```
0D82:0100 B402          MOV     AH,02
0D82:0102 B241          MOV     DL,41
0D82:0104 CD21          INT     21
0D82:0106 CD20          INT     20
0D82:0108 69            DB      69
0D82:0109 64            DB      64
0D82:010A 61            DB      61
0D82:010B 64            DB      64
0D82:010C 65            DB      65
0D82:010D 206573        AND     [DI+73],AH
0D82:0110 7065          JO      0177
0D82:0112 63            DB      63
0D82:0113 69            DB      69
0D82:0114 66            DB      66
0D82:0115 69            DB      69
0D82:0116 63            DB      63
0D82:0117 61            DB      61
```

Apêndice

```
0D82:0100 B402          MOV     AH,02
0D82:0102 B241          MOV     DL,41
0D82:0104 CD21          INT     21
0D82:0106 CD20          INT     20
0D82:0108 69            DB      69
```

Apêndice A

REFERÊNCIA OPERACIONAL

Este apêndice apresenta a listagem das instruções 8086/8088 em modo *Assembly* e em modo *Opcode* (código em linguagem de máquina), além de outras informações importantes.

O microprocessador 8086/8088 possui um conjunto de instruções que não segue um tamanho homogêneo. Dependendo da instrução em uso ou de seu formato operacional, a quantidade de *bits* usados será diferente, o que leva à necessidade de o programador conhecer a estrutura interna de formação das instruções e de seus operandos. Uma instrução 8086/8088 pode ocupar de 1 até 6 *bytes*, como pode ser constatado na coluna **Byte** da tabela **Instruções Assembly e Opcodes**. A característica de formatação heterogênea da composição de uma instrução 8086/8088 e de seus operandos pode parecer complexa ou de difícil administração, mas em contrapartida, fornece ao programador uma grande possibilidade de escrever códigos de programas mais versáteis. Assim sendo, uma instrução pode ser escrita em uma linha de acordo com a seguinte estrutura:

1º. Byte							
7	6	5	4	3	2	1	0
OPCODE						D	W

2º. Byte							
7	6	5	4	3	2	1	0
MOD		REG			R/M		

3o. Byte							
7	6	5	4	3	2	1	0
DESLOCAMENTO MENOS SIGNIFICATIVO							

4º. Byte							
7	6	5	4	3	2	1	0
DESLOCAMERNTO MAIS SIGNIFICATIVO							

5º. Byte							
7	6	5	4	3	2	1	0
DADO MENOS SIGNIFICATIVO							

6º. Byte							
7	6	5	4	3	2	1	0
DADO MAIS SIGNIFICATIVO							

A formatação de uma instrução pode ocorrer de algumas formas. Uma instrução pode ser composta pelos *bytes* 1, 2, 3, 4, 5 e 6, como pode ser composta apenas pelo *byte* 1 ou pelos *bytes* 1 e 2, bem como pode ser composta com os *bytes* 1, 5 e 6, ou qualquer outra forma, desde que sempre esteja definido inicialmente o *byte* 1.

Para facilitar o entendimento do exposto anteriormente, seguem as nomenclaturas utilizadas na representação da estrutura das instruções da linguagem Assembly 8086/8088. As instruções usadas para a programação da linguagem Assembly podem usar como indicado anteriormente de um até seis bytes de memória, que podem estar dispostos segundo a seguinte estrutura:

- Instrução – ocupa de um a dois bytes;
- dado – ocupa de de um a dois bytes;
- endereço – ocupa um, dois ou quatro bytes.

O *byte* 1 caracteriza-se por ser a forma mais simples de representação de uma instrução. Possui os bits 7, 6, 5, 4, 3 e 2 para representar o OPCODE (código de operação) em si, sucedidos dos bits 1 e 0, que possuem como significado o seguinte:

- O *bit* 1 do *byte* 1 é representado pela indicação D, a qual define o sentido de direção que uma operação de movimentação de dados terá sobre um registrador. Se D=0, indica que o conteúdo do campo REG (*byte* 2) será interpretado como fonte; se D=1, indica que o conteúdo do campo REG (*byte* 2) será interpretado como destino.
- O *bit* 0 do *byte* 1 é representado pela indicação W, a qual define o modo como a operação será realizada de acordo com o dado definido, podendo esse dado ser um *word* ou um *byte*. Se W=0, operação realizada como *byte*; se W=1, operação realizada como *word*.

O *byte* 2, quando utilizado, efetua o modo de endereçamento e acesso à memória por meio dos registradores. Possui sua estrutura segmentada em três partes, sendo MOD (bits 7 e 6), REG (bits 5, 4 e 3) e R/M (bits 2, 1 e 0), que possuem o seguinte significado:

- A parte MOD é usada para indicar o modo de endereçamento e definição dos operandos em relação ao uso de registradores ou memória, ou seja, informa de onde é obtido o operando. Olhe a tabela **Modo (MOD)**.

