFW
7. DEFS
8. DEFM

CAPÍTULO 18

Programas de Exemplo

Neste capítulo veremos alguns programas de exemplo. É importante observar que os comentários nas linhas das instruções não são aqui, de início, apresentados devido aos problemas de espaço para os programas. Contudo, nas páginas seguintes a estes primeiros exemplos, você encontrará comentários em detalhe, listados linha a linha.

Um outro aspecto a ressaltar é que os programas aqui apresentados são apenas exemplos e não se destinam a um uso específico. Se você pedisse a dez pessoas que escrevessem o mesmo programa, provavelmente obteria dez programas bastante diferentes, porém que realizariam a mesma coisa. Sendo esta a realidade, os programas aqui mostrados são apenas exemplos de como se fazer determinada tarefa.

Recomendo com ênfase que você se exercite com estes programas e use maneiras diferentes de executar as funções mostradas nos programas.

SUB-ROTINA PARA LIMPAR A TELA

Esta sub-rotina pode ser usada para limpar a tela do vídeo. Ela é executada através de um CALL para CLS. A parte do programa que limpa a tela é localizada em 7000H mas poderia ser localizada em qualquer outra posição. O programa é mostrado na Fig. 18-1.

A linha 100 estabelece o endereço de ORiGem para o programa em 5000H.

A linha 110 chama (CALL) a sub-rotina rotulada como CLS.

A linha 130 é o início do restante do programa. A posição de memória para a instrução inicial nessa linha seria 5003H.

*N.T. – O ponto e vírgula indica que a linha é um simples comentário e não deve ser processada pelo computador.

```
5000                00100           ORG     5000H
5000    CD0070      00110   START   CALL    CLS
                    00130   ;THE REST OF THE PROGRAM GOES HERE
7000                00500           ORG     7000H
7000    21003C      00510   CLS     LD      HL,3C00H
7003    11013C      00520           LD      DE,3C01H
7006    010004      00530           LD      BC,400H
7009    3620        00540           LD      (HL),20H
700B    EDB0        00550           LDIR
700D    C9          00560           RET
5000                00570           END     START
00000   TOTAL ERRORS
CLS     7000
START   5000
```

Fig. 18-1. *Sub-rotina para Limpeza da Tela.*

A linha 500 inicia a sub-rotina CLS. A instrução ORG nesta linha não é necessária em um uso real deste programa. Eu a coloquei neste exemplo para deslocar os comandos para a posição 7000H. Num uso real, você não usaria o ORG nesta linha e sim, simplesmente, saltaria para a linha 510.

A linha 510 inicia com o rótulo CLS. Ele é seguido por um comando que manda carregar o par de registradores HL com o valor 3C00H, que é o início da área de memória correspondente à tela do vídeo.

A linha 520 instrui ao computador para carregar o par DE com 3C01H.

A linha 530 manda carregar o contador de bytes, o par BC, com 400H. 400H é o número de bytes a serem preenchidos.

A linha 540 manda carregar o valor 20H (código hexa do caracter espaço) na posição apontada pelo par HL.

A linha 550 é o comando LDIR, ou seja Load Decrement Increment Repeat. Como vimos quando estudamos este comando, ele faz com que o conteúdo da posição apontada pelo par HL seja carregado na posição apontada pelo par DE. A seguir, o par DE é incrementado e o conteúdo do par BC é decrementado. Se, ao ser decrementado, o par BC se tornar zerado, então a próxima instrução é executada. Se não, o comando é executado novamente.

A linha 560 manda o computador retornar ao programa principal e retomar a execução na linha seguinte daquele programa. Neste exemplo não há nenhuma. Para exercitar, você poderia querer colocar um comando JUMP na linha 130 para mandar o programa saltar de volta para o rótulo START, (5000H) ou para a posição 5003H. Tal salto causaria uma condição de "loop" sem saída, com o programa saltando do fim da instrução para a posição 5000H ou 5003H, até que você pressione a tecla de restauração do sistema, que interromperá o processamento.

A linha 570 contém a necessária instrução END. Observe que o operando nesta linha é START. Este operando não é necessário mas ajuda ao montador Assembler a descarregar a versão montada do programa em fita.

UMA MANEIRA MAIS RÁPIDA DE LIMPAR A TELA

A figura 18-2 mostra uma forma mais rápida de limpar a tela. Este pro-

grama usa uma rotina em ROM, que é a mesma que é executada quando se pressiona a tecla CLEAR (limpe), ou se entra o comando CLS em BASIC.

A linha 100 estabelece o início do programa em 5000H.

A linha 110 chama a sub-rotina denominada CLS.

A linha 120 é o ponto no qual o restante do programa é inserido. O endereço do início do restante do programa é 5003H.

A linha 500 estabelece a origem para a sub-rotina CLS. De novo, aqui, como no exemplo anterior, esta linha não é necessária em um uso real do programa.

A linha 510 é o início da sub-rotina CLS. Aí encontramos o rótulo CLS e o comando CALL seguido do endereço da rotina em memória ROM que faz a limpeza da tela. O endereço na memória ROM, neste caso, é 01C9H. Nesta rotina, gravada na memória ROM, não há retorno automático.

```
5000                00100              ORG     5000H
5000    CD0070      00110    START     CALL    CLS
5003                00120;THE REST OF THE PROGRAM GOES HERE
7000                00500              ORG     7000H
7000    CDC901      00510    CLS       CALL    01C9H
7003    C9          00520              RET
5000                00530              END     START
00000   TOTAL ERRORS
CLS     7000
START   5000
```

Fig. 18-2. *Uma Limpeza mais Rápida da Tela.*

A linha 520 instrui ao computador para retornar ao programa principal.

A linha 530 é a necessária instrução END.

QUICK PRINTER II: CORREÇÃO NO "DRIVER" DE LINGUAGEM DE MÁQUINA

No manual do programa Quick Printer II (Impressora Veloz II) há uma sub-rotina que pode ser usada para enviar dados do computador para a impressora. Este programa apresenta erro e a Fig. 18-3 apresenta uma versão corrigida.

Primeiramente, há um termo novo, com o qual você precisa se familiarizar: *"driver"* (utilizamos o termo em inglês por ser de uso corrente entre nós, no Brasil). Um driver (pronuncia-se "Draiver") é uma sub-rotina que transfere dados da UCP para um dispositivo externo, tal como uma impressora, um dispositivo de disco, uma unidade de fita magnética, uma tela de vídeo etc.

A entrada para esta sub-rotina é através de um CALL para PTRDVR. O registrador C deverá conter o byte a ser enviado, antes que o CALL seja executado.

A linha 100 estabelece o endereço de início para a sub-rotina.

A linha 110 carrega o registrador A com o estado da impressora. Tal estado é encontrado testando-se o conteúdo de memória da localização 37E8H. Se a impressora não está ocupada, o conteúdo desta posição será zero.

A linha 120 mascara a metade superior do conteúdo de 37E8H, para lhe dar um valor verdadeiro.

A linha 130 compara o conteúdo do registrador A com o código hexa do zero (30H).

A linha 140 faz o computador saltar (Jump Relativo) para o início do driver se o conteúdo do registrador A não for zero. Esta é a linha em que o manual do Quick Printer apresenta um erro. A instrução no manual fará o programa saltar para o início da sub-rotina independentemente do estado da impressora. Lá não há nenhuma condição imposta para o Jump Relativo.

```
5000                00100           ORG     5000H
5000    3AE837      00110   PTRDVR  LD      A,(37E8H)
5003    E6F0        00120           AND     0F0H
5005    FE30        00130           CP      30H
5007    20F7        00140           JR      NZ,PTRDVR
5009    79          00150           LD      A,C
500A    32E837      00160           LD      (37E8H),A
500D    C9          00210           RET
5000                00220           END     PTRDVR
00000   TOTAL ERRORS
PTRDVR  5000
```

Fig. 18-3. *Correção no Driver de Impressão.*

```
500D    0605        00170           LD      B,5
500F    05          00180   LOOP    DEC     B
5010    78          00190           LD      A,B
5011    20FC        00200           JR      NZ,LOOP
5013    C9          00210           RET
```

Fig. 18-4. *Atraso de Tempo.*

A linha 150 carrega o conteúdo do registrador C no registrador A.

A linha 160 carrega o conteúdo da localização 37E8H com o conteúdo do registrador A. Isto faz com que o carácter seja impresso.

A linha 210 manda o computador retornar ao programa principal.

A linha 220 é a necessária instrução END.

Observe que as linhas 170 a 200 estão faltando neste programa. A razão para tal é que este driver é projetado para enviar apenas um byte de dados para a impressora. Para enviar mais de um byte para a impressora, é necessário introduzir um atraso.

Após os dados terem sido enviados à impressora, você necessitará enviar um comando de retorno do carro. O código em hexa do retorno de carro é 0DH. Portanto, você terá que carregar o registrador C com 0DH e chamar a

sub-rotina PTRDVR de novo, para forçar a impressora a parar de receber dados e imprimir apenas o que ela já tenha em sua memória intermediária (que, em geral, tem uma capacidade de 32 bytes). Se tal capacidade se esgotar, antes que seja enviado o 0DH, a impressora imprimirá o que ela tenha em sua memória e voltará a aceitar novos dados para impressão. A figura 18-4 é um atraso de tempo introduzido para se poder carregar mais de um byte de cada vez.

A linha 170 comanda a carga do registrador B com o valor 5. Este é o registrador que utilizaremos para contar o atraso. Como você pode verificar, o endereço desta instrução, 500DH, é o endereço onde se encontra a instrução RET no driver anterior. Isto se deve a que o atraso deve ocorrer antes que ocorra o retorno. Portanto, devemos substituir a instrução de retorno pela primeira instrução do atraso.

A linha 180 é a linha onde o "loop" (volta) de atraso realmente se localiza. Aqui, o registrador B é decrementado, preparando seu valor para o próximo passo.

A linha 200 ordena ao computador para saltar (Jump Relativo) para LOOP se o conteúdo do registrador B não for zero. Se tal ocorrer, o conteúdo do registrador B será decrementado de novo e o loop será repetido. Por outro lado, se o conteúdo de B for zero, a próxima instrução será executada.

A linha 210 é a necessária instrução de retorno e atua da mesma forma que no driver original. A única diferença é a localização da instrução.

A versão "montada", isto é, após o processamento pelo montador Assembler, também incluirá o endereço do rótulo LOOP.

ROTINA PARA PREENCHER A TELA COM QUALQUER CARACTER A PARTIR DO TECLADO

Esta rotina, mostrada na Fig. 18-5, permitirá que você entre qualquer caracter, a partir do teclado, com o qual a tela do vídeo será completamente preenchida.

A linha 100 estabelece o ponto de início em 5000H.

A linha 110 invoca a rotina em ROM para limpar a tela.

A linha 120 invoca uma outra rotina gravada em ROM. Esta rotina faz uma varredura do teclado para ver se houve alguma entrada e retorna o valor da tecla, eventualmente pressionada, armazenado no registrador A. Se nenhuma tecla tiver sido pressionada, então o registrador A conterá 00H.

A linha 130 verifica o valor de A para ver se ele é zero. Se tal suceder, o flag Z do registrador F será ligado.

A linha 140 instrui o computador para saltar de volta para o rótulo LOOP se o flag Z estiver ligado.

A linha 150 carrega o par de registradores HL com o endereço inicial da área de memória da tela do vídeo.

A linha 160 carrega o par de registradores DE com o endereço da segunda posição da área de memória da tela, 3C01H.

A linha 170 carrega o par de registradores BC, o contador de bytes, com o número de localizações a serem preenchidas com o caracter.

```
5000                    00100           ORG     5000H
5000    CDC901          00110   START   CALL    01C9H
5003    CD4900          00120   LOOP    CALL    049H
5006    FE00            00130           CP      00H
5008    CA0350          00140           JP      Z,LOOP
500B    21003C          00150           LD      HL,3C00H
500E    11013C          00160           LD      DE,3C01H
5011    010004          00170           LD      BC,400H
5014    77              00180           LD      (HL),A
5015    EDB0            00190           LDIR
5017    C30350          00200           JP      LOOP
5000                    00210           END     START
```

Fig. 18-5. *Rotina de Preenchimento da Tela.*

A linha 180 carrega o valor do registrador A no endereço apontado pelo par HL.

A linha 190 é a instrução LDIR. Esta instrução faz com que o conteúdo da posição de memória, especificada pelo par HL, seja carregado na localização de memória especificada pelo par DE. Ao mesmo tempo, faz com que o conteúdo do par BC seja decrementado. Se o decremento do par BC o fizer ficar com conteúdo zero, o programa continuará na linha seguinte. Se não, a instrução será repetida.

A linha 200 faz o contador de instruções saltar para trás, para o rótulo LOOP, localizado em 5003H. Esta ação, lhe permitirá entrar com outro caracter e repetir o processo. Até que uma tecla seja pressionada, o valor retornado no registrador A pelo CALL dado à rotina em 049H, será 0, o que fará o programa retornar a 5003H.

A linha 210 é a instrução obrigatória END. Aqui, mais uma vez, usamos o operando START por razões de segurança.

ROTINA PARA ENTRAR UM DETERMINADO NÚMERO DE CARACTERES A PARTIR DO TECLADO

Esta rotina (Fig. 18-6) lhe permitirá carregar um número específico de caracteres, a partir do teclado. Ela continuará a executar isto até que uma dentre duas condições ocorra:

1. A tecla de interrupção (break) seja pressionada.
2. A tecla de entrada seja pressionada.

Nós iremos armazenar os caracteres na tela para demonstrar este ponto. Duas coisas interessantes são aqui constatadas. A primeira é que, quando o

```
5000                    00100   ORG     5000H
5000    21003C          00110   AUTO    LD      HL,3C00H
5003    060A            00120           LD      B,10
5005    CD4000          00130           CALL    40H
5008    CDC901          00140           CALL    01C9H
500B    C30050          00150           JP      AUTO
5000                    00160           END
00000   TOTAL ERRORS
AUTO    5000
```

Fig. 18-6. *Entrada de Caracter a partir do Teclado.*

último caracter é entrado e um outro é pressionado, o último caracter será substituído por esta entrada adicional. A segunda é que a tecla de retrocesso é reconhecida.

A linha 100 determina o início do programa.

A linha 110 contém o rótulo AUTO e também comanda a carga do par HL com o endereço inicial da área do vídeo.

A linha 120 carrega o registrador B com o número de espaços a serem preenchidos. Você pode usar até 255 (FFH). O registrador B é usado como contador de bytes.

A linha 130 executa a chamada à rotina em ROM que faz varredura e carga.

A linha 140 faz um CALL para a rotina em 01C9H que limpa a tela quando as teclas de interrupção ou entrada forem pressionadas.

A linha 150 faz com que o programa desvie de volta para o início, em AUTO, para os próximos 10 caracteres.

A linha 160 é a instrução mandatória END.

ROTINA PARA TRANSFERIR O CONTEÚDO DE UMA POSIÇÃO DE MEMÓRIA PARA OUTRA

Esta rotina, mostrada na Fig. 18-7, pode ser usada para mover o conteúdo de uma localização de memória para outra. Para demonstrar isto, moveremos o conteúdo de um buffer para a tela.

A linha 100 estabelece o início do programa.

A linha 110 invoca a função em ROM que limpa a tela. Ela tem um rótulo denominado START.

A linha 120 instrui o computador para que carregue o par DE com o endereço inicial da tela.

A linha 130 manda carregar o par HL com o endereço do buffer. Neste programa, o rótulo BUFFER está localizado em 5011H e, desta forma, o par HL é carregado com 5011H.

A linha 140 carrega o contador de bytes, o par BC, com o número de posições cujo conteúdo deve ser transferido. Neste caso, queremos transferir

os conteúdos de 16 posições. Deve-se notar que o conteúdo dessas 16 posições, as quais constituem o buffer, não será alterado. Você pode comprovar isto carregando e rodando o programa com um T-BUG. Para tal, você pode estabelecer um ponto de quebra ou parada (break), usando o comando B do T-BUG na posição 500EH. Quando o programa encontra o ponto de break, ele retornará um caracter #. Ao mesmo tempo, você deve entrar um M seguido do endereço do buffer, 5011H. Você verá que o conteúdo das 16 posições do buffer não foi alterado.

A linha 150 é o comando LDIR. Este comando é o coração da rotina. Ele executa a transferência.

A linha 160 é a linha de loop (volta sobre si mesmo).

Quando a transferência está completa, o salto para LOOP evitará que o programa continue o processamento através de outras instruções contíguas na pilha. Este seria mais um bom exemplo do que dissemos anteriormente sobre um programa vir a executar uma instrução inválida. Se você examinar a lista numérica de códigos de operação de 41H até 50H, verá que se trata de instruções de carga para os registradores B, C e D. Se o loop não estivesse ali situado e o contador de instruções não tivesse, de alguma forma, sido redirecionado para a posição 5003H, essas cargas teriam sido realizadas e o computador penetraria em um espaço não previsto. Ele, eventualmente, alcançaria um CALL ou um JUMP para a posição 00H. Isto faria com que ele retornasse ao estado inicial de quando é ligado e imprimisse a mensagem MEMORY SIZE.

```
5000                00100                ORG     5000H
5000  CDC901        00110  START         CALL    01C9H
5003  11003C        00120                LD      DE,3C00H
5006  211150        00130                LD      HL,BUFFER
5009  012400        00140                LD      BC,16
500C  EDB0          00150                LDIR
500E  C30E50        00160  LOOP          JP      LOOP
5011  41            00170  BUFFER        DEFM    'ABCDEFGHIJKLMNOP'
5012  42
5013  43
5014  44
5015  45
5016  46
5017  47
5018  48
5019  49
501A  4A
501B  4B
501C  4C
501D  4D
501E  4E
501F  4F
5020  50
5000                00180                END     START
00000  TOTAL ERRORS
BUFFER 5011
LOOP   500E
START  5000
```

Fig. 18-7. *Transferência de Conteúdos de Posições de Memória.*

A linha 170 não só cria o rótulo BUFFER como também cria os dados para aquelas posições. Neste caso nós usamos o pseudocódigo DEFM, mas poderíamos ter armazenado a informação no buffer a partir do teclado ou de qualquer outro dispositivo de entrada.

A linha 180 contém a necessária instrução END.

PROGRAMA PARA IMPRIMIR UMA LISTA DE ENDEREÇOS

Esta rotina, mostrada na Fig. 18-8, imprime uma lista para endereçamento postal. Neste exemplo, nós iremos guardar os dados em um buffer de dados. Em um uso real desta aplicação, você, provavelmente, usaria um dispositivo de Entrada/Saída (E/S) para armazenar a informação. Este programa também utiliza delimitadores, para separar os diversos conjuntos de dados. Um delimitador é um caracter que você não usará, em hipótese alguma, como pertencente ao conjunto de dados. Neste caso, nós usaremos o símbolo) para indicar o fim da lista de dados para cada pessoa e o símbolo (para separar os diversos elementos de dados para uma dada pessoa.

A fim de encurtar a descrição do programa, nós passaremos por cima daquelas partes que já abordamos no programa anterior e nos concentraremos nos elementos novos aqui introduzidos.

A linha 100 define o início do programa. A linha 110 chama a rotina ROM que limpa a tela.

As linhas de 120 a 360 imprimem a mensagem 'NAME' 'ADDRESS' 'STREET' 'CITY' 'STATE' e 'ZIP CODE'. Elas também colocam as posições do vídeo no par de registradores DE e o comprimento da mensagem no par BC. Depois disto feito, a instrução LDIR é executada. As mensagens são numeradas de M1 a M6. As posições da tela para estas linhas são identificadas de V1 a V6. Os tamanhos das mensagens para estas linhas sofrem um EQUATE para o tamanho real das mensagens. Estes comprimentos têm como rótulos os nomes de M1L a M6L. Este programa se vale de uma técnica especial para posicionar cada localização na tela do vídeo, que discutiremos mais à frente.