Modo (MOD)	
Código	**Descrição**
00	A operação será em modo memória sem deslocamento, exceto quando o campo R/M do byte 2 for igual a 110, pois neste caso o deslocamento será de 16 bits.
01	A operação será em modo memória com deslocamento de 8 bits.
10	A operação será em modo memória com deslocamento de 16 bits.
11	A operação será em modo registrador. Neste caso, consultar a tabela Registrador/Memória (R/M).

- A parte REG indica o registrador que será usado na operação do *byte* 2, a partir do valor setado no *bit* 0 do *byte* 1. Olhe a tabela **Registrador (REG)**.

Registrador (REG)			
Código	**W = 0**	**W = 1**	**Segmento**
000	AL	AX	ES
001	CL	CX	CS
010	DL	DX	SS
011	BL	BX	DS
100	AH	SP	-
101	CH	BP	-
110	DH	SI	-
111	BH	DI	-

- A parte R/M é usada para indicar as operações de cálculo de endereços efetivos sobre a memória e sobre o registrador. Olhe a tabela **Registrador/Memória (R/M)**.

Registrador/Memória (R/M)					
MOD = 11			Cálculo do Endereço Efetivo		
R/M	W = 0	W = 1	MOD = 00	MOD = 01	MOD = 10
000	AL	AX	[BX + SI]	[BX + SI] + DESLB	[BX + SI] + DESLW
001	CL	CX	[BX + DI]	[BX + DI] + DESLB	[BX + DI] + DESLW
010	DL	DX	[BP + SI]	[BP + SI] + DESLB	[BP + SI] + DESLW
011	BL	BX	[BP + DI]	[BP + DI] + DESLB	[BP + DI] + DESLW
100	AH	SP	[SI]	[SI] + DESLB	[SI] + DESLW
101	CH	BP	[DI]	[DI] + DESLB	[DI] + DESLW
110	DH	SI	DESLW	[BP] + DESLB	[BP] + DESLW
111	BH	DI	[BX]	[BX] + DESLBb	[BX] + DESLW

- As indicações DESLB e DESLW se relacionam, respectivamente, à ação de deslocamento de *byte* ou *word* em memória.
- A estrutura do *byte* 2 é utilizada para a definição do conteúdo **ModR/M**. A tabela **Estrutura Padrão do Byte** ModR/M mostra a estrutura de formação de um byte ModR/M (byte 2). Observe que o byte ModR/M é dividido em três grupos operacionais.

Estrutura Padrão do Byte ModR/M								
ModR/M Byte 2	MOD		REG			R/M		
	7	6	5	4	3	2	1	0
	3º.		2º.			1º.		

- O campo **MOD** (*bits* 7 e 6) definido como terceiro grupo deve ser combinado com o campo **R/M** (*bits* 2, 1 e 0) definido como primeiro grupo no sentido de possibilitar a formação de 32 valores, sendo oito valores para os registradores de 8 *bits* (**AL**, **CL**, **DL**, **BL**, **AH**, **CH**, **DH** e **BH**) ou 16 *bits* (**AX**, **CX**, **DX**, **BX**, **SP**, **BP**, **SI** e **DI**), mais 24 valores para os modos de endereçamento representados como *byte* **ModR/M**.
- Os *bits* de **0** até **2** indicam a definição de registro **(R)** ou memória **(M)** para o campo **R/M**, os *bits* de **3** até **5** indicam a definição de registro ou de segmento de registro para o campo **Reg** e os *bits* de **6** até **7** indicam para o campo **Mod** como o campo **R/M** deve ser interpretado.
- A partir das tabelas **Modo (MOD)**, **Registrador (REG)**, **Registrador/Memória (R/M)** e **Estrutura Padrão do Byte ModR/M** fica definida como endereço efetivo a junção do primeiro e do terceiro grupos (MOD + R/M), o que possibilita obter os seguintes endereços:

DS:[BX+SI] 00xxx000

DS:[BX+DI] 00xxx001

SS:[BP+SI] 00xxx010

SS:[BP+DI] 00xxx011

DS:[SI] 00xxx100

DS:[DI] 00xxx101

DS:DESLW 00xxx110

DS:[BX] 00xxx111

DS:[BX+SI+DESLB] 01xxx000

DS:[BX+DI+DESLB] 01xxx001

SS:[BP+SI+DESLB]	01xxx010
SS:[BP+DI+DESLB]	01xxx011
DS:[SI+DESLB]	01xxx100
DS:[DI+DESLB]	01xxx101
SS:[BP+DESLB]	01xxx110
DS:[BX+DESLB]	01xxx111
DS:[BX+SI+DESLW]	10xxx000
DS:[BX+DI+DESLW]	10xxx001
SS:[BP+SI+DESLW]	10xxx010
SS:[BP+DI+DESLW]	10xxx011
DS:[SI+DESLW]	10xxx100
DS:[DI+DESLW]	10xxx101
SS:[BP+DESLW]	10xxx110
DS:[BX+DESLW]	10xxx111
AL | AX	11xxx000
CL | CX	11xxx001
DL | DX	11xxx010
BL | BX	11xxx011
AH | SP	11xxx100
CH | BP	11xxx101
DH | SI	11xxx110
BH | DI	11xxx111

- A partir das tabelas **Modo (MOD)**, **Registrador (REG)**, **Registrador/Memória (R/M)** e **Estrutura Padrão do Byte ModR/M** ficam definidos para o segundo grupo (REG) os seguintes registradores:

AL | AX	xx000xxx
CL | CX	xx001xxx
DL | DX	xx010xxx
BL | BX	xx011xxx
AH | SP	xx100xxx
CH | BP	xx101xxx
DH | SI	xx110xxx
BH | DI	xx111xxx

- A partir das tabelas **Modo (MOD)**, **Registrador (REG)**, **Registrador/Memória (R/M)** e **Estrutura Padrão do Byte ModR/M** ficam definidos para o segundo grupo (REG) os seguintes registradores de segmento:

ES	xx000xxx
CS	xx001xxx
SS	xx010xxx
DS	xx011xxx

Os *bytes* 3 e 4, quando utilizados, permitem definir os valores de deslocamento em memória. Esses *bytes* são utilizados apenas em uma parte do conjunto de instruções.

Os *bytes* 5 e 6, quando utilizados, permitem definir os valores dos dados a serem operacionalizados com a instrução em uso. Esses *bytes* são também utilizados apenas em uma parte do conjunto de instruções.

A tabela seguinte apresenta a estrutura geral do modo de operação em relação aos registradores e à memória.

Modo de Operação											
#	Mod	Reg/Mem	Endereço Efetivo	Reg/Opcodes em Valores Hexadecimais							
				0	1	2	3	4	5	6	7
				000	001	010	011	100	101	110	111
1	00	000	[BX+SI]	00	08	10	10	20	28	30	38
2	00	001	[BX+DI]	01	09	11	19	21	29	31	39
3	00	010	[BP+SI]	02	0A	12	1A	22	2A	32	3A
4	00	011	[BP+DI]	03	0B	13	1B	23	2B	33	3B
5	00	100	[SI]	04	0C	14	1C	24	2C	34	3C
6	00	101	[DI]	05	0D	15	1D	25	2D	35	3D
7	00	110	DESLW	06	0E	16	1E	26	2E	36	3E
8	00	111	[BX]	07	0F	17	1F	27	2F	37	3F
9	01	000	[BX+SI]+DESLB	40	48	50	58	60	68	70	78
10	01	001	[BX+DI]+DESLB	41	49	51	59	61	69	71	79
11	01	010	[BP+SI]+DESLB	42	4A	52	5A	62	6A	72	7A
12	01	011	[BP+DI]+DESLB	43	4B	53	5B	63	6B	73	7B
13	01	100	[SI]+DESLB	44	4C	54	5C	64	6C	74	7C
14	01	101	[DI]+DESLB	45	4D	55	5D	65	6D	75	7D
15	01	110	[BP]+DESLB	46	4E	56	5E	66	6E	76	7E
16	01	111	[BX]+DESLB	47	4F	57	5F	67	6F	77	7F
17	10	000	[BX+SI]+DESLW	80	88	90	98	A0	A8	B0	B8
18	10	001	[BX+DI]+DESLW	81	89	91	99	A1	A9	B1	B9
19	10	010	[BP+SI]+DESLW	82	8A	92	9A	A2	AA	B2	BA
20	10	011	[BP+DI]+DESLW	82	8B	93	9B	A3	AB	B3	BB
21	10	100	[SI]+DESLW	84	8C	94	9C	A4	AC	B4	BC
22	10	101	[DI]+DESLW	85	8D	95	9D	A5	AD	B5	BD
23	10	110	[BP]+DESLW	86	8E	96	9E	A6	AE	B6	BE
24	10	111	[BX]+DESLW	87	8F	97	9F	A7	AF	B7	BF
25	11	000	AL / AX	C0	C8	D0	D8	E0	E8	F0	F8
26	11	001	CL / CX	C1	C9	D1	D9	E1	E9	F1	F9
27	11	010	DL / DX	C2	CA	D2	DA	E2	EA	F2	FA
28	11	011	BL / BX	C3	CB	D3	DB	E3	EB	F3	FB
29	11	100	AH / SP	C4	CC	D4	DC	E4	EC	F4	FC
30	11	101	CH / BP	C5	CD	D5	DD	E5	ED	F5	FD
31	11	110	DH / SI	C6	CE	D6	DE	E6	EE	F6	FE
32	11	111	BH / DI	C7	CF	D7	DF	E7	EF	F7	FF