A linha 370 é o início da parte do programa que exibe os dados na tela. Aqui, o par HL é carregado com o rótulo (endereço) DATA. Este rótulo está localizado no endereço no qual está armazenado o primeiro byte de dados. Como dissemos anteriormente, os dados neste programa estão armazenados através de uma instrução DEFM. Contudo, se desejássemos, poderíamos carregar os dados, a partir de uma fonte externa, armazenando-os em um buffer. Se você decidir fazer isto, tudo o que será necessário é a criação de uma subrotina para carregar os dados no endereço especificado pelo rótulo DATA. Isto feito, você poderia continuar retomando o programa a partir da linha 370. *Atenção*: algum tipo de alteração seria necessário, já que este programa está preparado para aceitar um dado de cada vez. Eu sugeriria que se criasse um terceiro delimitador, para indicar o fim do conjunto total de dados. Introduza, então, uma rotina para testar o encontro desse delimitador.

```
5000                00100           ORG     5000H
5000    CDC901      00110   START   CALL    01C9H
5003    11003C      00120           LD      DE,V1
5006    21AA50      00130           LD      HL,M1
5009    010400      00140           LD      BC,M1L
500C    EDB0        00150           LDIR
500E    11403C      00160           LD      DE,V2
5011    21AE50      00170           LD      HL,M2
5014    010600      00180           LD      BC,M2L
5017    EDB0        00190           LDIR
5019    11803C      00200           LD      DE,V3
501C    21B550      00210           LD      HL,M3
501F    010600      00220           LD      BC,M3L
5022    EDB0        00230           LDIR
5024    11C03C      00240           LD      DE,V4
5027    21BB50      00250           LD      HL,M4
502A    010400      00260           LD      BC,M4L
502D    EDB0        00270           LDIR
502F    11003D      00280           LD      DE,V5
5032    21BF50      00290           LD      HL,M5
5035    010500      00300           LD      BC,M5L
5038    EDB0        00310           LDIR
503A    11403D      00320           LD      DE,V6
503D    21C450      00340           LD      HL,M6
5040    010800      00350           LD      BC,M6L
5043    EDB0        00360           LDIR
5045    21CC50      00370           LD      HL,DATA
5048    110A3C      00380           LD      DE,V7
504B    7E          00390   LP1     LD      A,(HL)
504C    FE28        00400           CP      28H
504E    CA5750      00410           JP      Z,LP2
5051    12          00420           LD      (DE),A
5052    13          00430           INC     DE
5053    23          00440           INC     HL
5054    C34B50      00450           JP      LP1
5057    23          00460   LP2     INC     HL
5058    114A3C      00470           LD      DE,V8
505B    7E          00480   LP2A    LD      A,(HL)
505C    FE28        00490           CP      28H
505E    CA6750      00500           JP      Z,LP3
5061    12          00510           LD      (DE),A
5062    13          00520           INC     DE
5063    23          00530           INC     HL
5064    C35B50      00540           JP      LP2A
5067    118A3C      00550   LP3     LD      DE,V9
506A    23          00560           INC     HL
506B    7E          00570   LP3A    LD      A,(HL)
506C    FE28        00580           CP      28H
506E    CA7750      00590           JP      Z,LP4
5071    12          00600           LD      (DE),A
5072    13          00610           INC     DE
```

Fig. 18-8. *Lista de Endereçamento Postal.*

```
5073    23        00620          INC     HL
5074    C36B50    00630          JP      LP3A
5077    11CA3C    00640    LP4   LD      DE,V10
507A    23        00650          INC     HL
507B    7E        00660    LP4A  LD      A,(HL)
507C    FE28      00670          CP      28H
507E    CA8A50    00680          JP      Z,LP5
5081    12        00690          LD      (DE),A
5082    13        00700          INC     DE
5083    23        00710          INC     HL
5084    C37B50    00720          JP      LP4A
5087    23        00730    LP5   INC     HL
5088    110A3D    00740          LD      DE,V11
508B    7E        00750    LP5A  LD      A,(HL)
508C    FE28      00760          CP      28H
508E    CA5097    00770          JP      Z,LP6
5091    12        00780          LD      (DE),A
5092    13        00790          INC     DE
5093    23        00800          INC     HL
5094    C38B50    00810          JP      LP5A
5097    23        00820    LP6   INC     HL
5098    114A3D    00830          LD      DE,V12
509B    7E        00840    LP6A  LD      A,(HL)
509C    FE29      00850          CP      29H
509E    CAA750    00860          JP      Z,STOP
50A1    12        00870          LD      (DE),A
50A2    13        00880          INC     DE
50A3    23        00890          INC     HL
50A4    C39B50    00900          JP      LP6A
50A7    C3A750    00910    STOP  JP      STOP
3C00              00920    V1    EQU     3C00H
3C40              00930    V2    EQU     V1+64
3C80              00940    V3    EQU     V2+64
3CC0              00950    V4    EQU     V3+64
3D00              00960    V5    EQU     V4+64
3D40              00970    V6    EQU     V5+64
3C0A              00980    V7    EQU     V1+10
3C4A              00990    V8    EQU     V2+10
3C8A              01000    V9    EQU     V3+10
3CCA              01010    V10   EQU     V4+10
3D0A              01020    V11   EQU     V5+10
3D4A              01030    V12   EQU     V6+10
50AA    4E        01040    M1    DEFM    'NAME'
50AB    41
50AC    4D
50AD    45
50AE    41        01050    M2    DEFM    'ADDRESS'
50AF    44
50B0    44
50B1    52
50B2    45
```

Fig. 18-8. *Continuação da pág. 127.*

```
50B3    53
50B4    53
50B5    53          01060   M3      DEFM    'STREET'
50B6    54
50B7    52
50B8    45
50B9    45
50BA    54
50BB    43          01070   M4      DEFM    'CITY'
50BC    49
50BD    54
50BE    59
50BF    53          01080   M5      DEFM    'STATE'
50C0    54
50C1    41
50C2    54
50C3    45
50C4    5A          01090   M6      DEFM    'ZIP CODE'
50C5    49
50C6    50
50C7    20
50C8    43
50C9    4F
50CA    44
50CB    45
0004                01100   M1L     EQU     $-M1
0007                01110   M2L     EQU     $-M2
0006                01120   M3L     EQU     $-M3
0004                01130   M4L     EQU     $-M4
0005                01140   M5L     EQU     $-M5
0008                01150   M6L     EQU     $-M6
50CC    59          01160   DATA    DEFM    'YOUR NAME
(123(S. WEST AVE(ANYTOWN(CA(12345)'
50CD    4F
50CE    55
50CF    52
50D0    20
50D1    4E
50D2    41
50D3    4D
50D4    45
50D5    28
50D6    31
50D7    32
50D8    33
50D9    28
50DA    53
50DB    2E
50DC    2E
50DD    57
50DE    45
```

```
50DF      53
50E0      54
50E1      20
50E2      41
50E3      56
50E4      45
50E5      28
50E6      41
50E7      4E
50E8      59
50E9      54
50EA      4F
50EB      57
50EC      4E
50ED      28
50EE      43
503F      41
50F0      28
50F1      31
50F2      32
50F3      33
50F4      34
50F5      35
50F6      29
5000              01170    END       START
00000     TOTAL ERRORS
LP6A      509B
V12       3D4A
LP6       5097
LP5A      508B
V11       3D0A
STOP      50A7
LP5       50A8
LP4A      50B7
V10       3CCA
LP4       5077
LP3A      506B
V9        3CBA
LP3       5067
LP2A      505B
V8        3C4A
LP2       5057
LP1       504B
V7        3C0A
DATA      50CF
M6L       0008
M6        50C7
V6        3D40
M5L       0005
M5        50C2
V5        3D00
```

Fig. 18-8. *Continuação da pág. 129.*

```
M4L     0004
M4      50BE
V4      3CC0
M3L     0006
M3      50BB
V3      3CB0
M2L     0007
M2      50B1
V2      3C40
M1L     0004
M1      50AD
V1      3C00
START   5000
```

Fig. 18-8. *Continuação da pág. 130.*

A linha 380 carrega o par DE com a localização da tela, onde o primeiro elemento de dados deve ser exibido.

A linha 390 inicia o loop para testar o primeiro delimitador. Aqui, o conteúdo da posição apontada pelo par HL é carregado no registrador A.

A linha 400 testa o conteúdo do registrador A, comparando-o com 28H. 28H é o código hexa do símbolo (. Ele atua como um delimitador de fim de linha.

A linha 410 comanda um salto para o início da rotina de pesquisa exibição (caso o delimitador seja encontrado).

A linha 420 carrega o caracter do registrador A (aquele que obtivemos pelo endereçamento dado pelo par HL) no endereço especificado pelo par DE.

A linha 430 faz o par de registradores DE ser incrementado.

A linha 440 faz o par de registradores HL ser incrementado.

A linha 450 faz o programa saltar de volta ao início do loop, que, neste caso, é LP1 e executa o teste para o próximo caracter. Este processo é repetido da linha 450 à linha 900. A única diferença é que, na linha 850, é feito um teste para o carácter), que é 29H. A linha 860 comanda um salto para a linha 910, ou seja, o rótulo STOP, caso a comparação seja verdadeira.

A linha 910 coloca o computador em um loop sem fim, de proteção, ao fim do programa.

A linha 920 estabelece o valor dado a V1 como 3C00H. Este valor também se encontra no par de registradores DE. Existem várias maneiras de se posicionar o endereço, no qual se deseja exibir uma informação. Eu escolhi a presente para lhe dar um exemplo das variedades de possibilidades. Além disso, esta é uma forma conveniente de indicar as linhas e tabulação sem ter que fazer cálculos à mão ou somar 3 ou 4 vezes ao endereço hexadecimal.

A linha 930 inicia nosso atalho para estabelecer os pontos de início para os dados de M2 a M6. Já fizemos um EQUATE do valor de V1 em 3C00H. Podemos, agora, posicionar V2 no início da próxima linha, fazendo um EQUATE de V2 como sendo V1 + 64 (há 63 caracteres por linha). O mesmo se fará para V3, com EQUATE para V2 + 64. Isto é permissível desde que já

tenhamos estabelecido o valor do rótulo sobre o qual estamos realizando a soma do deslocamento.

A linha 980 dá início à função de tabulação. Você poderia usar a rotina já pré-programada em ROM para tabulação, porém lembre-se de que ela tabula em incrementos de 8 posições. Aqui, nós desejamos incrementos de tabulação de 10 em 10. Os valores de V1 até V6 já foram estabelecidos e nós queremos, agora, tabular a linha em espaços de 10 posições cada. O que devemos fazer é estabelecer diferentes rótulos para linha e tabulação em particular. Fazemos o EQUATE de tal rótulo ao valor do rótulo da linha + 10. Como você pode constatar, V7, a localização da tela onde queremos exibir o nome da pessoa, sofreu um EQUATE a V1, o endereço da linha na tela + 10, o número de espaços a saltar. Não é necessário usar V1 como rótulo para a primeira linha do vídeo. Fizemos isto apenas para facilitar o acompanhamento do raciocínio. Eu disse para mim mesmo, o V significará vídeo e o número, o número da linha no vídeo. Já que havia apenas 6 categorias que desejava exibir, eu iniciei as linhas de informação na sétima linha. Agora que enxergamos sob este ponto de vista a codificação dos rótulos, vejo que seria igualmente simples, se não mais fácil, usar I1 em lugar de V7. O I significaria informação. Se eu desejasse bastante representatividade, poderia ter usado o rótulo LI1, significando linha de informação 1.

A linha 1040 é a linha onde se localiza o endereço do início da área de dados da categoria de cabeçalhos. Aqui definimos o rótulo M1, usando o pseudocódigo DEFM como 'NAME'. Este também é o caso das linhas 1050 até 1090.

As linhas 1100 até 1150 usam o pseudocódigo EQU para estabelecer o comprimento dos valores dos rótulos M1 até M6. Para lembrar melhor, usei, aqui, o rótulo e adicionei a letra L para indicar que se trata do comprimento (length) do rótulo. Isto facilitou as coisas, quando eu estava escrevendo o programa, já que, ao usar V1 e M1, eu não usaria um valor errado para carregar o par BC. Intrinsecamente, eu me lembraria que deveria fazer a carga de BC com M1L, ou seja, o comprimento da mensagem 1.

A linha 1160 estabelece o endereço para o início da nossa área de buffer. Este buffer tem o rótulo de DATA (dados). Para fins de demonstração, usei o pseudocódigo DEFM para colocar os dados no buffer.

A linha 1700 é a necessária instrução END.

COPIA DA MEMÓRIA

Este programa (Fig. 18-9) lhe fornecerá uma cópia impressa do conteúdo de um grupo especificado de posições de memória. Além dos conteúdos, as respectivas posições (endereços) são também impressas. Após a carga do programa, sob o comando SYSTEM, entre um caracter / e, a seguir, pressione a tecla de entrada.

O programa responderá perguntando-lhe o endereço a partir do qual você deseja iniciar a impressão. Entre o valor do endereço em hexa. O programa,

então, lhe perguntará em que endereço você deseja que termine a listagem das posições. Entre o endereço final + 1, em hexa, sente-se e espere. A impressora e a UCP farão o resto.

Quando você executar este programa pela primeira vez, experimente usar 3C00H para o endereço inicial e 3C0BH para o final. Depois da impressão, compare os valores impressos em hexa, para aquelas localizações, com os correspondentes códigos hexa dos 10 primeiros caracteres exibidos na tela do vídeo.

A linha 100 estabelece o início do programa.

A linha 110 chama a rotina em ROM de limpeza da tela.

```
5000                00100           ORG     5000H
5000    CDC901      00110           CALL    01C9H
5003    11003C      00120           LD      DE,3C00H
5006    216152      00130           LD      HL,MESS1
5009    011400      00140           LD      BC,20
500C    EDB0        00150           LDIR
500E    D5          00160   PRINT1  PUSH    DE
500F    CDAF51      00170           CALL    SCAN
5012    D1          00180           POP     DE
5013    12          00190           LD      (DE),A
5014    13          00200           INC     DE
5015    78          00210   P1A     LD      A,B
5016    FE00        00220           CP      0
5018    281F        00230           JR      Z,P1B
501A    210000      00240           LD      HL,0000H
501D    225B52      00250           LD      (STABUF),HL
5020    225D52      00260           LD      (ENDBUF),HL
5023    225F52      00270           LD      (TOTBUF),HL
5026    2A5B52      00280           LD      HL,(STABUF)
5029    D5          00290           PUSH    DE
502A    ED5B8952    00300           LD      DE,(TABLE)
502E    19          00310   P1L1    ADD     HL,DE
502F    05          00320           DEC     B
5030    78          00330           LD      A,B
5031    FE00        00340           CP      0
5033    20F9        00350           JR      NZ,P1L1
5035    22B552      00360           LD      (STABUF),HL
5038    D1          00370           POP     DE
5039    D5          00380   P1B     PUSH    DE
503A    CDAF51      00390           CALL    SCAN
503D    D1          00400           POP     DE
503E    12          00410           LD      (DE),A
503F    13          00420           INC     DE
5040    78          00430           LD      A,B
5041    FE00        00440           CP      0
5043    2813        00450           JR      Z,PC1
5045    D5          00460           PUSH    DE
5046    ED5B8B52    00470           LD      DE,(TABLE2)
504A    2AB552      00480           LD      HL,(STABUF)
504D    19          00490   P1L2    ADD     HL,DE
504E    05          00500           DEC     B
```

Fig. 18-9. *Rotina de Cópia da Memória.*

```
504F    78          00510           LD      A,B
5050    FE00        00520           CP      0
5052    20F9        00530           JR      NZ,P1L2
5054    225B52      00540           LD      (STABUF),HL
5057    D1          00550           POP     DE
5058    D5          00560   P1C     PUSH    DE
5059    CDAF51      00570           CALL    SCAN
505C    D1          00580           POP     DE
505D    12          00590           LD      (DE),A
505E    13          00600           INC     DE
505F    78          00610           LD      A,B
5060    FE00        00620           CP      0
5062    2813        00630           JR      Z,P1D
5064    D5          00640           PUSH    DE
5065    ED5B8D52    00650           LD      DE,(TABLE3)
5069    2A5B52      00660           LD      HL,(STABUF)
506C    19          00670   P1L3    ADD     HL,DE
506D    05          00680           DEC     B
506E    78          00690           LD      A,B
506F    FE00        00700           CP      0
5071    20F9        00710           JR      Z,P1L3
5073    D1          00720           POP     DE
5074    225B52      00730           LD      (STABUF),HL
5077    D5          00740   P1D     PUSH    DE
5078    CDAF51      00750           CALL    SCAN
507B    D1          00760           POP     DE
507C    12          00770           LD      (DE),A
507D    78          00780           LD      A,B
507E    FE00        00790           CP      0
5080    2811        00800           JR      Z,PRINT2
5082    ED5B8F52    00810           LD      DE,(TABLE4)
5086    2A5B52      00820           LD      HL,(STABUF)
5089    19          00830   P1L4    ADD     HL,DE
508A    05          00840           DEC     B
508B    78          00850           LD      A,B
508C    FE00        00860           CP      0
508E    20F9        00870           JR      NZ,P1L4
5090    225B52      00880           LD      (STABUF),HL
5093    217552      00890   PRINT2  LD      HL,MESS2
5096    11803C      00900           LD      DE,3C00H+128
5099    011400      00910           LD      BC,20
509C    EDB0        00920           LDIR
509E    D5          00930           PUSH    DE
509F    CDAF51      00940           CALL    SCAN
50A2    D1          00950           POP     DE
50A3    12          00960           LD      (DE),A
50A4    13          00970           INC     DE
50A5    78          00980   P2A     LD      A,B
50A6    FE00        00990           CP      0
50A8    2813        01000           JR      Z,P2B
50AA    2A5D52      01010           LD      HL,(ENDBUF)
50AD    D5          01020           PUSH    DE
50AE    ED5B8952    01030           LD      DE,(TABLE1)
50B2    19          01040   P2L1    ADD     HL,DE
50B3    05          01050           DEC     B
```

Fig. 18-9. *Continuação da pág. 133.*

```
50B4    78          01060          LD      A,B
50B5    FE00        01070          CP      0
50B7    20F9        01080          JR      NZ,P2L1
50B9    225D52      01090          LD      (ENDBUF),HL
50BC    D1          01100          POP     DE
50BD    D5          01110  P2B     PUSH    DE
50BE    CDAF51      01120          CALL    SCAN
50C1    D1          01130          POP     DE
50C2    12          01140          LD      (DE),A
50C3    13          01150          INC     DE
50C4    78          01160          LD      A,B
50C5    FE00        01070          CP      0
50C7    2813        01180          JR      Z,P2C
50C9    D5          01190          PUSH    DE
50CA    ED5B8B52    01200          LD      DE,(TABLE2)
50CE    2A5D52      01210          LD      HL,(ENDBUF)
50D1    19          01220  P2L2    ADD     HL,DE
50D2    05          01230          DEC     B
50D3    78          01240          LD      A,B
50D4    FE00        01250          CP      0
50D6    20F9        01260          JR      NZ,P2L2
50D8    225D52      01270          LD      (ENDBUF),HL
50DB    D1          01280          POP     DE
50DC    D5          01290  P2C     PUSH    DE
50DD    CDAF51      01300          CALL    SCAN
50E0    D1          01310          POP     DE
50E1    12          01320          LD      (DE),A
50E2    13          01330          INC     DE
50E3    78          01340          LD      A,B
50E4    FE00        01350          CP      0
50E6    2813        01360          JR      Z,P2D
50E8    D5          01370          PUSH    DE
50E9    ED5B8D52    01380          LD      DE,(TABLE3)
50ED    2A5D52      01390          LD      HL,(ENDBUF)
50F0    19          01400  P2L3    ADD     HL,DE
50F1    05          01410          DEC     B
50F2    78          01420          LD      A,B
50F3    FE00        01430          CP      0
50F5    20F9        01440          JR      NZ,P2L3
50F7    225D52      01450          LD      (ENDBUF),HL
50FA    D1          01460          POP     DE
50FB    D5          01470  P2D     PUSH    DE
50FC    CDAF51      01480          CALL    SCAN
50FF    D1          01490          POP     DE
5100    12          01500          LD      (DE),A
5101    78          01510          LD      A,B
5102    FE00        01520          CP      0
5104    2811        01530          JR      Z,PRINT3
5106    ED5B8F52    01540          LD      DE,(TABLE4)
510A    2A5D52      01550          LD      HL,(ENDBUF)
510D    19          01560  P2L4    ADD     HL,DE
510E    05          01570          DEC     B
510F    78          01580          LD      A,B
5110    FE00        01590          CP      0
5112    20F9        01600          JR      NZ,P2L4
5114    225D52      01610          LD      (ENDBUF),HL
```

```
5117    11003C  01620  PRINT3  LD      DE,3C00H
511A    0618    01630          LD      B,24
511C    C5      01640  P3L1    PUSH    BC
511D    1A      01650          LD      A,(DE)
511E    4F      01660          LD      C,A
511F    CD4752  01670          CALL    POUT
5122    13      01680          INC     DE
5123    C1      01690          POP     BC
5124    05      01700          DEC     B
5125    78      01710          LD      A,B
5126    FE00    01720          CP      0
5128    20F2    01730          JR      NZ,P3L1
512A    0E0D    01740          LD      C,0DH
512C    CD4752  01750          CALL    POUT
512F    CD4752  01760          CALL    POUT
5132    11803C  01770          LD      DE,3C00H+128
5135    0618    01780          LD      B,24
5137    C5      01790  P3L2    PUSH    BC
5138    1A      01800          LD      A,(DE)
5139    4F      01810          LD      C,A
513A    CD4752  01820          CALL    POUT
513D    13      01830          INC     DE
513E    C1      01840          POP     BC
513F    05      01850          DEC     B
5140    78      01860          LD      A,B
5141    FE00    01870          CP      0
5143    20F2    01880          JR      NZ,P3L2
5145    0E0D    01890          LD      C,0DH
5147    CD4752  01900          CALL    POUT
514A    0620    01910          LD      B,32
514C    C5      01920  P3L3    PUSH    BC
514D    0E2D    01930          LD      C,2DH
514F    CD4752  01940          CALL    POUT
5152    C1      01950          POP     BC
5153    05      01960          DEC     B
5154    78      01970          LD      A,B
5155    FE00    01980          CP      0
5157    20F3    01990          JR      NZ,P3L3
5159    0E0D    02000          LD      C,0DH
515B    CD4752  02010          CALL    POUT
515E    2A5D52  02020  PREP3   LD      HL,(ENDBUF)
5161    ED5B5B52 02030         LD      DE,(STABUF)
5165    ED52    02040          SBC     HL,DE
5167    225F52  02050          LD      (TOTBUF),HL
516A    ED4B5F52 02060         LD      BC,(TOTBUF)
516E    ED5B5B52 02070         LD      DE,(STABUF)
5172    C5      02080  START   PUSH    BC
5173    7A      02090          LD      A,D
5174    CD2852  02100          CALL    HEXCV
5177    4C      02110          LD      C,H
5178    CD4752  02120          CALL    POUT
517B    4D      02130          LD      C,L
517C    CD4752  02140          CALL    POUT
517F    7B      02150          LD      A,E
5180    CD2852  02160          CALL    HEXCV
```

Fig. 18-9. *Continuação da pág. 135.*

```
5183    4C        02170          LD      C,H
5184    CD4752    02180          CALL    POUT
5187    4D        02190          LD      C,L
5188    CD4752    02200          CALL    POUT
518B    0E20      02210          LD      C,20H
518D    CD4752    02220          CALL    POUT
5190    CD4752    02230          CALL    POUT
5193    1A        02240          LD      A,(DE)
5194    CD2852    02250          CALL    HEXCV
5197    4C        02260          LD      C,H
5198    CD4752    02270          CALL    POUT
519B    4D        02280          LD      C,L
519C    CD4752    02290          CALL    POUT
519F    0E0D      02300          LD      C,0DH
51A1    CD4752    02310          CALL    POUT
51A4    13        02320          INC     DE
51A5    C1        02330          POP     BC
51A6    0B        02340          DEC     BC
51A7    78        02350          LD      A,B
51A8    B1        02360          OR      C
51A9    C27251    02370          JP      NZ,START
51AC    C30050    02380          JP      5000H
51AF    3E23      02390  SCAN    LD      A,23H
51B1    12        02400          LD      (DE),A
51B2    D5        02410          PUSH    DE
51B3    CD4900    02420          CALL    049H
51B6    D1        02430          POP     DE
51B7    FE30      02440          CP      30H
51B9    2003      02450          JR      NZ,S1
51BB    0600      02460          LD      B,0
51BD    C9        02470          RET
51BE    FE31      02480  S1      CP      31H
51C0    2003      02490          JR      NZ,S2
51C2    0601      02500          LD      B,1
51C4    C9        02510          RET
51C5    FE32      02520  S2      CP      32H
51C7    2003      02530          JR      NZ,S3
51C9    0602      02540          LD      B,2
51CB    C9        02550          RET
51CC    FE33      02560  S3      CP      33H
51CE    2003      02570          JR      NZ,S4
51D0    0603      02580          LD      B,3
51D2    C9        02590          RET
51D3    FE34      02600  S4      CP      34H
51D5    2003      02610          JR      NZ,S5
51D7    0604      02620          LD      B,4
51D9    C9        02630          RET
51DA    FE35      02640  S5      CP      35H
51DC    2003      02650          JR      NZ,S6
51DE    0605      02660          LD      B,5
51E0    C9        02670          RET
51E1    FE36      02680  S6      CP      36H
51E3    2003      02690          JR      NZ,S7
51E5    0606      02700          LD      B,6
51E7    C9        02710          RET
51E8    FE37      02720  S7      CP      37H
```

```
51EA    2003    02730           JR      NZ,S8
51EC    0607    02740           LD      B,7
51EE    C9      02750           RET
51EF    FE38    02760   S8      CP      38H
51F1    2003    02770           JR      NZ,S9
51F3    0608    02780           LD      B,8
51F5    C9      02790           RET
51F6    FE39    02800   S9      CP      39H
51F8    2003    02810           JR      NZ,SA
51FA    0609    02820           LD      B,9
51FC    C9      02830           RET
51FD    FE41    02840   SA      CP      41H
51FF    2003    02850           JR      NZ,SN
5201    060A    02860           LD      B,10
5203    C9      02870           RET
5204    FE42    02880   SB      CP      42H
5206    2003    02890           JR      NZ,SC
5208    030B    02900           LD      B,11
520A    C9      02910           RET
520B    FE43    02920   SC      CP      43H
520D    2003    02930           JR      NZ,SD
520F    060C    02940           LD      B,12
5211    C9      02950           RET
5212    FE44    02960   SD      CP      44H
5214    2003    02970           JR      NZ,SE
5216    606D    02980           LD      B,13
5218    C9      02990           RET
5219    FE45    03000   SE      CP      45H
521B    2003    03010           JR      NZ,SF
521D    600E    03020           LD      B,14
521F    C9      03030           RET
5220    FE46    03040   SF      CP      46H
5222    C2AF51  03050           JP      NZ,SCAN
5225    060F    03060           LD      B,15
5227    C9      03070           RET
5228    4F      03080   HEXCV   LD      C,A
5229    CB3F    03090           SRL     A
522B    CB3F    03100           SRL     A
522D    CB3F    03110           SRL     A
522F    CB3F    03120           SRL     A
5231    CD3D52  03130           CALL    TEST
5234    67      03140           LD      H,A
5235    79      03150           LD      A,C
5236    E60F    03160           AND     0FH
5238    CD3D52  03170           CALL    TEST
523B    6F      03180           LD      L,A
523C    C9      03190           RET
523D    C630    03200   TEST    ADD     A,30H
523F    FE3A    03210           CP      3AH
5241    FA4652  03220           JP      M,TEST1
5244    C607    03230           ADD     A,7
5246    C9      03240   TEST1   RET
5247    3AE837  03250   POUT    LD      A,(37E8H)
524A    E6F0    03260           AND     0F0H
524C    FE30    03270           CP      30H
```

Fig. 18-9. *Continuação da pág. 137.*

```
524E  20F7    03280          JR      NZ,POUT
5250  79      03290          LD      A,C
5251  32E837  03300          LD      (37E8H),A
5254  0605    03310          LD      B,5
5256  05      03320  POUT1   DEC     B
5257  78      03330          LD      A,B
5258  20FC    03340          JR      NZ,POUT1
525A  C9      03350          RET
0002          03360  STABUF  DEFS    2
0002          03370  ENDBUF  DEFS    2
0002          03380  TOTBUF  DEFS    2
5261  53      03390  MESS1   DEFM    'STARTING
                                     ADDRESS
5262  54
5263  41
5264  52
5265  54
5266  49
5267  4E
5268  47
5269  20
526A  41
526B  44
526C  44
526D  52
526E  45
526F  53
5270  53
5271  20
5272  20
5273  20
5274  20
5275  45      03400  MESS2   DEFM    ENDING
                                     ADDRESS
5276  4E
5277  44
5278  49
5279  4E
527A  47
527B  20
527C  41
527D  44
527E  44
527F  52
5280  45
5281  53
5282  53
5283  20
5284  20
5285  20
5286  20
5287  20
5288  20
5289  0010    03410  TABLE1  DEFW    1000H
528B  0001    03420  TABLE2  DEFW    0100H
528D  1000    03430  TABLE3  DEFW    0010H
528F  0100    03440  TABLE4  DEFW    0001H
```

```
0000                    03450                   END
00000   TOTAL ERRORS
POUT1   5256
TEST1   5246
TEST    523D
SF      5220
SE      5219
SD      5212
SC      520B
SB      5204
SA      51FD
S9      51F6
S8      51EF
S7      51E8
S6      51E1
S5      51DA
S4      51D3
S3      51CC
S2      51C5
S1      51BE
HEXCV   5228
START   5172
PREP3   515E
P3L3    514C
P3L2    5137
POUT    5247
P3L1    511C
P2L4    510D
PRINT3  5111
P2L3    50F0
P2D     50FB
P2L2    50D1
P2C     50DC
P2L1    50B2
P2B     50BD
P2A     50A5
MESS2   5275
P1L4    5089
TABLE4  528F
PRINT2  5093
P1L3    506C
TABLE3  528D
P1D     5077
P1L2    504D
TABLE2  528B
P1C     5058
P1L1    502E
TABLE1  5289
TOTBUF  525F
ENDBUF  525D
STABUF  525B
P1B     5039
P1A     5015
SCAN    51AF
PRINT1  500E
MESS1   5261
```

Fig. 18-9. *Continuação da pág. 139.*

As linhas de 120 a 150 fazem os preparativos para a exibição da mensagem para que o operador entre com o endereço inicial.

A linha 160 é o início da rotina PRINT1. Aqui, o par DE sofre um PUSH, para que seja usado pela rotina ROM que executará em seguida.

A linha 170 chama a sub-rotina SCAN.

A linha 180 efetua um POP, buscando de volta o conteúdo de DE.

A linha 190 carrega o valor do registrador A (ele foi ali colocado pela rotina SCAN) na localização especificada pelo par DE.

A linha 200 incrementa o par DE para apontar para a próxima posição no vídeo.

A linha 210 é o início do "loop" (volta fechada) P1A. Neste trecho, o registrador A é carregado com o conteúdo do registrador B. Este terá sido o valor do endereço inicial, que ali foi colocado após a computação realizada pela rotina SCAN. O registrador B conterá o multiplicador.

A linha 220 compara o valor do registrador A com zero.

A linha 230 provoca um salto para P1B se o valor em A for zero, isto é, se a comparação anterior resultar verdadeira.

A linha 240 carrega zero no par de registradores HL. Isto será usado nas linhas 250 e 270, para inicializar os valores do buffer de início (STABUF), do buffer de fim (ENDBUF) e do buffer de total (TOTBUF) com zero. TOTBUF irá conter o número de bytes a serem enviados para a impressora.

A linha 280 carrega o par HL com o conteúdo do STABUF.

A linha 290 salva o valor atual do par DE.

A linha 300 carrega o par DE com o valor armazenado em TABLE1.

A linha 310 é o início do loop de adição. O conteúdo do par DE, o qual armazena a tabela, é repetidamente somado ao par HL, que contém o total atual. Após a execução de cada adição, o registrador B é decrementado e verificado se se tornou zerado. Se ele não for zero, a rotina é repetida. Caso seja, a próxima rotina é executada. Esta apanha o próximo dígito hexadecimal e soma seu valor ao buffer. O loop termina na linha 360, onde STABUF é carregado com o conteúdo do par HL. A carga de STABUF é completada na linha 880.

As linhas de 890 a 1610 são uma repetição das linhas 120 a 880. A única diferença reside em que estas linhas lidam com o ENDBUF, em lugar do STABUF.

As linhas de 1620 a 1760 enviam o conteúdo da primeira linha do vídeo para a impressora. Elas terminam com o envio de retornos do carro para obter espacejamento duplo na listagem.

As linhas 1770 a 1900 enviam a segunda linha do vídeo para a impressora e enviam um único retorno do carro, no final.

N.T. – Tenho, continuamente, utilizado os termos loop e buffer sem aspas, devido ao fato de serem termos já incorporados ao linguajar do pessoal de processamento de dados e computação em geral, no Brasil.

As linhas 1910 a 2010 imprimem uma seqüência de –'s ao longo da página e terminam com um retorno do carro.

As linhas 2020 a 2070 estabelecem os endereços de início e de fim, usando os conteúdos de STABUF e de ENDBUF. Elas também determinam o número de posições a serem impressas, subtraindo o endereço de início do endereço de fim e armazenando o resultado em TOTBUF. O par DE é, então, carregado com o endereço de início e o par BC com o número total de bytes a serem transmitidos.

As linhas de 2080 até 2380 compõem a rotina START. Os dados são convertidos pela rotina HEXCV e enviados para a impressora, fazendo-se inicialmente a carga do registrador A com o valor do registrador D e convertendo este valor antes da impressão. A seguir, o registrador A é carregado com o registrador E e o processo é repetido. Finalmente, o registrador A é carregado com o conteúdo da posição apontada pelo par DE e o processo continua pela última vez. Ao fim de toda a passagem, o par BC é recuperado por um POP e decrementado. Se isto fizer com que seu valor seja zero, causará o retorno do programa para 5000H (o início), para lhe proporcionar a chance de entrar com as próximas posições que você deseje que sejam impressas. Caso BC não se torne zerado, a rotina START é repetida.

A linha 2390 inicia a rotina SCAN. O registrador A é carregado com o caracter #, o qual é impresso, ou melhor, exibido na tela. O par DE é salvo, através de um PUSH e a rotina em ROM de varredura da tela (scan) é chamada. A seguir o par DE é recuperado pelo POP. Após tal procedimento, o conteúdo do registrador A, que possui o valor da tecla pressionada detectada pela rotina de varredura (scan), é comparado com cada um dos possíveis dígitos de 0 a F (hexadecimais). Se se encontra uma comparação verdadeira, o registrador B é carregado com o valor decimal representativo do caracter e o programa retorna à rotina principal. Se não se encontra uma comparação positiva, a rotina continua no teste seguinte, até chegar a testar o caracter F hexadecimal. Se não se deteta uma condição verdadeira com o caracter F, a rotina SCAN é repetida.

As linhas 3080 a 3240 convertem o valor armazenado no registrador A para a sua correspondente representação em código ASCII, para a impressão. Nesta rotina, o registrador C é usado para armazenar o valor original de A. A seguir, o registrador A é deslocado para a esquerda quatro vezes e faz-se um teste para se determinar se o valor do registrador A é um número hexadecimal válido. Na parte denominada TEST, o caracter sofre uma adição do valor 30H, para transformá-lo no correspondente ASCII. A seguir, ele é testado para ver se é menor ou igual a 9. Se for, o registrador H é carregado com este valor e o processo é repetido, com a carga do registrador A com o conteúdo de C e os novos testes. Após a conclusão do TEST, o registrador L estará carregado com o valor de A, e o par HL conterá os dois valores ASCII. No caso da verificação resultar na deteção de um valor superior a 9, soma-se 7 ao registrador A para que se reproduza o valor da letra em questão.

As linhas 3250 até 3350 constituem a rotina POUT, ou seja, de impres-

são. Aqui, o caracter é enviado à impressora da mesma forma como mencionado em exemplos anteriores. O loop de atraso nas linhas de 3320 a 3340 permite-nos enviar mais de um elemento de dados de cada vez.

As linhas 3360 a 3380 criam as áreas de buffer, através da definição de espaço para as mesmas (DEFS) de 2 bytes cada. A razão pela qual reservamos duas posições para os valores armazenados nestes buffers é que, raramente, eles serão inferiores a 255, ou seja, FFH.

As linhas 3390 a 3400 constituem as mensagens, cada qual com um comprimento de 20 caracteres (adicionamos espaço para tornar o tamanho par, a fim de facilitar a tabulação).

As linhas 3410 até 3440 estabelecem os valores para as tabelas, através do comando DEFW (Define Word). Estes valores são os multiplicadores utilizados nas rotinas de adição anteriores.

A linha 3450 é o comando END obrigatório.

PROGRAMA TPXFER (Tape Transfer — Cópia de Fita)

Este programa (Fig. 18-10) lhe possibilitará fazer a cópia de uma fita de sistema, desde que a mesma não tenha, em seu início, um programa especial de autocarga. Ele funciona nos sistemas TRS-80 Modelos I e III. A origem (ORG) pode ser alterada, se desejado, porém isto acarretará diminuição no buffer.

Aconselho a você entrar este programa em seu micro e tê-lo sempre à mão. Eu possuía diversas fitas importantes e me preocupava muito com a possibilidade de perdê-las e, assim sendo, escrevi este programa e usei-o para fazer diversas cópias daquelas fitas. Passei então a utilizar as cópias e arquivei as originais em local bastante seguro e apropriado.

Tenho, também, me deparado com casos em que uma certa fita de sistema não consegue ser carregada no computador. Fazendo uma cópia da fita, com o TPXFER, consegui produzir uma versão que funciona, apesar dos problemas com a fita original. Uma fita de sistema tem o formato mostrado na Fig. 18-11.

O processo de carga da fita no sistema é simples. A máquina lê a fita até reconhecer um "header" (cabeçalho – contudo, o termo header é familiar no meio da programação de computadores) de arquivo 55H. A seguir busca os próximos 6 bytes, que constituem o nome do arquivo. Esta é a maneira pela qual o sistema reconhece se está sendo carregado o programa que foi especificado. O próximo elemento que o sistema busca é um header de dados 3CH. Tal caracter informa ao sistema que os dados vêm a seguir. Em seguida ao header de dados, vem o byte de contagem, ou seja um byte que indica o número de bytes contidos no block que será transferido em seguida. Tal número pode variar de 6 a 255. Nem todos os blocos possuem 255 bytes.

Após tomar conhecimento do número de bytes que serão lidos, o computador lê o endereço inicial de carga. Este não é, necessariamente, o endere-

```
5000            00100           ORG     5000H
5000 CDC901     00110 START     CALL    01C9H   ;CLEAR THE SCREEN
5003 11193C     00120           LD      DE,3C00H+25
5006 213553     00130           LD      HL,MESS1
5009 010B00     00140           LD      BC,MESS1L
500C EDB0       00150           LDIR
500E 11B03D     00160 ML1A      LD      DE,3C00H+384
5011 214053     00170           LD      HL,MESS2
5014 011100     00180           LD      BC,MESS2L
5017 EDB0       00190           LDIR
5019 11003E     00200           LD      DE,3C00H+512
501C 215153     00210           LD      HL,MESS3
501F 012400     00220           LD      BC,MESS3L
5022 EDB0       00230           LDIR
5024 11403E     00240           LD      DE,3C00H+576
5027 217553     00250           LD      HL,MESS4
502A 012300     00260           LD      BC,MESS4L
502D EDB0       00270           LDIR
502F 11803E     00280           LD      DE,3C00H+640
5032 219853     00290           LD      HL,MESS5
5035 011A00     00300           LD      BC,MESS5L
5038 EDB0       00310           LDIR
503A 11C03E     00320           LD      DE,3C00H+704
503D 21B253     00330           LD      HL,MESS6
5040 012A00     00340           LD      BC,MESS6L
5043 EDB0       00350           LDIR
                00360 ;
                00370 ;         TURN CURSOR OFF
                00380 ;
5045 3E0F       00390           LD      A,0FH
5047 D5         00400           PUSH    DE
5048 FDE5       00410           PUSH    IY
504A CD3300     00420           CALL    0033H
504D FDE1       00430           POP     IY
                00440 ;
                00450 ;         SCAN THE KEYBOARD FOR THE FUNCTION REQUEST
                00460 ;
504F CD4900     00470 SCAN      CALL    049H
5052 FE4C       00480           CP      'L'
5054 CA6850     00490           JP      Z,LOAD
5057 FE53       00500           CP      'S'
5059 CA3251     00510           JP      Z,SAVE
505C FE56       00520           CP      'V'
505E CA8951     00530           JP      Z,VERR
5061 FE45       00540           CP      'E'
5063 CAE651     00550           JP      Z,QUIT
5066 18E7       00560           JR      SCAN
                00570 ;
                00580 ;         LOAD IN A PROGRAM HERE
                00590 ;
5068 CDC901     00600 LOAD      CALL    01C9H   ;CLEAR THE SCREEN
506B 210000     00610           LD      HL,0000
506E 220D55     00620           LD      (T1BUF),HL      ;ZERO THE BUFFER
5071 110A3C     00630           LD      DE,3C00H+10
5074 21EF52     00640           LD      HL,LMES1
5077 012F00     00650           LD      BC,LMES1L
507A EDB0       00660           LDIR
507C CDFA51     00670           CALL    BREAK   ;BE SURE THAT'S WHAT YOU W
    ANT.
507F CDC901     00680           CALL    01C9H   ;CLEAR THE SCREEN
5082 11003C     00690           LD      DE,3C00H
5085 211E53     00700           LD      HL,LDMS2
5088 010C00     00710           LD      BC,LDMS2L
508B EDB0       00720           LDIR
508D 3E00       00730           LD      A,0000
508F 320755     00740           LD      (BITBUF),A      ;ZERO THE BUFFER
5092 320855     00750           LD      (BITBUF+1),A    ;ZERO THIS ONE TOO

5095 210000     00760           LD      HL,0000
5098 111155     00770           LD      DE,TXTBUF       ;SET THE POINTER
509B CD1202     00780           CALL    0212H   ;DEFINE TAPE DRIVE
509E CD9602     00790           CALL    0296H   ;FIND SYNC BYTE
```