No sentido de entender a distribuição da tabela **Modo de Operação**, considere a instrução **DIV BX**, levando-se em consideração que estejam os registradores **AX** com o valor do dividendo e **BX** com o valor do divisor. Como orientado no capítulo três, o *Opcode* para essa operação é **F7F3**, sendo **F7h** o código em linguagem de máquina para a instrução **DIV BX**, como pode ser verificado na tabela **Instruções Assembly e Opcodes** (**DIV BX** = **F7 / 6**) e **F3h** é a indicação de que a operação será efetuada com o registrador **BX**, como pode ser comprovado na tabela **Modo de Operação**.

Tome por base o valor **F3h**, convertendo-o em seu equivalente binário (**11110011b**). Em seguida separe os termos **11-110-011** para identificar assim **Mod-Reg/Opcode-Reg/Mod**. Junte os termos **Mod** e **Reg/Mod** e ter-se-á o valor **11-011**

(linha 28 da tabela), localize na tabela a coluna referente ao código **Reg/Opcode** com valor **110** (coluna 6) e veja que o valor apresentado em hexadecimal é **F3** referente às operações sobre os registradores **BX** e **BL**.

Neste sentido, a tabela de **Notação Utilizada para Assembly/Opcodes** descreve os símbolos e seus significados que são utilizados nas tabelas **Modo de Operação** e **Instruções Assembly e Opcodes**.

Notação Utilizada para Assembly/Opcodes	
Símbolo	**Significado**
DRB	Indica deslocamento relativo para a próxima instrução: JMP, CALL, entre outras sinalizadas como segmento de *bytes*.
DRD	Indica deslocamento relativo para a próxima instrução: JMP, CALL, entre outras sinalizadas como segmento de *double words*.
DRW	Indica deslocamento relativo para a próxima instrução: JMP, CALL, entre outras sinalizadas como segmento de *words*.
MB	Operação realizada em endereço de memória com tamanho de um byte de DS:SI ou ES:DI (usado somente em instruções com *strings*).
MW	Operação realizada em endereço de memória com tamanho de um word de DS:SI ou ES:DI (usado somente em instruções com *strings*).
PW	Operação realizada com ponteiro de memória distante, indicado pela definição WORD ptr DS:[AX].
PTR16:16	Definição de operação com uso de valor imediato de ponteiro distante de segmento e deslocamento de memória. O formato 16:16 corresponde à definição de segmento:deslocamento descrita na forma NNNN:NNNN, em que NNNN são valores entre 0000h e FFFFh.
RW	Operação realizada em registro com tamanho de um word, podendo ser efetuado nos registradores AX, CX, DX, BX, SP, BP, SI ou DI.
RMB	Definição de operação realizada em endereços de registro ou endereços de memória com tamanho de um byte.
RMW	Definição de operação realizada em endereços de registro ou endereços de memória com tamanho de um word.

Notação Utilizada para Assembly/Opcodes	
Símbolo	**Significado**
VIB	Definição de valor imediato de tamanho de um byte entre 00h e FFh a ser definido para uma dada operação.
VIW	Definição de valor imediato de tamanho de um word entre 0000h e FFFFh a ser definido para uma dada operação.

A tabela de **Instruções Assembly e Opcode** apresenta a relação alfabética das Instruções *Assembly* em valores hexadecimais e os dados são apresentados com as simbologias descritas de acordo com as tabelas anteriores. Essa tabela apresenta as instruções suportadas pelo programa **emu8086**. Por esta razão, nela não são descritas algumas instruções 8086/8088, como INT3, LOCK e WAIT.