Fig. 18-10. *Copie Fitas de Sistema com esta Rotina.*

```
50A1 CD3502    00800          CALL    0235H   ;READ ONE BYTE
               00810 ;
               00820 ;        GET & GHECK FILE HEADER
               00830 ;
50A4 12        00840          LD      (DE),A  ;SAVE THE BYTE
50A5 13        00850          INC     DE      ;MOVE THE POINTER UP ONE
50A6 23        00860          INC     HL      ;HL IS THE COUNTER USED LA
     TER
50A7 FE55      00870          CP      55H     ;IS IT A FILE HEADER ??
50A9 CA1E51    00880          JP      Z,BADLD ; SHOULDN'T HAVE BEEN !!!
               00890 ;
               00900 ;        GET FILE NAME HERE
               00910 ;
50AC 010600    00920          LD      BC,6    ;6 CHARACTERS IN THE FILE
     NAME
50AF CD3502    00930 LL2      CALL    0235H   ;READ TAPE
50B2 12        00940          LD      (DE),A  ;SAVE THE BYTE
50B3 13        00950          INC     DE      ;MOVE POINTER UP ONE MORE
50B4 23        00960          INC     HL      ;ADD OND MORE TO THE COUNT
     ER
50B5 10F8      00970          DJNZ    LL2     ;GET ANOTHER ONE IF THAT W
     ASN'T #6
               00980 ;
               00990 ;        CHECK DATA HEADER HERE
               01000 ;
50B7 CD3502    01010          CALL    0235H   ;READ NEXT BYTE
50BA FE3C      01020          CP      3CH     ;WAS IT THE DATA HEADER??
50BC C21E51    01030          JP      NZ,BADLD        ;GO IF NOT
50BF 12        01040          LD      (DE),A  ;STORE IT IN THE BUFFER
50C0 13        01050          INC     DE      ;MOVE BUFFER POINTER UP ON
     E
50C1 23        01060          INC     HL      ;ADD 1 TO THE COUNTER
               01070 ;
               01080 ;        GET BYTE COUNT
               01090 ;
50C2 CD3502    01100 LL3      CALL    0235H   ;READ NEXT BYTE
50C5 320755    01110          LD      (BITBUF),A      ;SAVE THE BYTE COU
     NT
50C8 12        01120          LD      (DE),A  ;SAVE THE COUNT IN THE BUF
     FER TOO
50C9 13        01130          INC     DE      ;POINT TO NEXT LOCATION
50CA 23        01140          INC     HL      ;ADD 1 TO THE COUNTER
               01150 ;
               01160 ;        GET THE STARTING ADDRESS FOR THE BLOCK
               01170 ;
50CB 0602      01180          LD      B,2     ;# OF BYTES IN THE ADDRESS

50CD CD3502    01190 LL4      CALL    0235H   ;READ THE NEXT BYTE
50D0 12        01200          LD      (DE),A  ;SAVE IT IN THE BUFFER
50D1 13        01210          INC     DE      ;POINT TO NEXT LOCATION
50D2 23        01220          INC     HL      ;ADD TO THE COUNTER
50D3 10F8      01230          DJNZ    LL4     ;GET NEXT BYTE OR GO IF TH
     AT WAS 2
               01240 ;
               01250 ;        READ DATA
               01260 ;
50D5 3A0755    01270          LD      A,(BITBUF)      ;GET THE TOTAL # O
     F BYTES TO READ
50D8 47        01280          LD      B,A     ;MOVE THEM TO B
50D9 CD3502    01290 LL5      CALL    0235H   ;READ TAPE
50DC 12        01300          LD      (DE),A  ;STORE THE BYTE READ
50DD 13        01310          INC     DE      ;MOVE THE POINTER UP ONE
50DE 23        01320          INC     HL      ;ADD ONE TO THE COUNTER
50DF CDD052    01330          CALL    TOTAL
50E2 10F5      01340          DJNZ    LL5     ;READ NEXT ONE IF NOT LAST

               01350 ;
               01360 ;        GET CHECKSUM
               01370 ;
50E4 CD3502    01380          CALL    0235H   ;READ NEXT BYTE
50E7 12        01390          LD      (DE),A  ;SAVE IT
50E8 13        01400          INC     DE
50E9 23        01410          INC     HL
               01420 ;
```

```
                01430 ;         READ NEXT HEADER
                01440 ;
50EA CDBC52     01450           CALL    BLINK   ;LET'S BLINK AN *
50ED CD3502     01460           CALL    0235H   ;READ NEXT BYTE
50F0 12         01470           LD      (DE),A  ;SAVE IT
50F1 13         01480           INC     DE
50F2 23         01490           INC     HL
50F3 FE3C       01500           CP      3CH     ;WAS IT A DATA HEADER??
50F5 CAC250     01510           JP      Z,LL3   ;OK DO IT AGAIN
50F8 FE78       01520           CP      78H     ;WAS IT AN ENTRY HEADER??
50FA C21E51     01530           JP      NZ,BADLD        ;IT SHOULD HAVE BE
     EN
                01540 ;
                01550 ;         GET ENTRY ADDRESS
                01560 ;
50FD 0602       01570           LD      B,2     ;IT'LL HAVE 2 BYTES
50FF CD3502     01580 LL6       CALL    0235H   ;READ A BYTE
5102 12         01590           LD      (DE),A
5103 13         01600           INC     DE
5104 23         01610           INC     HL
5105 10F8       01620           DJNZ    LL6
5107 220755     01630           LD      (BITBUF),HL     ;STORE TOTAL # OF
     BYTES
510A CDF801     01640           CALL    01F8H   ;SHUT OFF THE MOTOR
510D CDC901     01650           CALL    01C9H   ;CLEAR THE SCREEN
5110 11143C     01660           LD      DE,3C00H+20
5113 212A53     01670           LD      HL,LDMS3
5116 010B00     01680           LD      BC,LDMS3L
5119 EDB0       01690           LDIR
511B C31652     01700           JP      BL2
                01710 ;
                01720 ;         BAD LOAD
                01730 ;
511E CDF801     01740 BADLD     CALL    01F8H   ;SHUT THE MOTOR OFF
5121 CDC901     01750           CALL    01C9H   ;CLEAR THE SCREEN
5124 11193C     01760           LD      DE,3C00H+25
5127 21DE52     01770           LD      HL,BADM1
512A 011100     01780           LD      BC,BADM1L
512D EDB0       01790           LDIR
512F C31652     01800           JP      BL2
                01810 ;
                01820 ;         RE WRITE THE PROGRAM TO NEW TAPE HERE
                01830 ;
5132 CDC901     01840 SAVE      CALL    01C9H
5135 11003C     01850           LD      DE,3C00H
5138 216D54     01860           LD      HL,SAVMS3
513B 011D00     01870           LD      BC,SAVM3L
513E EDB0       01880           LDIR
5140 118A3C     01890           LD      DE,3C00H+138
5143 212954     01900           LD      HL,VERMS4
5146 011C00     01910           LD      BC,VERM4L
5149 EDB0       01920           LDIR
514B CDFA51     01930           CALL    BREAK
514E CDC901     01940           CALL    01C9H
5151 11143C     01950           LD      DE,3C00H+20
5154 214554     01960           LD      HL,SAVMS1
5157 010F00     01970           LD      BC,SAVM1L
515A EDB0       01980           LDIR
515C 3E00       01990           LD      A,0
515E CD1202     02000           CALL    0212H
5161 CD8702     02010           CALL    0287H   ;WRITE LEADER
5164 ED4B0755   02020           LD      BC,(BITBUF)     ;GET TOTAL TO WRIT
     E
5168 111155     02030           LD      DE,TXTBUF       ;POINT TO FIRST BY
     TE
516B 1A         02040 SL2       LD      A,(DE)
516C CD6402     02050           CALL    0264H   ;WRITE IT TO TAPE
516F 13         02060           INC     DE
5170 0B         02070           DEC     BC
5171 78         02080           LD      A,B
5172 B1         02090           OR      C
5173 20F6       02100           JR      NZ,SL2
5175 CDF801     02110           CALL    01F8H
```

Fig. 18-10. *Continuação da pág. 145.*

```
5178 CDC901    02120          CALL     01C9H
517B 11003C    02130          LD       DE,3C00H
517E 215454    02140          LD       HL,SAVMS2
5181 011900    02150          LD       BC,SAVM2L
5184 EDB0      02160          LDIR
5186 C31652    02170          JP       BL2
               02180 ;
               02190 ;        VERIFY LOAD HERE
               02200 ;
5189 CDC901    02210 VERR     CALL     01C9H
518C 11143C    02220          LD       DE,3C00H+20
518F 21DC53    02230          LD       HL,VERMS1
5192 012600    02240          LD       BC,VERM1L
5195 EDB0      02250          LDIR
5197 118A3C    02260          LD       DE,3C00H+138
519A 212954    02270          LD       HL,VERMS4
519D 011C00    02280          LD       BC,VERM4L
51A0 EDB0      02290          LDIR
51A2 CDFA51    02300          CALL     BREAK
51A5 CDC901    02310          CALL     01C9H
51A8 11143C    02320          LD       DE,3C00H+20
51AB 210254    02330          LD       HL,VERMS2
51AE 010E00    02340          LD       BC,VERM2L
51B1 EDB0      02350          LDIR
51B3 ED4B0755  02360          LD       BC,(BITBUF)
51B7 211155    02370          LD       HL,TXTBUF
51BA CD1202    02380          CALL     0212H
51BD CD9602    02390          CALL     0296H
51C0 CD3502    02400 VL2      CALL     0235H
51C3 BE        02410          CP       (HL)
51C4 C21E51    02420          JP       NZ,BADLD
51C7 FE3C      02430          CP       3CH
51C9 CCBC52    02440          CALL     Z,BLINK
51CC 23        02450          INC      HL
51CD 0B        02460          DEC      BC
51CE 78        02470          LD       A,B
51CF B1        02480          OR       C
51D0 20EE      02490          JR       NZ,VL2
51D2 CDF801    02500          CALL     01F8H
51D5 CDC901    02510          CALL     01C9H
51D8 11193C    02520          LD       DE,3C00H+25
51DB 211054    02530          LD       HL,VERMS3
51DE 012854    02540          LD       BC,VERM3L
51E1 EDB0      02550          LDIR
51E3 C31652    02560          JP       BL2
               02570 ;
               02580 ;        RETURN TO READY STATE
               02590 ;
51E6 CDC901    02600 QUIT     CALL     01C9H
51E9 110A3C    02610          LD       DE,3C00H+10
51EC 21E354    02620          LD       HL,QUIT1
51EF 012400    02630          LD       BC,QUIT1L
51F2 EDB0      02640          LDIR
51F4 CDFA51    02650          CALL     BREAK
51F7 C3191A    02660 LOOP     JP       1A19H    ;RETURN TO BASIC READY
               02670 ;
               02680 ;        BREAK FOR SAFETY MEASURES HERE
               02690 ;
51FA 11803D    02700 BREAK    LD       DE,3C00H+384
51FD 218A54    02710          LD       HL,BRKMS
5200 012200    02720          LD       BC,BRKMSL
5203 EDB0      02730          LDIR
5205 CD4900    02740 BL1      CALL     049H
5208 FE0D      02750          CP       0DH
520A C8        02760          RET      Z
520B FE4D      02770          CP       'M'
520D C20552    02780          JP       NZ,BL1
5210 CDC901    02790          CALL     01C9H
5213 C30E50    02800          JP       ML1A
5216 ED5B1A55  02810 BL2      LD       DE,(HLDBUF)
521A ED530955  02820          LD       (STABUF),DE
521E 2A0955    02830          LD       HL,(STABUF)
5221 ED5B0D55  02840          LD       DE,(T1BUF)
5225 19        02850          ADD      HL,DE
```

```
5226 220B55    02860          LD      (ENDBUF),HL
5229 ED4B0755  02870          LD      BC,(BITBUF)
522D 211155    02880          LD      HL,TXTBUF
5230 23        02890 EL1      INC     HL
5231 0B        02900          DEC     BC
5232 78        02910          LD      A,B
5233 B1        02920          OR      C
5234 20FA      02930          JR      NZ,EL1
5236 2B        02940          DEC     HL
5237 2B        02950          DEC     HL
5238 7E        02960          LD      A,(HL)
5239 320B55    02970          LD      (ENDBUF),A
523C 23        02980          INC     HL
523D 7E        02990          LD      A,(HL)
523E 320C55    03000          LD      (ENDBUF+1),A
5241 118F3C    03010          LD      DE,3C00H+143
5244 21AC54    03020          LD      HL,SCMS1
5247 011000    03030          LD      BC,SCMS1L
524A EDB0      03040          LDIR
524C 211255    03050          LD      HL,TXTBUF+1
524F 010600    03060          LD      BC,6
5252 11003D    03070          LD      DE,3C00H+256
5255 21BC54    03080          LD      HL,SCMS2
5258 010900    03090          LD      BC,SCMS2L
525B EDB0      03100          LDIR
525D 2A0955    03110          LD      HL,(STABUF)
5260 CD8252    03120          CALL    LP2
5263 21C554    03130          LD      HL,SCMS3
5266 010E00    03140          LD      BC,SCMS3L
5269 EDB0      03150          LDIR
526B 2A0B55    03160          LD      HL,(ENDBUF)
526E CD8252    03170          CALL    LP2
5271 21D354    03180          LD      HL,SCMS4
5274 011000    03190          LD      BC,SCMS4L
5277 EDB0      03200          LDIR
5279 2A0F55    03210          LD      HL,(ENTBUF)
527C CD8252    03220          CALL    LP2
527F C30E50    03230          JP      ML1A
5282 7C        03240 LP2      LD      A,H
5283 E5        03250          PUSH    HL
5284 CD9652    03260          CALL    CONV
5287 CDB552    03270          CALL    PRNT
528A E1        03280          POP     HL
528B 7D        03290          LD      A,L
528C CD9652    03300          CALL    CONV
528F CDB552    03310          CALL    PRNT
5292 13        03320          INC     DE
5293 13        03330          INC     DE
5294 13        03340          INC     DE
5295 C9        03350          RET
               03360 ;
               03370 ;        CONVERT TO ASCII HERE
               03380 ;
5296 4F        03390 CONV     LD      C,A
5297 CB3F      03400          SRL     A
5299 CB3F      03410          SRL     A
529B CB3F      03420          SRL     A
529D CB3F      03430          SRL     A
529F CDAB52    03440          CALL    CT1
52A2 67        03450          LD      H,A
52A3 79        03460          LD      A,C
52A4 E60F      03470          AND     0FH
52A6 CDAB52    03480          CALL    CT1
52A9 6F        03490          LD      L,A
52AA C9        03500          RET
52AB C630      03510 CT1      ADD     A,30H
52AD FE3A      03520          CP      3AH
52AF FAB452    03530          JP      M,LP9
52B2 C607      03540          ADD     A,7
52B4 C9        03550 LP9      RET
               03560 ;
               03570 ;        PRINT CHARACTER ON SCREEN HERE
               03580 ;
```

Fig. 18-10. *Continuação da pág. 147.*

```
52B5 7C        03590 PRNT    LD      A,H
52B6 12        03600         LD      (DE),A
52B7 13        03610         INC     DE
52B8 7D        03620         LD      A,L
52B9 12        03630         LD      (DE),A
52BA 13        03640         INC     DE
52BB C9        03650         RET
               03660 ;
               03670 ;       BLINK * HERE
               03680 ;
52BC 3A3F3C    03690 BLINK   LD      A,(3C00H+63)
52BF FE2A      03700         CP      2AH     ;IS IT AN * ??
52C1 CACA52    03710         JP      Z,AOFF
52C4 3E2A      03720         LD      A,2AH
52C6 323F3C    03730         LD      (3C00H+63),A
52C9 C9        03740         RET
               03750 ;
               03760 ;       TURN OFF THE *
               03770 ;
52CA 3E20      03780 AOFF    LD      A,20H
52CC 323F3C    03790         LD      (3C00H+63),A
52CF C9        03800         RET
               03810 ;
               03820 ;       CALCULATE TOTAL # OF BYTES HERE
               03830 ;
52D0 C5        03840 TOTAL   PUSH    BC
52D1 D5        03850         PUSH    DE
52D2 E5        03860         PUSH    HL
52D3 2A0D55    03870         LD      HL,(T1BUF)
52D6 23        03880         INC     HL
52D7 220D55    03890         LD      (T1BUF),HL
52DA E1        03900         POP     HL
52DB D1        03910         POP     DE
52DC C1        03920         POP     BC
52DD C9        03930         RET
               03940 ;
               03950 ;       MESSAGES & BUFFER ASIGNMENTS HERE
               03960 ;
52DE 42        03970 BADM1   DEFM    'BAD CASSETTE LOAD'
     41 44 20 43 41 53 53 45
     54 54 45 20 4C 4F 41 44
0011           03980 BADM1L  EQU     $-BADM1
52EF 49        03990 LMES1   DEFM    'INCERT TAPE TO BE COPIED, THEN PR
     ESS  * ENTER *'
     4E 43 45 52 54 20 54 41
     50 45 20 54 4F 20 42 45
     20 43 4F 50 49 45 44 2C
     20 54 48 45 4E 20 50 52
     45 53 53 20 20 2A 20 45
     4E 54 45 52 20 2A
002F           04000 LMES1L  EQU     $-LMES1
531E 4C        04010 LDMS2   DEFM    'LOADING TAPE'
     4F 41 44 49 4E 47 20 54
     41 50 45
000C           04020 LDMS2L  EQU     $-LDMS2
532A 54        04030 LDMS3   DEFM    'TAPE LOADED'
     41 50 45 20 4C 4F 41 44
     45 44
000B           04040 LDMS3L  EQU     $-LDMS3
5335 54        04050 MESS1   DEFM    'TAPE COPIER'
     41 50 45 20 43 4F 50 49
     45 52
000B           04060 MESS1L  EQU     $-MESS1
5340 45        04070 MESS2   DEFM    'ENTER           TO'
     4E 54 45 52 20 20 20 20
     20 20 20 20 20 20 54 4F
0011           04080 MESS2L  EQU     $-MESS2
5351 20        04090 MESS3   DEFM    '  L         LOAD PROGRAM TO BE COP
     IED'
     20 4C 20 20 20 20 20 20
     20 20 4C 4F 41 44 20 50
     52 4F 47 52 41 4D 20 54
     4F 20 42 45 20 43 4F 50
     49 45 44
0024           04100 MESS3L  EQU     $-MESS3
```

```
5375 20        04110 MESS4   DEFM    '  S         SAVE PROGRAM ON NEW TA
     PE'
     20 53 20 20 20 20 20 20
     20 20 53 41 56 45 20 50
     52 4F 47 52 41 4D 20 4F
     4E 20 4E 45 57 20 54 41
     50 45
0023           04120 MESS4L  EQU     $-MESS4
5398 20        04130 MESS5   DEFM    '  V         VERIFY NEW TAPE'
     20 56 20 20 20 20 20 20
     20 20 56 45 52 49 46 59
     20 4E 45 57 20 54 41 50
     45
001A           04140 MESS5L  EQU     $-MESS5
53B2 20        04150 MESS6   DEFM    '  E         END SESSION AND RETURN
      TO BASIC'
     20 45 20 20 20 20 20 20
     20 20 45 4E 44 20 53 45
     53 53 49 4F 4E 20 41 4E
     44 20 52 45 54 55 52 4E
     20 54 4F 20 42 41 53 49
     43
002A           04160 MESS6L  EQU     $-MESS6
53DC 50        04170 VERMS1  DEFM    'PREPARE RECORDER TO PLAY BACK NEW
      TAPE'
     52 45 50 41 52 45 20 52
     45 43 4F 52 44 45 52 20
     54 4F 20 50 4C 41 59 20
     42 41 43 4B 20 4E 45 57
     20 54 41 50 45
0026           04180 VERM1L  EQU     $-VERMS1
5402 56  '     04190 VERMS2  DEFM    'VERIFYING COPY'
     45 52 49 46 59 49 4E 47
     20 43 4F 50 59
000E           04200 VERM2L  EQU     $-VERMS2
5410 47        04210 VERMS3  DEFM    'GOOD DUMP TO NEW TAPE !!'
     4F 4F 44 20 44 55 4D 50
     20 54 4F 20 4E 45 57 20
     54 41 50 45 20 21 21
5428 18        04220 VERM3L  DEFM    $-VERMS3
5429 50        04230 VERMS4  DEFM    'PRESS  * ENTER *  WHEN READY'
     52 45 53 53 20 20 2A 20
     45 4E 54 45 52 20 2A 20
     20 57 48 45 4E 20 52 45
     41 44 59
001C           04240 VERM4L  EQU     $-VERMS4
5445 53        04250 SAVMS1  DEFM    'SAVEING PROGRAM'
     41 56 45 49 4E 47 20 50
     52 4F 47 52 41 4D
000F           04260 SAVM1L  EQU     $-SAVMS1
5454 50        04270 SAVMS2  DEFM    'PROGRAM SAVED ON NEW TAPE'
     52 4F 47 52 41 4D 20 53
     41 56 45 44 20 4F 4E 20
     4E 45 57 20 54 41 50 45
0019           04280 SAVM2L  EQU     $-SAVMS2
546D 50        04290 SAVMS3  DEFM    'PREPARE TO RECORD ON NEW TAPE'
     52 45 50 41 52 45 20 54
     4F 20 52 45 43 4F 52 44
     20 4F 4E 20 4E 45 57 20
     54 41 50 45
001D           04300 SAVM3L  EQU     $-SAVMS3
548A 20        04310 BRKMS   DEFM    '        OR  - -  PRESS  M  FOR MEN
     U'
     20 20 20 20 20 20 4F 52
     20 20 2D 20 2D 20 20 50
     52 45 53 53 20 20 4D 20
     20 46 4F 52 20 4D 45 4E
     55
0022           04320 BRKMSL  EQU     $-BRKMS
54AC 46        04330 SCMS1   DEFM    'FILE NAME IS
     49 4C 45 20 4E 41 4D 45
     20 49 53 20 20 20 20
```