Instruções Assembly e Opcodes			
Instrução	**Opcodes e Dados**	**Relaciona-se com as Instruções**	**Bytes**
AAA	37	AAD AAM AAS	1
AAD	D5	AAA AAM AAS	2
AAM	D4	AAA AAD AAS	2
AAS	3F	AAA AAD AAM	1
ADC RMB, AL	10	AAD SBB SUB	2-4
ADC RMW, AX	11	AAD SBB SUB	2-4
ADC AL, RMB	12	AAD SBB SUB	2-4
ADC AX, RMW	13	AAD SBB SUB	2-4
ADC AL, VIB	14 VIB	AAD SBB SUB	3-4
ADC AX, VIW	15 VIW	AAD SBB SUB	3-4
ADC RMB, VIB	80 GRP1/2 VIB	AAD SBB SUB	3-6
ADC RMW, VIW	81 GRP1/2 VIW	AAD SBB SUB	3-6
ADC RMW, VIB	83 GRP1/2 VIB	AAD SBB SUB	3-6
ADD RMB, AL	00	ADC SBB SUB	2-4
ADD RMW, AX	01	ADC SBB SUB	2-4
ADD AL, RMB	02	ADC SBB SUB	2-4
ADD AX, RMW	03	ADC SBB SUB	2-4
ADD AL, VIB	04 VIB	ADC SBB SUB	3-4
ADD AX, VIW	05 VIW	ADC SBB SUB	3-4
ADD RMB, VIB	80 GRP1/0 VIB	ADC SBB SUB	3-6
ADD RMW, VIW	81 GRP1/0 VIW	ADC SBB SUB	3-6
ADD RMW, VIB	83 GRP1/0 VIB	ADC SBB SUB	3-6

Instruções Assembly e Opcodes			
Instrução	Opcodes e Dados	Relaciona-se com as Instruções	Bytes
AND RMB, AL	20	TEST OR NOT NEG XOR	2-4
AND RMW, AX	21	TEST OR NOT NEG XOR	2-4
AND AL, RMB	22	TEST OR NOT NEG XOR	2-4
AND AX, RMW	23	TEST OR NOT NEG XOR	2-4
AND AL, VIB	24 VIB	TEST OR NOT NEG XOR	3-4
AND AX, VIW	25 VIW	TEST OR NOT NEG XOR	3-4
AND RMB, VIB	80 GRP1/4 VIB	TEST OR NOT NEG XOR	3-6
AND RMW, VIW	81 GRP1/4 VIW	TEST OR NOT NEG XOR	3-6
AND RMW, VIB	83 GRP1/4 VIB	TEST OR NOT NEG XOR	3-6
CALL PTR16:16 (FAR)	9A	RET	5
CALL NOME (NEAR)	E8	RET	3
CALL RMW (NEAR)	FF GRP5/2	RET	2
CALL PW (FAR)	FF GRP5/3	RET	2-4
CBW	98	CWD	1
CLC	F8	STC CMD	1
CLD	FC	CMPS LODS MOVS SCAS STD STOS	1
CLI	FA	STI	1
CMC	F5	CLC STC	1
CMP RMB, AL	38	SUB CMPS SCAS	2-4
CMP RMW, AX	39	SUB CMPS SCAS	2-4
CMP AL, RMB	3A	SUB CMPS SCAS	2-4
CMP AX, RMW	3B	SUB CMPS SCAS	2-4
CMP AL, VIB	3C VIB	SUB CMPS SCAS	3-4
CMP AX, VIW	3D VIW	SUB CMPS SCAS	3-4
CMP RMB, VIB	80 GRP1/7 VIB	SUB CMPS SCAS	3-6
CMP RMW, VIW	81 GRP1/7 VIW	SUB CMPS SCAS	3-6
CMP RMW, VIB	83 GRP1/7 VIB	SUB CMPS SCAS	3-6
CMPSB	A6	CMP SCAS	1
CMPSW	A7	CMP SCAS	1
CWD	99	CBW	1
DAA	27	DAS	1
DAS	2F	DAA	1

Instruções Assembly e Opcodes			
Instrução	Opcodes e Dados	Relaciona-se com as Instruções	Bytes
DEC AX	48	INC SUB	1-2
DEC CX	49	INC SUB	1-2
DEC DX	4A	INC SUB	1-2
DEC BX	4B	INC SUB	1-2
DEC SP	4C	INC SUB	1-2
DEC BP	4D	INC SUB	1-2
DEC SI	4E	INC SUB	1-2
DEC DI	4F	INC SUB	1-2
DEC RMB	FE GRP4/1	INC SUB	2-4
DEC RMW	FF GRP5/1	INC SUB	2-4
DIV BL	F6 GRP3a/6	MUL	2
DIV BX	F7 GRP3b/6	MUL	2
HLT	F4	STI CLI	1
IDIV BL	F6 GRP3a/7	IMUL	2-4
IDIV BX	F7 GRP3a/7	IMUL	2-4
IMUL RMB	F6 GRP3a/5	IDIV	2
IMUL RMW	F7 GRP3b/5	IDIV	2
IN AL, VIB	E4 VIB	OUT	2
IN AX, VIB	E5 VIB	OUT	2
IN AL, DX	EC	OUT	2
IN AX, DX	ED	OUT	1
INC AX	40	ADD DEC	1
INC CX	41	ADD DEC	1
INC DX	42	ADD DEC	1
INC BX	43	ADD DEC	1
INC SP	44	ADD DEC	1
INC BP	45	ADD DEC	1
INC SI	46	ADD DEC	1
INC DI	47	ADD DEC	1
INC RMB	FE GRP4/0	ADD DEC	2-4
INC RMW	FF GRP5/0	ADD DEC	2-4
INT VIB	CD VIB	INTO	2