Fig. 18-10. *Continuação da pág. 149.*

```
0010            04340 SCMS1L  EQU     $-SCMS1
54BC 53         04350 SCMS2   DEFM    'START @  '
     54 41 52 54 20 40 20 20
0009            04360 SCMS2L  EQU     $-SCMS2
54C5 20         04370 SCMS3   DEFM    '       END @    '
     20 20 20 20 20 45 4E 44
     20 40 20 20 20
000E            04380 SCMS3L  EQU     $-SCMS3
54D3 20         04390 SCMS4   DEFM    '       ENTRY @    '
     20 20 20 20 20 45 4E 54
     52 59 20 40 20 20 20
0010            04400 SCMS4L  EQU     $-SCMS4
54E3 50         04410 QUIT1   DEFM    'PRESS  * ENTER *  TO RETURN TO BA
     SIC'
     52 45 53 53 20 20 2A 20
     45 4E 54 45 52 20 2A 20
     20 54 4F 20 52 45 54 55
     52 4E 20 54 4F 20 42 41
     53 49 43
0024            04420 QUIT1L  EQU     $-QUIT1
0002            04430 BITBUF  DEFS    2
0002            04440 STABUF  DEFS    2
0002            04450 ENDBUF  DEFS    2
0002            04460 T1BUF   DEFS    2
0002            04470 ENTBUF  DEFS    2
5511            04480 TXTBUF  EQU     ENTBUF+2
551A            04490 HLDBUF  EQU     TXTBUF+9
5000            04500         END     START
00000 Total errors
```

Fig. 18-10. *Continuação da pág. 150.*

ço inicial do programa mas sim o endereço onde este específico bloco deve iniciar a sua carga. É importante observar este ponto, porque alguns autores costumam fazer a carga de seus programas em zigue-zague, ou seja, carregar o primeiro bloco em 5000H, o segundo em 6000H, o seguinte em 5500H e assim por diante. Embora todos os blocos sejam usados pelo programa, o autor, possivelmente, agiu assim para tornar mais difícil a interpretação ou cópia do programa, ou então ele pode ter sido montado com um montador Assembler especial, que permite a composição de um programa a partir de diversos outros.

Depois da leitura do número de bytes, indicado no byte de contagem, o computador lê o byte de verificação de soma (checksum byte). O valor deste deve bater com o número de bytes lidos pelo sistema, caso contrário obteremos a familiar mensagem *C, que significa que houve problemas na carga e que se deve tentar de novo.

O próximo byte que é lido é um outro header. Aqui o sistema tem que detetar se irá ler mais um bloco de dados, 3CH, ou se acabou de ler o último bloco do programa, 78H.

Se ele encontrar um 3CH, percorrerá o mesmo processo mais uma vez, começando pelo byte de contagem. Se não, ele busca os dois últimos bytes, que constituem o ponto de carga do programa. Quando obtém este endereço, ele passará o controle para este ponto, após você ter teclado / e apertado a tecla de entrada.

Agora que você já sabe como uma fita de sistema é carregada no computador, vamos examinar o programa de cópia.

A linha 100 é o comando ORG. Este ponto pode ser alterado mas o tamanho do programa que pode ser copiado será reduzido. Este programa utiliza um buffer interno para armazenar os dados lidos da fita. Todos os dados que chegam vão para o buffer e não para o endereço de memória indicado na fita.

As linhas de 110 a 350 limpam a tela e exibem um menu.

As linhas de 390 a 430 apagam o cursor. Se você desejar deixá-lo, elimine estas linhas.

As linhas de 470 a 560 chamam a rotina, em ROM, de varredura da tela e detetam a tecla pressionada. Se nenhuma tecla foi pressionada a rotina é repetida. Se a seleção feita for válida, o programa desvia para a rotina apropriada. Entradas inválidas causam a repetição da execução da rotina de varredura (scan).

A linha 600 inicia a rotina de carga (LOAD). A primeira coisa que ela faz é zerar o buffer de memória temporária. A seguir, ela exibe uma mensagem na tela que lhe permite optar por continuar ou voltar ao menu. Isto é importante, porque você pode ter feito uma escolha equivocada, como, por exemplo, comandar uma carga (load) em lugar de mandar salvar (save) ou verificar (verify) o programa. Se você realmente quer fazer a carga do programa na memória, ela então continua na linha 680.

As linhas de 680 a 720 exibem a mensagem de carga de programa.

As linhas 730 a 760 zeram os buffers restantes e fazem a preparação para a carga do programa na memória.

A linha 770 posiciona o ponteiro no início do buffer de texto do programa.

A linha 780 define o mecanismo de fita a ser usado.

A linha 790 chama a rotina, em ROM, que localiza o byte de sincronismo no início da fita.

A linha 800 chama a rotina ROM para a leitura de um byte da fita.

File header	55H	1 byte
File name		6 bytes
Data header	3CH	1 byte
Byte count		1 byte
Starting address		2 bytes
Data		up to 255 bytes
Checksum		1 byte
Entry header	78H	1 byte
Entry address		2 bytes

Fig. 18-11. *Formato das Fitas de Sistema.*

As linhas de 840 a 870 incrementam os ponteiros e contadores e verificam se o byte lido é um header de arquivo.

A linha 880 indica ao programa para que lhe envie uma mensagem de falha na carga, caso o byte lido tenha sido um header de arquivo. Neste ponto, *não deveríamos ter encontrado um header de arquivo.*

As linhas 920 a 970 executam a carga dos próximos 6 bytes. Eles constituem o nome do arquivo da fita.

As linhas 1010 até 1060 buscam o próximo byte e verificam se se trata de um header de dados. Se não for, o programa lhe enviará uma mensagem de falha na carga (bad load). Se for, os ponteiros são incrementados, assim como os contadores.

As linhas de 1100 a 1140 lêem o próximo byte e posicionam os contadores e ponteiros em função da informação ali contida.

As linhas 1180 a 1230 obtêm o endereço de início da carga do bloco que será lido. Os ponteiros e contadores são novamente incrementados e posicionados.

As linhas 1270 a 1340 lêem o bloco de dados propriamente dito.

As linhas 1380 a 1410 lêem o byte de verificação da soma (Checksum).

As linhas 1450 a 1530 lêem o próximo header e testam-no para ver se se trata de header de dados ou de entrada (entry). Se for um header de dados, o programa retorna a LL3 e lê o próximo bloco. Se não for header de dados ou de entrada, que assinalaria fim dos blocos, o programa desvia para a rotina de falha na carga.

As linhas 1570 a 1630 lêem o endereço do ponto de carga e o armazenam.

As linhas 1640 até 1700 desligam o motor da unidade de fita, limpam a tela e desviam para BL2.

As linhas 1740 a 1800 desligam o motor, limpam a tela e exibem a mensagem BAD LOAD (carga malsucedida). O programa, então, desvia para BL2.

As linhas 1840 a 1980 limpam a tela, exibem uma mensagem para que a fita seja salva e chamam uma rotina que ainda fornece uma última chance para continuação do processamento (BREAK).

As linhas 1990 a 2170 definem o mecanismo de fita, ligam o motor, gravam a informação inicial na fita (leader) e, então, fazem uma transferência bit a bit dos dados da memória de buffer para a nova fita. O buffer contém todos os dados e headers da fita original e, desta forma, não precisamos nos preocupar em colocar headers na nova fita.

As linhas 2210 até 2350 limpam a tela e exibem as necessárias mensagens para verificação da fita.

As linhas 2360 a 2390 posicionam os ponteiros do buffer e o mecanismo de fita para a rotina de leitura-verificação bit a bit.

As linhas 2400 a 2560 lêem um byte da fita. Este byte é, então, comparado com o conteúdo do buffer. Se não houver coincidência, o programa salta para a rotina de falha na carga. Se eles coincidem, o ponteiro é incrementado e o próximo byte é lido. Observe aqui que o par de registradores BC é usado para acompanhar o número de bytes lidos para a rotina de verificação.

As linhas 2600 a 2660 constituem a rotina de saída do programa (exit). Aqui, o programa lhe dá a costumeira chance de retornar ao menu. Se você optar por terminar a execução, o programa faz um desvio para 1A19H, o ponto de execução BASIC READY. Os usuários de sistemas com discos podem alterar este endereço, fazendo o programa retornar ao estado DOS READY. Para tal, você deve obter aquele endereço no seu manual de operações do DOS.

As linhas 2700 a 2800 são as linhas que constituem a rotina de última chance. Aqui você recebe a mensagem para confirmar a seleção feita. Pressionando-se a tecla de entrada (ENTER) a rotina fará um retorno à rotina que a invocou. Pressionando-se um 'M' faremos o programa retornar ao menu principal. Observe que, em vez de usar a rotina de varredura do teclado localizada em 02BH, nós usamos a rotina residente em 049H. Esta rotina não retorna à rotina que a invocou, neste caso a rotina BREAK, até que uma tecla seja pressionada. Você também pode constatar que as únicas entradas válidas são a tecla de entrada (ENTER), sem nenhum caracter, e a tecla M. Qualquer outra tecla entrada fará o programa voltar a pesquisar por uma nova entrada.

As linhas 2810 a 2880 constituem a rotina para a contagem do número total de bytes do programa que foram lidos. Isto é feito através da soma dos conteúdos dos campos TIBUF e STABUF. Um terceiro buffer, STABUF, é usado aqui para ficar com o conteúdo de HLDBUF.

As linhas 2890 a 3000 computam o endereço da última posição.

As linhas 3010 a 3230 exibem as mensagens sobre os endereços de início, de fim e do ponto de carga, bem como os endereços respectivos.

As linhas 3240 a 3500 constituem a rotina de conversão de hexadecimal para ASCII, que nós usamos no programa de cópia da memória. Aqui, ela é usada para converter os conteúdos dos buffers, para que eles possam ser exibidos na tela.

As linhas 3590 a 3650 representam uma rotina simples e rápida para exibir o conteúdo do par HL na tela.

As linhas de 3690 a 3809 contêm a rotina para fazer piscar o caracter* no canto direito superior da tela. Eu sei que existe uma rotina em ROM que executaria isto para você, mas vamos ser um pouco criativos aqui. Esta rotina verifica se existe um asterisco na tela. Se existir, ela exibe um espaço (branco) naquele local, se não, ela exibe um asterisco. O asterisco *não* pisca apenas no fim de cada bloco, como o faz na rotina em ROM. Aqui, ele pisca constantemente e, assim, nós sabemos que o programa está fazendo alguma coisa enquanto aguardamos. A rotina ROM faz o asterisco piscar depois da cópia de cada bloco. Você pode modificar este programa para fazer a mesma coisa. Vou dar-lhe uma dica: tudo o que você precisa fazer é deslocar uma, duas ou três linhas da codificação. Mudar de posição o que, para onde, é o que eu quero que você procure descobrir. . .

As linhas 3840 até 3930 lista a rotina que faz o acompanhamento de quantos bytes estão sendo lidos.

O restante do programa define as mensagens e estabelece os tamanhos dos buffers. Mais uma vez, eu enfatizo a sugestão de que você catalogue este

programa em sua biblioteca e o use. Ele pode lhe poupar muitos dissabores e dinheiro também.

CAPÍTULO 19

Do BASIC para a Linguagem de Máquina: O Programa das Formas

Sinto que, a essa altura, você deve estar dizendo: "Deve haver uma maneira mais fácil". Bom, existe e chama-se compilador BASIC. Contudo, é bom lembrar que, como acontece com quase tudo na vida, o caminho mais fácil não é o melhor sob todos os pontos de vista.

Um compilador BASIC lê um programa escrito em linguagem BASIC e o compila, transformando-o em uma versão em linguagem de máquina. O princípio que norteia o processo é que o compilador deve tentar formar rotinas que possam ser utilizadas repetidas vezes ao longo do programa compilado. A partir daí ele faz a tradução das instruções para a linguagem de máquina.

Existem algumas desvantagens quanto ao uso da compilação BASIC:

1. O compilador, em si, toma 12K bytes de memória
2. É necessário possuir um sistema com disco para o uso do compilador.
3. Você não pode compilar um programa tão extenso quanto se utilizasse Assembler (montagem do programa), ou, diretamente, a linguagem de máquina.

Nós não iremos compilar um programa aqui mas vamos dar uma demonstração de como converter um programa em BASIC para linguagem de máquina. A propósito, eu, na verdade, cheguei a compilar o programa que vamos ver a seguir, para ver qual o tamanho da listagem final.

Foram necessárias 147 páginas de papel para listá-lo. Uma das razões pelas quais o programa compilado se torna tão extenso é que o compilador tem que ser incorporado dentro do programa para que possa funcionar.

O programa que veremos desenha dois retângulos e um quadrado na tela. O quadrado é posicionado aleatoriamente na posição 1, 2 ou 3. Eu escrevi este programa para um programa de desenvolvimento mental de crianças de uma escola local, para treiná-las em descobrir a forma que não coincidisse.

Vejamos a conversão de um programa em BASIC para a linguagem Assembler. A primeira coisa de que necessitamos é do programa em BASIC. Pa-

ra construí-lo, precisamos planejar o que queremos que o programa execute. Neste caso, temos um cliente que precisa de um programa que exiba três formas na tela. Duas dessas formas devem ser iguais e a terceira, diferente.

Nosso cliente deseja utilizar esta programação como uma ferramenta educacional. O seu objetivo é ensinar a crianças em fase pré-escolar a diferenciar formas. Nosso cliente também pretende ensinar às crianças a contar até dez.

Mantendo o programa simples e oferecendo recompensas ao aluno, vamos fazer o computador desenhar dois retângulos e um quadrado. Estas figuras serão numeradas, sendo que o quadrado será localizado aleatoriamente em uma das três posições de figuras. Cada figura será numerada, o que nos permitirá, ao mesmo tempo, ensinar os três primeiros números. A versão BASIC do programa está mostrada na Fig. 19-1.

A linha 10 limpa um espaço para a fila, limpa a tela e move o cursor duas linhas para baixo, a partir do topo. Ela também invoca o gerador de números aleatórios, para obter um número qualquer entre 1 e 3 na variável N. A seguir ela testa o valor de N e, dependendo deste teste, desvia para a rotina do formato correspondente.

A linha 100 imprime os números de 1 a 3 nos locais adequados na tela e, a seguir, chama a rotina para exibir o qudrado e os retângulos. Ela, então, posiciona o valor de J em 1. Isto servirá para indicar ao computador que o quadrado se encontra na posição 1. Após isto feito, ela desvia para a linha 400, onde iremos verificar a resposta do estudante. As linhas 200 e 300 fazem um procedimento idêntico, com a localização do quadrado na posição 2 e 3, respectivamente.

A linha 400 solicita à criança que indique que figura está errada. Ela, a seguir, testa se a criança apertou a tecla correta. Se tal ocorreu, ela desvia para a linha 420.

A linha 410 informa à criança que sua resposta está errada. Ela também dá ao aluno a resposta certa e desvia para a linha 430. A linha 420 diz à criança que ela acertou e desvia para a linha 440. A linha 430 pede à criança para apertar a tecla branca para continuar.

A linha 440 constitui uma rotina de varredura INKEY$, que verifica a tecla de entrada. Se ela não foi pressionada, ela verifica de novo. Se ela foi pressionada, o programa volta ao seu início. Uma observação interessante aqui é que a rotina INKEY$ é um "call" (chamada) da rotina em 02BH, e o valor CHR$(13) é 0DH, ou seja, o hexadecimal correspondente ao retorno do cursor.

A linha 1000 desenha um quadrado na posição 1.
A linha 2000 desenha um quadrado na posição 2.
A linha 3000 desenha um quadrado na posição 3.
A linha 4000 desenha um retângulo na posição 1.
A linha 5000 desenha um retângulo na posição 2.
A linha 6000 desenha um retângulo na posição 3.

```
10 CLEAR70:CLS:PRINT:PRINT:N=RND(3):ONNGOTO100,200,300
100 PRINTTAB(5)"1"TAB(30)"2"TAB(50)"3":GOSUB1000:
    GOSUB5000:GOSUB6000:J=1:GOTO400
200 PRINTTAB(10)"1"TAB(30)"2"TAB(50)"3":GOSUB4000:
    GOSUB2000:GOSUB6000:J=2:GOTO400
300 PRINTTAB(10)"1"TAB(30)"2"TAB(50)"3":GOSUB4000:
    GOSUB5000:GOSUB3000:J=3
400 PRINT@400,:::INPUT"WHICH FIGURE IS WRONG";A:IFA=JTHEN420
410 PRINT@400,:::PRINTSTRING$(40," "):PRINT@400,:::
    PRINT"WRONG IT WAS ";J:GOTO430
420 PRINT@400,:::PRINT"GOOD    YOU GOT IT RIGHT  ":PRINT"
    PRESS THE WHITE KEY TO CONTINUE":GOTO440
430 PRINT"PRESS THE WHITE KEY TO CONTINUE"
440 A$=INKEY$:IFA$=CHR$(13)THENGOTO10ELSE440
1000 FORX=1TO25:SET(X,1):SET(X,10):NEXTX:FORY=1
    TO10:SET(1,Y):SET(25,Y):NEXTY:RETURN
2000 FORX=51TO75:SET(X,1):SET(X,10):NEXTX:FORY=1
    TO10:SET(51,Y):SET(75,Y):NEXTY:RETURN
3000 FORX=85TO110:SET(X,1):SET(X,10):NEXTX:FORY=1
    TO10:SET(85,Y):SET(110,Y):NEXTY:RETURN
4000 FORX=1TO35:SET(X,1):SET(X,10):NEXTX:FORY=1
    TO10:SET(1,Y):SET(35,Y):NEXTY:RETURN
5000 FORX=41TO76:SET(X,1):SET(X,10):NEXTX:FORY=1
    TO10:SET(41,Y):SET(76,Y):NEXTY:RETURN
6000 FORX=86TO121:SET(X,1):SET(X,10):NEXTX:FORY=1
    TO10:SET(86,Y):SET(121,Y):NEXTY:RETURN
```

Fig. 19-1. *O Programa das Formas em BASIC.*

Está pronto. Um programa simples, em BASIC, para atender às necessidades de nosso cliente. Agora, façamos sua conversão para linguagem de máquina. O princípio é o mesmo. Buscamos atender às mesmas necessidades do cliente e, na maior parte do programa, podemos seguir o mesmo fluxo lógico da programação em BASIC. A conversão está mostrada na Fig. 19.2.

```
00100 ;
00110 ;  ORIGIN POINT MAY BE CHANGED IF DESIRED
00120 ;
00130                    ORG     8000H
00140 ;
00150 ;  CLEAR THE SCREEN
00160 ;
00170          START     CALL    01C9H
00172 ;
00174 ;TOSS THE COIN FOR A NUMBER FROM 1 TO 3
00176 ;
00180          TOSS      LD      A,R       ;GET CONTENTS OF
REFRESH REGISTER
00190                    LD      B,A       ;STORE IN THE B
REGISTER
00200                    LD      A,R       ;GET IT AGAIN (IT
CONTINUALY CHANGES)
00210                    LD      C,A       ;STORE IT IN THE C
REGISTER
00220          DELAY     DEC     BC        ;DELAY TO BE SURE
IT'S NOT THE SAME
00230                    LD      A,B
00240                    OR      C
00250                    JR      NZ,DELAY
00260                    LD      A,R       ;NOW IT'S MIXED UP
GOOD !!!
00270                    AND     03H       ;MASK IT
00280                    LD      (RND),A
00290                    CP      1
00300                    JP      Z,ONE
00310                    CP      2
00320                    JP      Z,TWO
00330                    CP      3
00340                    JP      Z,THREE
00350                    JP      TOSS      ;IF IT IS MORE TRY
AGAIN
00360          ONE       LD      DE,3C00H+64+10
00370                    LD      A,'1'
00380                    LD      (DE),A
00390                    LD      DE,3C00H+64+29
00400                    LD      A,'2'
00410                    LD      (DE),A
00420                    LD      DE,3C00H+64+51
00430                    LD      A,'3'
00440                    LD      (DE),A
00450                    LD      DE,3C00H+6
00460                    LD      A,140
00470                    LD      BC,10
00480                    CALL    DRAW
00490                    LD      DE,3C00H+192+6
00500                    LD      BC,10
00510                    LD      A,140
00520                    CALL    DRAW
00530                    LD      DE,3C00H+5
00540                    CALL    SIDE1
00550                    LD      DE,3C00H+16
00560                    CALL    SIDE2
00570                    LD      DE,3C00H+21
00580                    LD      BC,17
00590                    LD      A,140
00600                    CALL    DRAW
00610                    LD      DE,3C00H+192+21
00620                    LD      BC,17
```