Instruções Assembly e Opcodes			
Instrução	Opcodes e Dados	Relaciona-se com as Instruções	Bytes
INTO	CE	INT	1
IRET	CF	INT INTO	1
JA/JNBE NOME	77 DRB	JMP	2
JAE/JNB NOME	73 DRB	JMP	2
JB/JC/JNAE NOME	72 DRB	JMP	2
JBE/JNA NOME	76 DRB	JMP	2
JCXZ NOME	E3 DRB	JMP	2
JE/JZ NOME	74 DRB	JMP	2
JG/JNLE NOME	7F DRB	JMP	2
JGE/JNL NOME	7D DRB	JMP	2
JL/JNGE NOME	7C DRB	JMP	2
JLE/JNG NOME	7E DRB	JMP	2
JMP NOME	E9 DRW	Todas as demais de saltos	2
JMP PTR16:16	EA DRD	Todas as demais de saltos	2-4
JMP NOME	EB DRB	Todas as demais de saltos	2
JMP RMW	FF GRP5/4	Todas as demais de saltos	2-4
JMP PW	FF GRP5/5	Todas as demais de saltos	2-4
JNC NOME	73 DRB	JMP	2
JNE/JNZ NOME	75 DRB	JMP	2
JNO NOME	71 DRB	JMP	2
JNP/JPO NOME	7B DRB	JMP	2
JNS NOME	79 DRB	JMP	2
JO NOME	70 DRB	JMP	2
JP/JPE NOME	7A DRB	JMP	2
JS NOME	78 DRB	JMP	2
LAHF	9F	SAHF	1
LDS AX, PW	C5	Não possui	2-4
LEA AX, RMW	8D	MOV	2-4
LES AX, PW	C4	Não possui	2-4
LODSB	AC	MOVS STOS	1
LODSW	AD	MOVS STOS	1
LOOP NOME	E2 DRB	Não possui	2

Instruções Assembly e Opcodes			
Instrução	Opcodes e Dados	Relaciona-se com as Instruções	Bytes
LOOPE/LOOPZ NOME	E1 DRB	Não possui	2
LOOPNE/LOOPNZ NOME	E0 DRB	Não possui	2
MOV RMB, AL	88	MOVS	2-4
MOV RMW, AX	89	MOVS	2-4
MOV AL, RMB	8A	MOVS	2-4
MOV AX, RMW	8B	MOVS	2-4
MOV RW, ES	8C	MOVS	2
MOV ES, RW	8E	MOVS	2
MOV AL, MB	A0 VIW	MOVS	2-4
MOV AX, MW	A1 VIW	MOVS	2-4
MOV MB, AL	A2 VIW	MOVS	2-4
MOV MW, AX	A3 VIW	MOVS	2-4
MOV AL, VIB	B0 VIB	MOVS	2-3
MOV CL, VIB	B1 VIB	MOVS	2-3
MOV DL, VIB	B2 VIB	MOVS	2-3
MOV BL, VIB	53 VIB	MOVS	2-3
MOV AH, VIB	B4 VIB	MOVS	2-3
MOV CH, VIB	B5 VIB	MOVS	2-3
MOV DH, VIB	B6 VIB	MOVS	2-3
MOV BH, VIB	B7 VIB	MOVS	2-3
MOV AX, VIW	B8 VIW	MOVS	2-3
MOV CX, VIW	B9 VIW	MOVS	2-3
MOV DX, VIW	BA VIW	MOVS	2-3
MOV BX, VIW	BB VIW	MOVS	2-3
MOV SP, VIW	BC VIW	MOVS	2-3
MOV BP, VIW	BD VIW	MOVS	2-3
MOV SI, VIW	BE VIW	MOVS	2-3
MOV DI, VIW	BF VIW	MOVS	2-3
MOV RMB, VIB	C6 /0 VIB	MOVS	3-6
MOV RMW, VIW	C7 /0 VIW	MOVS	3-6
MOVSB	A4	MOV LODS STOS	1
MOVSW	A5	MOV LODS STOS	1