Fig. 19-2. *O Programa das Formas em Linguagem de Máquina.*

```
00630                   LD      A,140
00640                   CALL    DRAW
00650                   LD      DE,3C00H+20
00660                   CALL    SIDE1
00670                   LD      DE,3C00H+20+18
00680                   CALL    SIDE2
00690                   LD      DE,3C00H+44
00700                   LD      BC,17
00710                   LD      A,140
00720                   CALL    DRAW
00730                   LD      DE,3C00H+192+44
00740                   LD      BC,17
00750                   LD      A,140
00760                   CALL    DRAW
00770                   LD      DE,3C00H+43
00780                   CALL    SIDE1
00790                   LD      DE,3C00H+43+17
00800                   CALL    SIDE2
00802 ;
00804 ;  SAVE A ONE IN THE RND BUFFER  (IT HAD AN A IN IT)
00806 ;
00810                   LD      A,'1'
00820                   LD      (RND),A
00830                   JP      ANSWER
00840           TWO     LD      DE,3C00H+64+8
00850                   LD      A,'1'
00860                   LD      (DE),A
00870                   LD      DE,3C00H+64+27
00880                   LD      A,'2'
00890                   LD      (DE),A
00900                   LD      DE,3C00H+64+46
00910                   LD      A,'3'
00920                   LD      (DE),A
00930                   LD      DE,3C00H+1
00940                   LD      BC,17
00950                   LD      A,140
00960                   CALL    DRAW
00970                   LD      DE,3C00H+192+1
00980                   LD      BC,17
00990                   LD      A,140
01000                   CALL    DRAW
01010                   LD      DE,3C00H
01020                   CALL    SIDE1
01030                   LD      DE,3C00H+18
01040                   CALL    SIDE2
01050                   LD      DE,3C00H+22+1
01060                   LD      BC,10
01070                   LD      A,140
01080                   CALL    DRAW
01090                   LD      DE,3C00H+192+22+1
01100                   LD      BC,10
01110                   LD      A,140
01120                   CALL    DRAW
01130                   LD      DE,3C00H+22
01140                   CALL    SIDE1
01150                   LD      DE,3C00H+22+11
01160                   CALL    SIDE2
01170                   LD      DE,3C00H+38+1
01180                   LD      BC,17
01190                   LD      A,140
01200                   CALL    DRAW
01210                   LD      DE,3C00H+192+38+1
01220                   LD      BC,17
```

Fig. 19-2. *Continuação da pág. 159.*

```
01230                   LD      A,140
01240                   CALL    DRAW
01250                   LD      DE,3C00H+38
01260                   CALL    SIDE1
01270                   LD      DE,3C00H+38+17+1
01280                   CALL    SIDE2
01282
01284 ;  STORE A 2 IN RND
01286 ;
01290                   LD      A,'2'
01300                   LD      (RND),A
01310                   JP      ANSWER
01320           THREE   LD      DE,3C00H+64+8
01330                   LD      A,'1'
01340                   LD      (DE),A
01350                   LD      DE,3C00H+64+31
01360                   LD      A,'2'
01370                   LD      (DE),A
01380                   LD      DE,3C00H+64+51
01390                   LD      A,'3'
01400                   LD      (DE),A
01410                   LD      DE,3C00H+1
01420                   LD      BC,17
01430                   LD      A,140
01440                   CALL    DRAW
01450                   LD      DE,3C00H+192+1
01460                   LD      BC,17
01470                   LD      A,140
01480                   CALL    DRAW
01490                   LD      DE,3C00H
01500                   CALL    SIDE1
01510                   LD      DE,3C00H+18
01520                   CALL    SIDE2
01522 ;
01524 ;  STORE A THREE IN RND
01526 ;
01530                   LD      A,'3'
01540                   LD      (RND),A
01550                   LD      DE,3C00H+18+5+1
01560                   LD      A,140
01570                   LD      BC,17
01580                   CALL    DRAW
01590                   LD      DE,3C00H+192+18+5+
01600                   LD      BC,17
01610                   LD      A,140
01620                   CALL    DRAW
01630                   LD      DE,3C00H+18+5
01640                   CALL    SIDE1
01650                   LD      DE,3C00H+18+5+18
01660                   CALL    SIDE2
01670                   LD      DE,3C00H+46+1
01680                   LD      BC,10
01690                   LD      A,140
01700                   CALL    DRAW
01710                   LD      DE,3C00H+192+46+1
01720                   LD      BC,10
01730                   LD      A,140
01740                   CALL    DRAW
01750                   LD      DE,3C00H+46
01760                   CALL    SIDE1
01770                   LD      DE,3C00H+46+11
01780                   CALL    SIDE2
01790                   JP      ANSWER
```

```
01792 ;
01794 ;  THIS DRAWS THE TOPS AND BOTTOMS
01796 ;
01800           DRAW    LD      (TEMP),BC
01810                   LD      (TEMP1),A
01820           DRAW1   LD      A,(TEMP1)
01830                   LD      (DE),A
01840                   INC     DE
01850                   DEC     BC
01860                   LD      A,B
01870                   OR      C
01880                   JR      NZ,DRAW1
01890                   RET
01900           TEMP    DEFS    2
01902 ;
01904 ;  NOW GET AND CHECK THE ANSWER
01906 ;
01910           ANSWER  LD      DE,3C00H+455
01920                   LD      HL,MESS1
01930                   LD      BC,MESS1L
01940                   LDIR
01950                   LD      A,(RND)
01960                   LD      H,A
01970           ANSL    CALL    02BH
01980                   JR      Z,ANSL
01990                   CP      H
02000                   JR      Z,RIGHT
02010                   JP      WRONG
02012 ;
02014 ;  YOU DID IT !!!!
02016 ;
02020           RIGHT   CALL    01C9H
02030                   LD      DE,3C00H+70
02040                   LD      HL,MESS2
02050                   LD      BC,MESS2L
02060                   LDIR
02070                   JP      WAIT
02072 ;
02074 ;  OOPS, YOU GOOFED UP!
02076 ;
02080           WRONG   CALL    01C9H
02090                   LD      DE,3C00H+448
02100                   LD      HL,MESS3
02110                   LD      BC,MESS3L
02120                   LDIR
02130                   INC     DE
02140                   LD      A,(RND)
02150                   LD      (DE),A
02160                   JP      WAIT
02170           RND     DEFS    2
02180           TEMP1   DEFS    2
02182 ;
02184 ;  DRAW THE LEFT SIDE OF THE FIGURES
02186 ;
02190           SIDE1   LD      A,168
02200                   LD      (DE),A
02210                   CALL    UP64
02220                   LD      A,170
02230                   LD      (DE),A
02240                   CALL    UP64
02250                   LD      (DE),A
02260                   CALL    UP64
02270                   LD      A,138
```

Fig. 19-2. *Continuação da pág. 161.*

```
02280                   LD      (DE),A
02290                   RET
02292 :
02294 :  DRAW THE RIGHT SIDE OF THE FIGURES
02296 :
02300           SIDE2   LD      A,148
02310                   LD      (DE),A
02320                   CALL    UP64
02330                   LD      A,149
02340                   LD      (DE),A
02350                   CALL    UP64
02360                   LD      (DE),A
02370                   CALL    UP64
02380                   LD      A,133
02390                   LD      (DE),A
02400                   RET
02402 :
02404 :  INCREMENT DE 64 TIMES AND PUT IT BELOW WHERE IT IS
NOW
02406 :
02410           UP64    PUSH    AF
02420                   LD      BC,64
02430           UP64A   INC     DE
02440                   DEC     BC
02450                   LD      A,B
02460                   OR      C
02470                   JR      NZ,UP64A
02480                   POP     AF
02490                   RET
02492 :
02494 :  WAIT FOR THE ENTER KEY TO BE PRESSED
02496 :
02500           WAIT    LD      DE,3C00H+650
02510                   LD      HL,MESS4
02520                   LD      BC,MESS4L
02530                   LDIR
02540           WAIT1   CALL    02BH
02550                   CP      0DH
02560                   JR      NZ,WAIT1
02570                   JP      START
02580           MESS1   DEFM    'WHICH FIGURE IS WRONG?'
02590           MESS1L  EQU     $-MESS1
02600           MESS2   DEFM    'G O O D   YOU  GOT  IT
RIGHT'
02610           MESS2L  EQU     $-MESS2
02620           MESS3   DEFM    'SORRY, WRONG ANSWER.  IT
WAS ..
02630           MESS3L  EQU     $-MESS3
02640           MESS4   DEFM    'PLEASE PRESS THE WHITE KEY
TO TRY ANOTHER ONE'
02650           MESS4L  EQU     $-MESS4
02660                   END     START
```

APÊNDICE A - *Códigos*

CÓDIGOS HEXA DE TECLAS E DE CONTROLE

CÓDIGO	FUNÇÃO	CÓDIGO	FUNÇÃO
00H	NENHUMA	10H	NENHUMA
01H	NENHUMA	11H	NENHUMA
02H	NENHUMA	12H	NENHUMA
03H	NENHUMA	13H	NENHUMA
04H	NENHUMA	14H	NENHUMA
05H	NENHUMA	15H	NENHUMA
06H	NENHUMA	16H	NENHUMA
07H	NENHUMA	17H	MODO DE 32 CARACTERES
08H	RETROCESSO	18H	RETROCESSO
09H	NENHUMA	19H	AVANÇO DO CURSOR
0AH	SALTA LINHA C/RETORNO DO CURSOR	1AH	SALTO P/BAIXO
0BH	TOPO DA PÁGINA	1BH	SALTO P/CIMA
0CH	TOPO DA PÁGINA	1CH	RETORNO A 3C00H
0DH	RETORNO DO CARRO OU CURSOR	1DH	RETORNO AO INÍCIO DA LINHA
0EH	LIGAR O CURSOR	1EH	APAGAR ATÉ O FIM DA LINHA
0FH	DESLIGAR O CURSOR	1FH	APAGAR ATÉ O FIM DA TELA.

CÓDIGOS DE CARACTERES

HEXA	CARACTER	HEXA	CARACTER
20H	SPACE	30H	0
21H	!	31H	1
22H	"	32H	2
23H	#	33H	3
24H	$	34H	4
25H	%	35H	5
26H	&	36H	6
27H	'	37H	7
28H	(	38H	8
29H	)	39H	9
2AH	*	3AH	:
2BH	+	3BH	;
2CH	,	3CH	<
2DH	-	3DH	=
2EH	.	3EH	>
2FH	/	3FH	?
40H	@	50H	P
41H	A	51H	Q
42H	B	52H	R
43H	C	53H	S

44H	D	54H	T
45H	E	55H	U
46H	F	56H	V
47H	G	57H	W
48H	H	58H	X
49H	I	59H	Y
4AH	J	5AH	X
4BH	K	5BH	SETA P/CIMA
4CH	L	5CH	SETA P/BAIXO
4DH	M	5DH	SETA P/ESQUERDA
4EH	N	5EH	SETA P/DIREITA
4FH	O	5FH	—

CÓDIGOS P/LETRAS MINÚSCULAS

60H	(a	70H	P
61H	a	71H	q
62H	b	72H	r
63H	c	73H	s
64H	d	74H	t
65H	e	75H	u
66H	f	76H	v
67H	g	77H	w
68H	h	78H	x
69H	i	79H	y
6AH	j	7AH	z
6BH	k	7BH	NENHUMA
6CH	l	7CH	NENHUMA
6DH	m	7DH	NENHUMA
6EH	n	7EH	NENHUMA
6FH	□	7FH	NENHUMA

APÊNDICE B – *Sufixos JR*

DESLOCAMENTO P/TRÁS A PARTIR DO FIM DO COMANDO

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

DESLOCAMENTO À FRENTE DO COMANDO

JR	DESLOC.	JR	DESLOC.	JR	DESLOC.
00	0	30	48	60	96
01	1	31	49	61	97
02	2	32	50	62	98
03	3	33	51	63	99
04	4	34	52	64	100
05	5	35	53	65	101
06	6	36	54	66	102
07	7	37	55	67	103
08	8	38	56	68	104
09	9	39	57	69	105
0A	10	3A	58	6A	106
0B	11	3B	59	6B	107
0C	12	3C	60	6C	108
0D	13	3D	61	6D	109
0E	14	3E	62	6E	110
0F	15	3F	63	6F	111
10	16	40	64	70	112
11	17	41	65	71	113
12	18	42	66	72	114
13	19	43	67	73	115
14	20	44	68	74	116
15	21	45	69	75	117
16	22	46	70	76	118
17	23	47	71	77	119
18	24	48	72	78	120
19	25	49	73	79	121
1A	26	4A	74	7A	122
1B	27	4B	75	7B	123
1C	28	4C	76	7C	124
1D	29	4D	77	7D	125
1E	30	4E	78	7E	126
1F	31	4F	79	7F	127
20	32	50	80	80	128
21	33	51	81		
22	34	52	82		
23	35	53	83		
24	36	54	84		
25	37	55	85		
26	38	56	86		
27	39	57	87		
28	40	58	88		
29	41	59	89		
2A	42	5A	90		
2B	43	5B	91		
2C	44	5C	92		
2D	45	5D	93		
2E	46	5E	94		
2F	47	5F	95		

APÊNDICE C – *Lista Numérica dos Códigos de Máquina*

CÓDIGO DE MÁQUINA	COMANDO	CÓDIGO DE MÁQUINA	COMANDO
00	NOP	35	DEC (HL)
018405	LD (BC),NN	3620	LD (HL),N
02	LD (BC),A	37	SCF
03	INC BC	382E	JR C,DIS
04	INC B	39	ADD HL,SP
05	DEC B	3A8405	LD A,(NN)
0620	LD B,N	3B	DEC SP
07	RLC	3C	INC A
08	EX AF,AF'	3D	DEC A
09	ADD HL,BC	3E20	LD A,N
0A	LD A,(BC)	3F	CCF
0B	DEC B	40	LD B,B
0C	INC C	41	LD B,C
0D	DEC C	42	LD B,D
0E20	LD C,N	43	LD B,E
0F	RRCA	44	LD B,H
102E	DJNZ DIS	45	LD B,L
118405	LD DE,NN	46	LD B,(HL)
12	LD (DE),A	47	LD B,A
13	INC DE	48	LD C,B
14	INC D	49	LD C,C
15	DEC D	4A	LD C,D
1620	LD D,D	4B	LD C,E
17	RLA	4C	LD C,H
182E	JR DIS	4D	LD C,L
19	ADD HL,DE	4E	LD C,(HL)
1A	LD A,(DE)	4F	LD C,A
1B	DEC DE	50	LD D,B
1C	INC E	51	LD D,C
1D	DEC E	52	LD D,D
1E20	LD E,N	53	LD D,E
1F	RRA	54	LD D,H
202E	JR NZ,DIS	55	LD D,L
218405	LD HL,NN	56	LD D,(HL)
228405	LD (NN),HL	57	LD D,A
23	INC HL	58	LD E,B
24	INC H	59	LD E,C
25	DEC H	5A	LD E,D
2620	LD H,N	5B	LD E,E
27	DAA	5C	LD E,H
282E	JR Z,DIS	5D	LD E,L
29	ADD HL,(NN)	5E	LD E,(HL)
2A8405	LD HL,(NN)	5F	LD E,A
2B	DEC HL	60	LD H,B
2C	INC L	61	LD H,C
2D	DEC L	62	LD H,D
2E20	LD L,N	63	LD H,E
2F	CPL	64	LD H,H
302E	JR NC,DIS	65	LD H,L

CÓDIGO DE MÁQUINA	COMANDO	CÓDIGO DE MÁQUINA	COMANDO
318405	LD SP,NN	66	LD H,(HL)
328405	LD (NN),A	67	LD H,A
33	INC SP	68	LD L,B
34	INC (HL)	69	LD L,C
6A	LD L,D	9F	SBC A,A
6B	LD L,E	A0	AND B
6C	LD L,H	A1	AND C
6D	LD L,L	A2	AND D
6E	LD L,(HL)	A3	AND E
6F	LD L,A	A4	AND H
70	LD (HL),B	A5	AND L
71	LD (HL),C	A6	AND (HL)
72	LD (HL),D	A7	AND A
73	LD (HL),E	A8	XOR B
74	LD (HL),H	A9	XOR C
75	LD (HL),L	AA	XOR D
76	HALT	AB	XOR E
77	LD (HL),A	AC	XOR H
78	LD A,B	AD	XOR L
79	LD A,C	AE	XOR (HL)
7A	LD A,D	AF	XOR A
7B	LD A,E	B0	OR B
7C	LD A,H	B1	OR C
7D	LD A,L	B2	OR D
7E	LD A,(HL)	B3	OR E
7F	LD A,A	B4	OR H
80	ADD A,B	B5	OR L
81	ADD A,C	B6	OR (HL)
82	ADD A,D	B7	OR A
83	ADD A,E	B8	CP B
84	ADD A,H	B9	CP C
85	ADD A,L	BA	CP D
86	ADD A,(HL)	BB	CP E
87	ADD A,A	BC	CP H
88	ADC A,B	BD	CP L
89	ADC A,C	BE	CP (HL)
8A	ADC A,D	BF	CP A
8B	ADC A,E	C0	RET NZ
8C	ADC A,H	C1	POP BC
8D	ADC A,L	C28405	JP NZ,NN
8E	ADC A,(HL)	C38405	JP NN
8F	ADC A	C48405	CALL NZ,NN
90	SUB B	C5	PUSH BC
91	SUB C	C620	ADD A,N
92	SUB D	C7	RST 0
93	SUB E	C8	RET Z
94	SUB H	C9	RET
95	SUB L	CA8405	JP Z,NN
96	SUB (HL)	CC8405	CALL Z,NN
97	SUB A,B	CD8405	CALL NN
98	SBC A,B	CE20	ADC A,N
99	SBC A,C	CF	RST 8
9A	SBC A,D	D0	RET NC
9B	SBC A,E	D1	POP DE
9C	SBC A,H	D28405	JP NC,NN
9D	SBC A,L	D320	OUT N,A
9E	SBC A,(HL)	D48405	CALL NC,CC

CÓDIGO DE MÁQUINA	COMANDO	CÓDIGO DE MÁQUINA	COMANDO
CB4A	BIT 1,D	CB7F	BIT 7,A
CB4B	BIT 1,E	CB80	RES 0,B
CB4C	BIT 1,H	CB81	RES 0,C
CB4D	BIT 1,L	CB82	RES 0,D
CB4E	BIT 1,(HL)	CB83	RES 0,E
CB4F	BIT 1,A	CB84	RES 0,H
CB50	BIT 2,B	CB85	RES 0,L
CB51	BIT 2,C	CB86	RES 0,(HL)
CB52	BIT 2,D	CB87	RES 0,A
CB53	BIT 2,E	CB88	RES 1,B
CB54	BIT 2,H	CB89	RES 1,C
CB55	BIT 2,L	CB8A	RES 1,D
CB56	BIT 2,(HL)	CB8B	RES 1,E
CB57	BIT 2,A	CB8C	RES 1,H
CB58	BIT 3,B	CB8D	RES 1,L
CB59	BIT 3,C	CB8E	RES 1,(HL)
CB5A	BIT 3,D	CB8F	RES 1,A
CB5B	BIT 3,E	CB90	RES 2,B
CB5C	BIT 3,H	CB91	RES 2,C
CB5D	BIT 3,L	CB92	RES 2,D
CB5E	BIT 3,(HL)	CB93	RES 2,E
CB5F	BIT 3,A	CB94	RES 2,H
CB60	BIT 4,B	CB95	RES 2,L
CB61	BIT 4,C	CB96	RES 2,(HL)
CB62	BIT 4,D	CB97	RES 2,A
CB63	BIT 4,E	CB98	RES 3,B
CB64	BIT 4,H	CB99	RES 3,C
CB65	BIT 4,L	CB9A	RES 3,D
CB66	BIT 4,(HL)	CB9B	RES 3,E
CB67	BIT 4,A	CB9C	RES 3,H
CB68	BIT 5,B	CB9D	RES 3,L
CB69	BIT 5,C	CB9E	RES 3,(HL)
CB6A	BIT 5,D	CB9F	RES 3,A
CB6B	BIT 5,E	CBA0	RES 4,B
CB6C	BIT 5,H	CBA1	RES 4,C
CB6D	BIT 5,L	CBA2	RES 4,D
CB6E	BIT 5,(HL)	CBA3	RES 4,E
CB6F	BIT 5,A	CBA4	RES 4,H
CB70	BIT 6,B	CBA5	RES 4,L
CB71	BIT 6,C	CBA6	RES 4,(HL)
CB72	BIT 6,D	CBA7	RES 4,A
CB73	BIT 6,E	CBA8	RES 5,B
CB74	BIT 6,H	CBA9	RES 5,C
CB75	BIT 6,L	CBAA	RES 5,D
CB76	BIT 6,(HL)	CBAB	RES 5,E
CB77	BIT 6,A	CBAC	RES 5,H
CB78	BIT 7,B	CBAD	RES 5,L
CB79	BIT 7,C	CBAE	RES 5,(HL)
CB7A	BIT 7,D	CBAF	RES 5,A
CB7B	BIT 7,E	CBB0	RES 6,B
CB7C	BIT 7,H	CBB1	RES 6,C
CB7D	BIT 7,L	CBB2	RES 6,D
CB7E	BIT 7,(HL)	CBB3	RES 6,E
CBB4	RES 6,H	CBE9	SET 5,C
CBB5	RES 6,L	CBEA	SET 5,D
CBB6	RES 6,(HL)	CBEB	SET 5,E
CBB7	RES 6,A	CBEC	SET 5,H

CÓDIGO DE MÁQUINA	COMANDO	CÓDIGO DE MÁQUINA	COMANDO
CBB8	RES 7,B	CBED	SET 5,L
CBB9	RES 7,C	CBEE	SET 5,(HL)
CBBA	RES 7,D	CBEF	SET 5,A
CBBB	RES 7,E	CBF0	SET 6,B
CBBC	RES 7,H	CBF1	SET 6,C
CBBD	RES 7,L	CBF2	SET 6,D
CBBE	RES 7,(HL)	CBF3	SET 6,E
CBBF	RES 7,A	CBF4	SET 6,H
CBC0	SET 0,B	CBF5	SET 6,L
CBC1	SET 0,C	CBF6	SET 6,(HL)
CBC2	SET 0,D	CBF7	SET 6,A
CBC3	SET 0,E	CBF8	SET 7,B
CBC4	SET 0,H	CBF9	SET 7,C
CBC5	SET 0,L	CBFA	SET 7,D
CBC6	SET 0,(HL)	CBFB	SET 7,E
CBC7	SET 0,A	CBFC	SET 7,H
CBC8	SET 1,B	CBFD	SET 7,L
CBC9	SET 1,C	CBFE	SET 7,(HL)
CBCA	SET 1,D	CBFF	SET 7,A
CBCB	SET 1,E	DD09	ADD IX,BC
CBCC	SET 1,H	DD19	ADD IX+DE
CBCD	SET 1,L	DD218405	LD IX,NN
CBCE	SET 1,(HL)	DD228405	LD (NN),IX
CBCF	SET 1,A	DD23	INC IX
CBD0	SET 2,B	DD29	ADD IX,IX
CBD1	SET 2,C	DD2A8405	LD IX,(NN)
CBD2	SET 2,D	DD2B	DEC IX
CBD3	SET 2,E	DD3405	INC (IX+IND)
CBD4	SET 2,H	DD3505	DEC (IX+IND)
CBD5	SET 2,L	DD360520	LD (IX+IND),NN
CBD6	SET 2,(HL)	DD39	ADD IX,SP
CBD7	SET 2,A	DD4605	LD B,(IX+IND)
CBD8	SET 3,B	DD4E05	LD C,(IX+IND)
CBD9	SET 3,C	DD5605	LD D,(IX+IND)
CBDA	SET 3,D	DD5E05	LD E,(IX+IND)
CBDB	SET 3,E	DD6605	LD H,(IX+IND)
CBDC	SET 3,H	DD6E05	LD L,(IX+IND)
CBDD	SET 3,L	DD7005	LD (IX+IND),B
CBDE	SET 3,(HL)	DD7105	LD (IX+IND),C
CBDF	SET 3,A	DD7205	LD (IX+IND),D
CBE0	SET 4,B	DD7305	LD (IX+IND),E
CBE1	SET 4,C	DD7405	LD (IX+IND),H
CBE2	SET 4,D	DD7505	LD (IX+IND),L
CBE3	SET 4,E	DD7705	LD (IX+IND),A
CBE4	SET 4,H	DD7E05	LD A,(IX+IND)
CBE5	SET 4,L	DD8605	ADD A,(IX+IND)
CBE6	SET 4,(HL)	DD8E05	ADC A,(IX+IND)
CBE7	SET 4,A	DD9605	SUB (IX+IND)
CBE8	SET 5,B	DD9E05	SBC A,(IX+IND)
DDA605	AND (IX+IND)	ED50	IN D,(C)
DDAE05	XOR (IX+IND)	ED51	OUT (C),D
DDB605	XOR (IX+IND)	ED52	SBC HL,DE
DDBE05	CP (IX+IND)	ED538405	LD (NN),DE
DDE1	POP IX	ED56	IM 1
DDE3	EX (SP),IX	ED57	LD A,I
DDE5	PUSH IX	ED58	IN E,(C)
DDE9	JP (IX)	ED59	OUT (C),E

CÓDIGO DE MÁQUINA	COMANDO
DDF9	LD SP,IX
DDCB0506	RLC (IX+IND)
DDCB050E	RRC (IX+IND)
DDCB0516	RL (IX+IND)
DDCB051E	RR (IX+IND)
DDCB0526	SLA (IX+IND
DDCB052E	SRA (IX+IND)
DDCB053E	SRL (IX+IND)
DDCB0546	BIT 0,(IX+IND)
DDCB054E	BIT 1,(IX+IND)
DDCB0556	BIT 2,(IX+IND)
DDCB055E	BIT 3,(IX+IND)
DDCB0566	BIT 4,(IX+IND)
DDCB056E	BIT 5,(IX+IND)
DDCB0576	BIT 6,(IX+IND)
DDCB057E	BIT 7,(IX+IND)
DDCB0586	RES 0,(IX+IND)
DDCB058E	RES 1,(IX+IND)
DDCB0596	RES 2,(IX+IND)
DDCB059F	RES 3,(IX+IND)
DDCB05A6	RES 4,(IX+IND)
DDCB05AE	RES 5,(IX+IND)
DDCB05B6	RES 6,(IX+IND)
DDCB05BE	RES 7,(IX+IND)
DDCB05C6	SET 0,(IX+IND)
DDCB05CE	SET 1,(IX+IND)
DDCB05D6	SET 2,(IX+IND)
DDCB05DE	SET 3,(IX+IND)
DDCB05E6	SET 4,(IX+IND)
DDCB05EE	SET 5,(IX+IND)
DDCB05F6	SET 6,(IX+IND)
DDCB05FE	SET 7,(IX+IND)
ED40	IN B,(C)
ED41	OUT (C),B
ED42	SBC HL,BC
ED438405	LD (NN),BC
ED44	NEG
ED45	RETN
ED46	IM 0
ED47	LD I,A
ED48	IN C,(C)
ED49	OUT (C),C
ED4A	ADC HL,BC
ED4B8405	LD BC,(NN)
ED4D	RETI
FD4605	LD B,(IY+IND)
FD4E05	LD C,(IY+IND)
FD5605	LD D,(IY+IND)
FD5E05	LD E,(IY+IND)
FD6605	LD H,(IY+IND)
FD6E05	LD L,(IY+IND)
FD7005	LD (IY+IND),B
FD7105	LD (IY+IND),C
FD7205	LD (IY+IND),D
FD7305	LD (IY+IND),E
FD7405	LD (IY+IND),H
FD7505	LD (IY+IND),L

CÓDIGO DE MÁQUINA	COMANDO
ED5A	ADC HL,DE
ED5B8405	LD DE,(NN)
ED5E	IM 2
ED60	IN H,(C)
ED61	OUT (C),H
ED62	SBC HL,HL
ED67	RRD
ED68	IN L,(C)
ED69	OUT (C),L
ED6A	ADC HL,HL
ED6F	RLD
ED72	SBC HL,SP
ED738405	LD (NN),SP
ED78	IN A,(C)
ED79	OUT (C),A
ED7A	ADC HL,SP
ED7B8405	LD SP,(NN)
EDA0	LDI
EDA1	CPI
EDA2	INI
EDA3	OUTI
EDA8	LDD
EDA9	CPD
EDAA	IND
EDAB	OUTD
EDB0	LDIR
EDB1	CPIR
EDB2	INIR
EDB3	OTIR
EDB8	LDDR
EDB9	CPDR
EDBA	INDR
EDBB	OTDR
FD09	ADD IY,BC
FD19	ADD IY,DE
FD218405	LD IY,NN
FD228405	LD (NN),IY
FD23	INC IY
FD29	ADD IY,IY
FD2A8405	LD IY,(NN)
FD2B	DEC IY
FD3405	INC (IY+IND)
FD3505	DEC (IY+IND)
FD360520	LD (IY+IND),N
FD39	ADD IY+SP
FDCB05DE	SET 3,(IY+IND)
FDCB05E6	SET 4,(IY+IND)
FDCB05EE	SET 5,(IY+IND)
FDCB05F6	SET 6,(IY+IND)
FDCB05FE	SET 7,(IY+IND)

CÓDIGO DE MÁQUINA	COMANDO
FD7705	LD (IY+IND),A
FD7E05	LD A,(IY+IND)
FD8605	ADD A,(IY+IND)
FD8E05	ADC A,(IY+IND)
FD9605	SUB (IY+IND)
FD9E05	SBC A,(IY+IND)
FDA605	AND (IY+IND)
FDAE05	XOR (IY+IND)
FDB605	OR (IY+IND)
FDBE05	CP (IY+IND)
FDE1	POP IY
FDE3	EX (SP),IY
FDE5	PUSH IY
FDE9	JP (IY)
FDF9	LD SP,IY
FDCB0506	RLC (IY+IND)
FDCB050E	RRC (IY+IND)
FDCB0516	RL (IY+IND)
FDCB051E	RR (IY+IND)
FDCB0526	SLA (IY+IND)
FDCB052E	SRA (IY+IND)
FDCB053E	SRL (IY+IND)
FDCB0546	BIT 0,(IY+IND)
FDCB054E	BIT 1,(IY+IND)
FDCB0556	BIT 2,(IY+IND)
FDCB055E	BIT 3,(IY+IND)
FDCB0566	BIT 4,(IY+IND)
FDCB056E	BIT 5,(IY+IND)
FDCB0576	BIT 6,(IY+IND)
FDCB057E	BIT 7,(IY+IND)
FDCB0586	RES 0,(IY+IND)
FDCB058E	RES 1,(IY+IND)
FDCB0596	RES 2,(IY+IND)
FDCB059E	RES 3,(IY+IND)
FDCB05A6	RES 4,(IY+IND)
FDCB05AE	RES 5,(IY+IND)
FDCB05B6	RES 6,(IY+IND)
FDCB05BE	RES 7,(IY+IND)
FDCB05C6	SET 0,(IY+IND)
FDCB05CE	SET 1,(IY+IND)
FDCB05D6	SET 2,(IY+IND)

APÊNDICE D - *Lista Alfabética dos Códigos de Máquina*

COMANDO	CÓDIGO	COMANDO	CÓDIGO
ADC A,(HL)	8E	BIT 0,B	CB40
ADC A,(IX+IND)	DD8E05	BIT 0,C	CB41
ADC A,(IY+IND)	FD8E05	BIT 0,D	CB42
ADC A,A	8F	BIT 0,E	CB43
ADC A,B	88	BIT 0,H	CB44
ADC A,C	89	BIT 0,L	CB45
ADC A,D	8A	BIT 1,(HL)	CB4E
ADC A,E	8B	BIT 1,(IX+IND)	DDCB054E
ADC A,H	8C	BIT 1,(IY+IND)	FDCB054E
ADC A,L	8D	BIT 1,A	CB4F
ADC A,N	CE20	BIT 1,B	CB48
ADC HL,BC	ED4A	BIT 1,C	CB49
ADC HL,DE	ED5A	BIT 1,D	CB4A
ADC HL,HL	ED6A	BIT 1,E	CB4B
ADC HL,SP	ED7A	BIT 1,H	CB4C
ADD A,(HL)	86	BIT 1,L	CB4D
ADD A,(IX+IND)	DD8605	BIT 2,(HL)	CB56
ADD A,(IY+IND)	FD8605	BIT 2,(IX+IND)	DDCB0556
ADD A,A	87	BIT 2,(IY+IND)	FDCB0556
ADD A,B	80	BIT 2,A	CB57
ADD A,C	81	BIT 2,B	CB50
ADD A,D	82	BIT 2,C	CB51
ADD A,E	83	BIT 2,D	CB52
ADD A,H	84	BIT 2,E	CB53
ADD A,L	85	BIT 2,H	CB54
ADD A,N	C620	BIT 2,L	CB55
ADD HL,BC	09	BIT 3,(HL)	CB5E
ADD HL,DE	19	BIT 3,(IX+IND)	DDCB055E
ADD HL,HL	29	BIT 3,(IY+IND)	FDCB055E
ADD HL,SP	39	BIT 3,A	CB5F
ADD IX,BC	DD09	BIT 3,B	CB58
ADD IX,DE	DD19	BIT 3,C	CB59
ADD IX,IX	DD29	BIT 3,D	CB5A
ADD IX,SP	DD39	BIT 3,E	CB5B
ADD IY,BC	FD09	BIT 3,H	CB5C
ADD IY,DE	FD19	BIT 3,L	CB5D
ADD IY,IY	FD29	BIT 4,(HL)	CB66
ADD IY,SP	FD39	BIT 4,(IX+IND)	DDCB0566
AND (HL)	A6	BIT 4,(IY+IND)	FDCB0566
AND (IX+IND)	DDA605	BIT 4,A	CB67
AND (IY+IND)	FDA605	BIT 4,B	CB60
AND A	A7	BIT 4,C	CB61
AND B	A0	BIT 4,D	CB62
AND C	A1	BIT 4,E	CB63
AND D	A2	BIT 4,H	CB64
AND E	A3	BIT 4,L	CB65
AND H	A4	BIT 5,(HL)	CB6E
AND L	A5	BIT 5,(IX+IND)	DDCB056E
AND N	E620	BIT 5,(IY+IND)	FDCB056E
BIT 0,(HL)	CB46	BIT 5,A	CB6F
BIT 0,(IX+IND)	DDCB0546	BIT 5,B	CB68
BIT 0,(IY+IND)	FDCB0546	BIT 5,C	CB69
BIT 0,A	CB47	BIT 5,D	CB6A

COMANDO	CÓDIGO	COMANDO	CÓDIGO
BIT 5,E	CB6B	DEC A	3D
BIT 5,H	CB6C	DEC B	05
BIT 5,L	CB6D	DEC BC	0B
BIT 6,(HL)	CB76	DEC C	0D
BIT 6,(IX+IND)	DDCB0576	DEC D	15
BIT 6,(IY+IND)	FDCB0576	DEC DE	1B
BIT 6,A	CB77	DEC E	1D
BIT 6,B	CB70	DEC H	25
BIT 6,C	CB71	DEC HL	2B
BIT 6,D	CB72	DEC IX	DD2B
BIT 6,E	CB73	DEC IY	FD2B
BIT 6,H	CB74	DEC L	2D
BIT 6,L	CB75	DEC SP	3B
BIT 7,(HL)	CB7E	DI	F3
BIT 7,(IX+IND)	DDCB057E	DJNZ DIS	102E
BIT 7,(IY+IND)	FDCB057E	EI	FB
BIT 7,A	CB7F	EX (SP),HL	E3
BIT 7,B	CB78	EX (SP),IX	DDE3
BIT 7,C	CB79	EX (SP),IY	FDE3
BIT 7,D	CB7A	EX AF,AF`	08
BIT 7,E	CB7B	EX DE,HL	EB
BIT 7,H	CB7C	EXX	D9
BIT 7,L	CB7D	HALT	76
CALL C,NN	DC8405	IM 0	ED46
CALL M,NN	FC8405	IM 1	ED56
CALL NC,NN	D48405	IM 2	ED5E
CALL NN	CD8405	IN A,(C)	ED78
CALL NZ,NN	C48405	IN A,N	DB20
CALL P,NN	F48405	IN B,(C)	ED40
CALL PE,NN	EC8405	IN C,(C)	ED48
CALL PO,NN	E48405	IN D,(C)	ED50
CALL Z,NN	CC8405	IN E,(C)	ED58
CCF	3F	IN H,(C)	ED60
CP (HL)	BE	IN L,(C)	ED68
CP (IX+IND)	DDBE05	INC (HL)	34
CP (IY+IND)	FDBE05	INC (IX+IND)	DD3405
CP A	BF	INC (IY+IND)	FD3405
CP B	B8	INC A	3C
CP C	B9	INC B	04
CP D	BA	INC BC	03
CP E	BB	INC C	0C
CP H	BC	INC D	14
CP L	BD	INC DE	13
CP N	FE20	INC E	1C
CPD	EDA9	INC H	24
CPDR	EDB9	INC HL	23
CPI	EDA1	INC IX	DD23
CPIR	EDB1	INC IY	FD23
CPL	2F	INC L	2C
DAA	27	INC SP	33
DEC (HL)	35	IND	EDAA
DEC (IX+IND)	DD3505	INDR	EDBA
INI	EDA2	LD A,(DE)	1A
INIR	EDB2	LD A,(HL)	7E
JP (HL)	E9	LD A,(IX+IND)	DD7E05
JP (IX)	DDE9	LD A,(IY+IND)	FD7E05
JP (IY)	FDE9	LD A,(NN)	3A8405
JP C,NN	DA8405	LD A,A	7F
JP M,NN	FA8405	LD A,B	78
JP NC,NN	D28405	LD A,C	79
JP NN	C38405	LD A,D	7A
JP NZ,NN	C28405	LD A,E	7B

COMANDO	CÓDIGO	COMANDO	CÓDIGO
JP P,NN	F28405	LD A,H	7C
JP PE,NN	EA8405	LD A,I	ED57
JP PO,NN	E28405	LD A,L	7D
JP Z,NN	CA8405	LD A,N	3E20
JR C,DIS	382E	LD B,(HL)	46
JR DIS	182E	LD B,(IX+IND)	DD4605
JR NC,DIS	302E	LD B,(IY+IND)	FD4605
JR NZ,DIS	202E	LD B,A	47
JR Z,DIS	282E	LD B,B	40
LD (BC),A	02	LD B,C	41
LD (DE),A	12	LD B,D	42
LD (HL),A	77	LD B,E	43
LD (HL),B	70	LD B,H,NN	44
LD (HL),C	71	LD B,L	45
LD (HL),D	72	LD B,N	0620
LD (HL),E	73	LD BC,(NN)	ED4B8405
LD (HL),H	74	LD BC,NN	018405
LD (HL),N	3620	LD C,(HL)	4E
LD (IX+IND),A	DD7705	LD C,(IX+IND)	DD4E05
LD (IX+IND),B	DD7005	LD C,(IY+IND)	FD4E05
LD (IX+IND),C	DD7105	LD C,A	4F
LD (IX+IND),D	DD7205	LD C,B	48
LD (IX+IND),E	DD7305	LD C,C	49
LD (IX+IND),H	DD7405	LD C,D	4A
LD (IX+IND),L	DD7505	LD C,E	4B
LD (IX+IND),N	DD360520	LD C,H	4C
LD (IY+IND),A	FD7705	LD C,L	4D
LD (IY+IND),B	FD7005	LD C,N	0E20
LD (IY+IND),C	FD7105	LD D,(HL)	56
LD (IY+IND),D	FD7205	LD D,(IX+IND)	DD5605
LD (IY+IND),E	FD7305	LD D,(IY+IND)	FD5605
LD (IY+IND),H	FD7405	LD D,A	57
LD (IY+IND),L	FD7505	LD D,B	50
LD (IY+IND),N	FD360520	LD D,C	51
LD (NN),A	328405	LD D,D	52
LD (NN),BC	ED438405	LD D,E	53
LD (NN),DE	ED538405	LD D,H	54
LD (NN),HL	228405	LD D,L	55
LD (NN),IX	DD228405	LD D,N	1620
LD (NN),IY	FD228405	LD DE,(NN)	ED5B8405
LD (NN),SP	ED738405	LD DE,NN	118405
LD A,(BC)	0A	LD D,(HL)	5E
LD E,(IX+IND)	DDE505	OR A	B7
LD E,(IY+IND)	FDE505	OR B	B0
LD E,A	5F	OR C	B1
LD E,B	58	OR D	B2
LD E,C	59	OR E	B3
LD E,D	5A	OR H	B4
LD E,E	5B	OR L	B5
LD E,H	5C	OR N	F620
LD E,L	5D	OTDR	EDBB
LD E,N	1E20	OTIR	EDB3
LD H,(HL)	66	OUT (C),A	ED79
LD H,(IX+IND)	DD6605	OUT (C),B	ED41
LD H,(IY+IND)	FD6605	OUT (C),C	ED49
LD H,A	67	OUT (C),D	ED51
LD H,B	60	OUT (C),E	ED59
LD H,C	61	OUT (C),H	ED61
LD H,D	62	OUT (C),L	ED69
LD H,E	63	OUT N,A	DE20
LD H,H	64	OUTD	EDA8
LD H,L	65	OUTI	EDA3