Instruções Assembly e Opcodes			
Instrução	Opcodes e Dados	Relaciona-se com as Instruções	Bytes
MUL AL	F6 GRP3a/4	DIV	2
MUL AX	F7 GRP3B/4	DIV	2
NEG RMB	F6 GRP3a/3	AND NOT OR XOR	2-4
NEG RMW	F7 GRP3b/3	AND NOT OR XOR	2-4
NOP	90	Não possui	1
NOT RMB	F6 GRP3a/2	AND NEG OR XOR	2
NOT RMW	F7 GRP3b/2	AND NEG OR XOR	2
OR RMB, AL	08	AND NEG NOT XOR	2-4
OR RMW, AX	09	AND NEG NOT XOR	2-4
OR AL, RMB	0A	AND NEG NOT XOR	1
OR AX, RMW	0B	AND NEG NOT XOR	1
OR AL, VIB	0C VIB	AND NEG NOT XOR	-
OR AX, VIW	0D VIW	AND NEG NOT XOR	-
OR RMB, VIB	80 GRP1/1 VIB	AND NEG NOT XOR	3-6
OR RMW, VIW	81 GRP1/1 VIW	AND NEG NOT XOR	3-6
OR RMW, VIB	83 GRP1/1 VIB	AND NEG NOT XOR	3-6
OUT VIB, AL	E6 VIB	IN	2
OUT VIB, AX	E7 VIW	IN	2
OUT DX, AL	EE	IN	1
OUT DX, AX	EF	IN	1
POP ES	07	PUSH	1
POP SS	17	PUSH	1
POP DS	1F	PUSH	1
POP AX	58	PUSH	1
POP CX	59	PUSH	1
POP DX	5A	PUSH	1
POP BX	5B	PUSH	1
POP SP	5C	PUSH	1
POP BP	5D	PUSH	1
POP SI	5E	PUSH	1
POP DI	5F	PUSH	1
POP RMW	8F /0 VIW	PUSH	2

Instruções Assembly e Opcodes			
Instrução	Opcodes e Dados	Relaciona-se com as Instruções	Bytes
POPA	61	PUSHA	1
POPF	9D	PUSHF	1
PUSH ES	06	POP	1
PUSH CS	0E	POP	1
PUSH SS	16	POP	1
PUSH DS	1E	POP	1
PUSH AX	50	POP	1
PUSH CX	51	POP	1
PUSH DX	52	POP	1
PUSH BX	53	POP	1
PUSH SP	54	POP	1
PUSH BP	55	POP	1
PUSH SI	56	POP	1
PUSH DI	57	POP	1
PUSH VIW	68 VIW	POP	1
PUSH VIB	6A VIB	POP	1
PUSH RMW	FF GRP5/6	POP	2-4
PUSHA	60	POPA	1
PUSHF	9C	POPF	1
RCL RMB, VIB	C0 GRP2/2 VIB	RCR ROL ROR	3-5
RCL RMW, VIB	C1 GRP2/2 VIB	RCR ROL ROR	3-5
RCL RMB, 1	D0 GRP2/2	RCR ROL ROR	2-4
RCL RMW, 1	D1 GRP2/2	RCR ROL ROR	2-4
RCL RMB, CL	D2 GRP2/2	RCR ROL ROR	2-4
RCL RMW, CL	D3 GRP2/2	RCR ROL ROR	2-4
RCR RMB, VIB	C0 GRP2/3 VIB	RCL ROR ROL	3-3
RCR RMW, VIB	C1 GRP2/3 VIB	RCL ROR ROL	3-5
RCR RMB, 1	D0 GRP2/3	RCL ROR ROL	2-4
RCR RMW, 1	D1 GRP2/3	RCL ROR ROL	2-4
RCR RMB, CL	D2 GRP2/3	RCL ROR ROL	2-4
RCR RMW, CL	D3 GRP2/3	RCL ROR ROL	2-4
REP/REPNE/REPNZ	F2	Não possui	1