COMANDO	CÓDIGO	COMANDO	CÓDIGO
LD H,N	2620	POP AF	F1
LD HL,(NN)	2A8405	POP BC	C1
LD HL,NN	218405	POP DE	D1
LD I,A	ED47	POP HL	E1
LD IX,(NN)	DD2A8405	POP IX	DDE1
LD IX,NN	DD218405	POP IY	FDE1
LD IY,(NN)	FD2A8405	PUSF AF	F5
LD IY,NN	FD218405	PUSH BC	C5
LD L,(HL)	6E	PUSH DE	D5
LD L,(IX+IND)	DD6E05	PUSH HL	E5
LD L,(IY+IND)	FD6E05	PUSH IX	DDE5
LD L,A	6F	PUSH IY	FDE5
LD L,B	68	RES 0,(HL)	CB86
LD L,C	69	RES 0,(IX+IND)	DDCB0586
LD L,D	6A	RES 0,(IY+IND)	FDCB0586
LD L,E	6B	RES 0,A	CB87
LD L,H	6C	RES 0,B	CB80
LD L,L	6D	RES 0,C	CB81
LD L,N	2E20	RES 0,D	CB82
LD SP,(NN)	ED7B8405	RES 0,E	CB83
LD SP,HL	F9	RES 0,H	CB84
LD SP,IX	DDF9	RES 0,L	CB85
LD SP,IY	FDF9	RES 1,(HL)	CB8E
LD SP,NN	318405	RES 1,(IX+IND)	DDCB058E
LDD	EDA8	RES 1,(IY+IND)	FDCB058E
LDDR	EDB8	RES 1,A	CB8F
LDI	EDA0	RES 1,B	CB88
LDIR	EDB0	RES 1,C	CB89
NEG	ED44	RES 1,D	CB8A
NOP	00	RES 1,E	CB8B
OR (HL)	B6	RES 1,H	CB8C
OR (IX+IND)	DDB605	RES 1,L	CB8D
OR (IY+IND)	FDB605	RES 2,(HL)	CB96
RES 2,(IY+IND)	FDCB0596	RES 5,C	CBA9
RES 0,A	CB87	RES 5,D	CBAA
RES 0,B	CB80	RES 5,E	CBAB
RES 0,C	CB81	RES 5,H	CBAC
RES 0,D	CB82	RES 5,L	CBAD
RES 0,E	CB83	RES 6,(HL)	CBB6
RES 0,H	CB84	RES 6,(IX+IND)	DDCB05B6
RES 0,L	CB85	RES 6,(IY+IND)	FDCB05B6
RES 1,(HL)	CB8E	RES 6,A	CBB7
RES 1,(IX+IND)	DDCB058E	RES 6,B	CBB0
RES 1,(IY+IND)	FDCB058E	RES 6,C	CBB1
RES 1,A	CB8F	RES 6,D	CBB2
RES 1,B	CB88	RES 6,E	CBB3
RES 1,C	CB89	RES 6,H	CBB4
RES 1,D	CB8A	RES 6,L	CBB5
RES 1,E	CB8B	RES 7,(HL)	CBBE
RES 1,H	CB8C	RES 7,(IX+IND)	DDCB05BE
RES 1,L	CB8D	RES 7,(IY+IND)	FDCB05BE
RES 2,(HL)	CB96	RES 7,A	CBBF
RES 2,(IX+IND)	DDCB0596	RES 7,B	CBB8
RES 2,(IY+IND)	FDCB0596	RES 7,C	CBB9
RES 2,A	CB97	RES 7,D	CBBA
RES 2,B	CB90	RES 7,E	CBBB
RES 2,C	CB91	RES 7,H	CBBC
RES 2,D	CB92	RES 7,L	CBBD
RES 2,E	CB93	RET	C9
RES 2,H	CB94	RET C	D8
RES 2,L	CB95	RET M	F8
RES 3,(HL)	CB9E	RET NC	D0

COMANDO	CÓDIGO	COMANDO	CÓDIGO
RES 3,(IX+IND)	DDCB059E	RET NZ	C0
RES 3,(IY+IND)	FDCB059E	RET P	F0
RES 3,A	CB9F	RET PE	E8
RES 3,B	CB98	RET PO	E0
RES 3,C	CB99	RET Z	C8
RES 3,D	CB9A	RETI	ED4D
RES 3,E	CB9B	RETN	ED45
RES 3,H	CB9C	RL (HL)	CB16
RES 3,L	CB9D	RL (IX+IND)	DDCB0516
RES 4,(HL)	CBA6	RL (IY+IND)	FDCB0516
RES 4,(IX+IND)	DDCB05A6	RL A	CB17
RES 4,A	CBA7	RL B	CB10
RES 4,B	CBA0	RL C	CB11
RES 4,C	CBA1	RL D	CB12
RES 4,D	CBA2	RL E	CB13
RES 4,E	CBA3	RL H	CB14
RES 4,H	CBA4	RL L	CB15
RES 4,L	CBA5	RLA	17
RES 5,(HL)	CBAE	RLC (HL)	CB06
RES 5,(IX+IND)	DDCB05AE	RLC (IX+IND)	DDCB0506
RES 5,(IY+IND)	FDCB05AE	RLC (IY+IND)	FDCB0506
RES 5,A	CBAF	RLC A	CB07
RES 5,B	CBA8	RLC B	CB00
RLC C	CB01	SCF	37
RLC D	CB02	SET 0,(HL)	CBC6
RLC E	CB03	SET 0,(IX+IND)	DDCB05C6
RLC H	CB04	SET 0,(IY+IND)	FDCB05C6
RLC L	CB05	SET 0,A	CBC7
RLCA	07	SET 0,B	CBC0
RLD	ED6F	SET 0,C	CBC1
RR (HL)	CB1E	SET 0,D	CBC2
RR (IX+IND)	DDCB051E	SET 0,E	CBC3
RR (IY+IND)	FDCB051E	SET 0,H	CBC4
RR A	CB1F	SET 0,L	CBC5
RR B	CB18	SET 1,(HL)	CBCE
RR C	CB19	SET 1,(IX+IND)	DDCB05CE
RR D	CB1A	SET 1,(IY+IND)	FDCB05CE
RR E	CB1B	SET 1,A	CBCF
RR H	CB1C	SET 1,B	CBC8
RR L	CB1D	SET 1,C	CBC9
RRA	1F	SET 1,D	CBCA
RRC (HL)	CB0E	SET 1,E	CBCB
RRC (IX+IND)	DDCB050E	SET 1,H	CBCC
RRC (IY+IND)	FDCB050E	SET 1,L	CBCD
RRC A	CB0F	SET 2,(HL)	CBD6
RRC B	CB08	SET 2,(IX+IND)	DDCB05D6
RRC C	CB09	SET 2,(IY+IND)	FDCB05D6
RRC D	CB0A	SET 2,A	CBD7
RRC E	CB0B	SET 2,B	CBD0
RRC H	CB0C	SET 2,C	CBD1
RRC L	CB0D	SET 2,D	CBD2
RRCA	0F	SET 2,E	CBD3
RRD	ED67	SET 2,H	CBD4
RST 0	C7	SET 2,L	CBD5
RST 10H	D7	SET 3,B	CBD8
RST 18H	DF	SET 3,(HL)	CBDE
RST 20H	E7	SET 3,(IX+IND)	DDCB05DE
RST 28H	EF	SET 3,(IY+IND)	FDCB05DE
RST 30H	F7	SET 3,A	CBDF
RST 38H	FF	SET 3,C	CBD9
RST 8	CF	SET 3,D	CBDA
SBC A,(HL)	9E	SET 3,E	CBDB

COMANDO	CÓDIGO
SBC A,(IX+IND)	DD9E05
SBC A,(IY+IND)	FD9E05
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
SBC A,N	DE20
SBC HL,BC	ED42
SBC HL,DE	ED52
SBC HL,HL	ED62
SBC HL,SP	ED72
SET 5,(IY+IND)	FDCB05EE
SET 5,A	CBEF
SET 5,B	CBE8
SET 5,C	CBE9
SET 5,D	CBEA
SET 5,E	CBEB
SET 5,H	CBEC
SET 5,L	CBED
SET 6,(HL)	CBF6
SET 6,(IX+IND)	DDCB05F6
SET 6,(IY+IND)	FDCB05F6
SET 6,A	CBF5
SET 6,B	CBF0
SET 6,C	CBF1
SET 6,D	CBF2
SET 6,E	CBF3
SET 6,H	CBF4
SET 6,L	CBF5
SET 7,(HL)	CBFE
SET 7,(IX+IND)	DDCB05FE
SET 7,(IY+IND)	FDCB05FE
SET 7,A	CBFF
SET 7,B	CBF8
SET 7,C	CBF9
SET 7,D	CBFA
SET 7,E	CBFB
SET 7,H	CBFC
SET 7,L	CBFD
SLA (HL)	CB26
SLA (IX+IND)	DDCB0526
SLA (IY+IND)	FDCB0526
SLA A	CB27
SLA B	CB20
SLA C	CB21
SLA D	CB22
SLA E	CB23
SLA H	CB24
SLA L	CB25
SRA (HL)	CB2E
SRA (IX+IND)	DDCB052E
SRA (IY+IND)	FDCB052E
SRA A	CB2F
SRA B	CB28
SRA C	CB29
SRA D	CB2A
SRA E	CB2B
SRA H	CB2C
SRA L	CB2D

COMANDO	CÓDIGO
SET 3,H	CBDC
SET 3,L	CBDD
SET 4,(HL)	CBE6
SET 4,(IX+IND)	DDCB05E6
SET 4,(IY+IND)	FDCB05E6
SET 4,A	CBE7
SET 4,B	CBE0
SET 4,C	CBE1
SET 4,D	CBE2
SET 4,E	CBE3
SET 4,H	CBE4
SET 4,L	CBE5
SET 5,(HL)	CBEE
SET 5,(IX+IND)	DDCB05EE
SRL C	CB39
SRL D	CB3A
SRL E	CB3B
SRL H	CB3C
SRL L	CB3D
SUB (HL)	96
SUB (IX+IND)	DD9605
SUB (IY+IND)	FD9605
SUB A	97
SUB B	90
SUB C	91
SUB D	92
SUB E	93
SUB H	94
SUB L	95
SUB N	D620
XOR (HL)	AE
XOR (IX+IND)	DDAE05
XOR (IY+IND)	FDAE05
XOR A	AF
XOR B	A8
XOR C	A9
XOR D	AA
XOR E	AB
XOR H	AC
XOR L	AD
XOR N	EE20

COMANDO	CÓDIGO
SRL (HL)	CB3E
SRL (IX+IND)	DDCB053E
SRL (IY+IND)	FDCB053E
SRL A	CB3F
SRL B	CB38

APÊNDICE E – *Palavras Reservadas*

A lista abaixo é o conjunto das palavras reservadas que são utilizadas como códigos de operação e, portanto, NÃO PODEM SER USADAS COMO RÓTULOS.

LD	AND	RLA	DJNZ
PUSH	OR	RRCA	CALL
POP	XOR	RRA	RET
EX	CP	RLC	RETI
EXX	INC	RL	RETN
LDI	DEC	RRC	RST
LDIR	DAA	RR	IN
LDD	CPL	SLA	INI
LDDR	NEG	SRA	INIR
CPI	CCF	SRL	IND
CPIR	SCF	RLD	INDR
CPD	NOP	RRD	OUT
CPDR	HALT	BIT	OUTI
ADD	DI	SET	OTIR
ADC	EI	RES	OUTD
SUB	IM	JP	OTDR
SBC	RLCA	JR	

Glossário

alfanumérico – Característica dos caracteres representativos de letras e números.

Assembler – Tipo de linguagem de computador, de baixo nível, que trabalha com instruções em códigos mnemônicos, que são traduzidos para os correspondentes códigos de máquina.

BASIC – Uma linguagem de computador, de alto nível, semelhante à língua inglesa, muito utilizada em microcomputadores, devido à facilidade de aprendizado, dentre outras características.

binário – Um sistema numérico que utiliza a base 2 e possui, apenas, dois dígitos.

bit – A menor unidade dentro de uma posição de memória. (O termo provém da contração de binary digit, isto é, dígito binário.)

bloco – Um grupo de bytes ou de posições de memória.

buffer – Uma área de memória, reservada pelo programa, para o armazenamento de dados.

byte – Um elemento de informação (composto de 8 bits). Também é mencionado como uma posição de memória.

CALL – Desviar para outra parte do programa, tendo um método de poder retornar a execução para a instrução subseqüente ao CALL.

caracter – Qualquer das letras, ou números, ou símbolos especiais do teclado. Cada caracter é representado por um byte.

carregar – Armazenar um valor em algum lugar ou em alguma coisa.

código de operação – conjunto de bytes que o computador reconhece como o comando para alguma ação predeterminada.

compilador – Um programa usado para traduzir a informação de uma linguagem para código de máquina.

decimal – De base 10.

deslocar – Mover o conteúdo de um registrador.

desvio – Passar a execução para um outro ponto do programa, saltando por sobre outras instruções. O mesmo que salto.

endereço – Localização ou posição de memória, que pode armazenar um valor ou uma instrução.

flag – Cada um dos bits do registrador F, usados como assinaladores de várias condições. (O termo inglês foi incorporado ao jargão de processamento de dados no Brasil.)

hexadecimal – De base 16.

incremento – Aumento de uma unidade.

linguagem de máquina – A linguagem de programação de mais baixo nível, isto é, a mais próxima do formato inteligível pelo computador.

localização de memória – O mesmo que posição, endereço.

operando – O registrador, par de registradores, ou localização de memória, sobre a qual a operação deve ser executada.

paridade – Qualidade de ser igual.

pilha – A área de memória disponível, localizada após a última instrução do programa.

POP – Instrução que extrai dados da pilha.

posição – O mesmo que localização, endereço.

potência – Número de vezes que um número deve ser multiplicado por ele mesmo.

pseudocódigo – Um comando operacional especial para a linguagem Assembler.

PUSH – Instrução que executa o armazenamento de dados na pilha, tipicamente utilizada para salvar o conteúdo de pares de registradores.

RAM – Random Access Memory – Memória de acesso aleatório, ou direto.

registrador – Circuito especial de 8 bits, usado para o trabalho da informação durante a execução de programas.

rótulo – Um grupo de caracteres alfanuméricos, iniciado por uma letra, que associa um nome a uma rotina ou localização dentro do programa.

salto – O mesmo que desvio.

zerar – fazer igual a zero.

Índice Analítico

A
Abreviaturas, 35
Acumulador, 41
ADC (instrução), 71, 85
ADD (especial), 71
ADD (instruções), 69, 84, 85
Adição, 69
AND (instrução), 72
AND Lógico (operação), 72

B
BASIC, 11, 23, 29
BCD (código), 79
Bit, 38
Bit (instrução), 96
Buffer, 35, 114
Byte, 38

C
CALL (instrução), 107
CCF (instrução), 80
COBOL, 11
Código de máquina, 30
Códigos das teclas, 164
Códigos de caracteres, 164
Códigos de letras minúsculas, 165
Códigos de operação, 31
Códigos hexa de controle, 164
Comentário, 31
COMPARE (instruções), 64, 75
Compiladores, 29, 156
Contador de instruções, 66
Conversão binário para decimal, 14, 15
Conversão de Assembler para linguagem de máquina, 29
Conversão de BASIC para linguagem de máquina, 156
Conversão decimal para hexa, 17
Conversão hexa para decimal, 17
Cópia de programas, 23
CPD (instrução), 66
CPDR (instrução), 66
CPI (instrução), 64
CPIR (instrução), 66
CPL (instrução), 80

D
DEC (instrução), 76, 86
Decimal codificado em binário, 79
DEFB (pseudocódigo), 113
DEFL (pseudocódigo), 113
DEFM (pseudocódigo), 114
DEFS (pseudocódigo), 114
DEFW (pseudocódigo), 113
Deslocamento da instrução JR, 181
Divisão, 69
Divisão binária, 16
DJNZ (instrução), 104
Driver, 121

E
END (pseudocódigo), 112
Endereços, 25
EQU (pseudocódigo), 113
EX (instrução), 58

F
Flags, 63, 64
Flags, condições, 46, 63, 64, 103
FORTRAN, 3
Funções matemáticas, 69, 84

H
Hexadecimal, 13, 17

I
INC (instrução), 75, 86
INPUT (grupo de instruções), 107

Interpretadores, 29
Interrupções, 43, 66

J
JP (instrução), 101
JR (instrução), 102
Jump (instrução), 101

L
LD (instrução), 38
LDD (instrução), 62
LDI (instrução), 60
LIFO, 34
Limpeza da tela, 118, 120
Língua inglesa, 11
Linguagem Assembler, 11, 23
Linguagem de computadores, 11
Linguagem de máquina, 11, 23

M
Mascaramento, 73
Memória, 34
Memória que só pode ser lida (ROM), 25
Montador/editor Assembler, 23, 29
Multiplicação, 69

N
NEG (instrução), 80
NOP (instrução), 81
Números binários, 13, 14
Números decimais, 13
Números hexadecimais, 13, 17

O
OR (instrução), 73
ORG (pseudocódigo), 111
OU exclusivo (instrução), 74
OUTPUT (grupo de instruções), 107

P
Par de registradores, 25
Par de registradores AF, 62
Par de registradores HL, 48
Par de registradores IX, 26, 47
Par de registradores IY, 11, 42
Par de registradores SP, 46
Pesquisa para comparação, 63
Pesquisas, 63
Pilha, 34, 35
PL/I, 11
Ponteiro da pilha, 25, 52
POP (instrução), 53
Posições de memória, 25
Programa de cópia de memória, 132
Programa de lista de endereçamento, 126
Programa para cópia de fitas, 143
Programa T-BUG, 23, 30
Programação, 11
Pseudocódigos, 30, 111
PUSH (instrução), 53

Q
Quick Printer II, 120

R
Recuperação de dados, 55
"Refresh" da memória, 43
Registrador A, 63
Registrador F, 46, 62, 80
Registradores, 25, 26
RETURN (instrução), 108
RL (instrução), 91
RLA (instrução), 90
RLC (instrução), 91
RLCA (instrução), 90
ROM (Read-Only Memory), 25
Rotações, 89-93
Rótulos, 30
RR (instrução), 92
RRA (instrução), 90
RRC (instrução), 92
RRCA (instrução), 90

S
SBC (instrução), 72, 85
SCF (instrução), 80
SET (instrução), 97
SHIFT (instruções), 92
Sistemas numéricos, 13
SLA (instrução), 92
SRA (instrução), 92
SUB (instrução), 72
Sub-rotinas, 24

T
Tabela de potências de 10, 61
Tela, preenchimento com um carácter, 122
Tempo de execução, 23
Transferência de dados, 124
Transferências em bloco, 60

U
Unidade Central de Processamento (UCP), 23

V
Vetor de interrupção, 43

X
XOR (instrução), 74

Programação em ASSEMBLER e Linguagem de Máquina

Iniciantes que procuram uma introdução simples e clara às linguagens de máquina e montagem encontrarão neste livro uma orientação que, a partir dos "comos" e dos "porquês", os levará à prática real da programação.

O texto é bastante objetivo e fácil de acompanhar, não exigindo experiência prévia ou habilidades específicas em matemática. Ao final dos capítulos um breve questionário ajuda a verificar a compreensão de cada etapa, antes que seja iniciado o estudo da seguinte.

Sistemas numéricos e registradores, códigos operacionais e instruções PUSH e POP, Grupo de Controle de UCP e Grupo de Instruções de Teste são alguns dos tópicos que antecedem o capítulo sobre os "Programas de Exemplo", com que o leitor já estará então apto a trabalhar, criando, inclusive, seus próprios programas.

ISBN 85-7001-237-3
(Edição original: ISBN 0-8306-1389-7 Tab Books Inc., USA.)