Instruções Assembly e Opcodes			
Instrução	Opcodes e Dados	Relaciona-se com as Instruções	Bytes
REPE/REPZ	F3	Não possui	1
RET VIW	C2 VIW	CALL	3
RET	C3	CALL	1
RETF VIW	CA VIW	CALL	3
RETF	DRB	CALL	1
ROL RMB, VIB	C0 GRP2/0 VIB	RCL RCR ROR	3
ROL RMW, VIB	C1 GRP2/0 VIB	RCL RCR ROR	3-5
ROL RMB, 1	D0 GRP2/0	RCL RCR ROR	2-4
ROL RMW, 1	D1 GRP2/0	RCL RCR ROR	2-4
ROL RMB, CL	D2 GRP2/0	RCL RCR ROR	2
ROL RMW, CL	D3 GRP2/0	RCL RCR ROR	2
ROR RMB, VIB	C0 GRP2/1 VIB	RCL RCR ROR	3
ROR RMW, VIB	C1 GRP2/1 VIB	RCL RCR ROR	3-5
ROR RMB, 1	D0 GRP2/1	RCL RCR ROR	2-4
ROR RMW, 1	D1 GRP2/1	RCL RCR ROR	2-4
ROR RMB, CL	D2 GRP2/1	RCL RCR ROR	2
ROR RMW, CL	D3 GRP2/1	RCL RCR ROR	2
SAHF	9E	LAHF	1
SAR RMB, VIB	C0 GRP2/7 VIB	SAL SHL SHR	3-5
SAR RMW, VIB	C1 GRP2/7 VIB	SAL SHL SHR	3-5
SAR RMB, 1	D0 GRP2/7	SAL SHL SHR	2-4
SAR RMW, 1	D1 GRP2/7	SAL SHL SHR	2-4
SAR RMB, CL	D2 GRP2/7	SAL SHL SHR	2-4
SAR RMW, CL	D3 GRP2/7	SAL SHL SHR	2-4
SBB RMB, AL	18	SUB ADD ADC	2-4
SBB RMW, AX	19	SUB ADD ADC	2-4
SBB AL, RMB	1ª	SUB ADD ADC	2-4
SBB AX, RMW	1B	SUB ADD ADC	2-4
SBB AL, VIB	1C VIB	SUB ADD ADC	3-4
SBB AX, VIW	1D VIW	SUB ADD ADC	3-4
SBB RMB, VIB	80 GRP1/3 VIB	SUB ADD ADC	3-6
SBB RMW, VIW	81 GRP1/3 VIW	SUB ADD ADC	3-6

Instruções Assembly e Opcodes			
Instrução	Opcodes e Dados	Relaciona-se com as Instruções	Bytes
SBB RMW, VIB	83 GRP1/3 VIB	SUB ADD ADC	3-6
SEG. CS (SEGMENTO)	2E	-	-
SEG. DS (SEGMENTO)	3E	-	-
SEG. ES (SEGMENTO)	26	-	-
SEG. SS (SEGMENTO)	36	-	-
SCASB	AE	CMP	1
SCASW	AF	CMP	1
SHL/SAL RMB, 1	D0 GRP2/4	SAR SHR	2-4
SHL/SAL RMB, CL	D2 GRP2/4	SAR SHR	2-4
SHL/SAL RMB, VIB	C0 GRP2/4 VIB	SAR SHR	2-4
SHL/SAL RMW, VIB	C1 GRP2/4 VIB	SAR SHR	2-4
SHL/SAL RMW, 1	D1 GRP2/4	SAR SHR	2-4
SHL/SAL RMW, CL	D3 GRP2/4	SAR SHR	2-4
SHR RMB, VIB	C0 GRP2/5 VIB	SHL SAL SAR	3
SHR RMW, VIB	C1 GRP2/5 VIB	SHL SAL SAR	3
SHR RMB, 1	D0 GRP2/5	SHL SAL SAR	2-4
SHR RMW, 1	D1 GRP2/5	SHL SAL SAR	2-4
SHR RMB, CL	D2 GRP2/5	SHL SAL SAR	2-4
SHR RMW, CL	D3 GRP2/5	SHL SAL SAR	2-4
STC	F9	CLC CMC	1
STD	FD	CLD LODS MOVS SCAS STOS	1
STI	FB	CLI	1
STOSB	AA	LODS MOVS	1
STOSW	AB	LODS MOVS	1
SUB RMB, AL	28	ADC ADD SBB	2-4
SUB RMW, AX	29	ADC ADD SBB	2-4
SUB AL, RMB	2A	ADC ADD SBB	2-4
SUB AX, RMW	2B	ADC ADD SBB	2-4
SUB AL, VIB	2C VIB	ADC ADD SBB	3-4
SUB AX, VIW	2D VIW	ADC ADD SBB	3-4
SUB RMB, VIB	80 GRP1/5 VIB	ADC ADD SBB	3-6
SUB RMW, VIW	81 GRP1/5 VIW	ADC ADD SBB	3-6

Instruções Assembly e Opcodes			
Instrução	Opcodes e Dados	Relaciona-se com as Instruções	Bytes
SUB RMW, VIB	83 GRP1/5 VIB	ADC ADD SBB	3-6
TEST RMB, AL	84	AND CMP	2-4
TEST RMW, AX	85	AND CMP	2-4
TEST AL, VIB	A8 VIB	AND CMP	3-4
TEST AX, VIW	A9 VIW	AND CMP	3-4
TEST RMB, VIB	F6 GRP3a/0 VIB	AND CMP	3-6
TEST RMW,VIW	F7 GRP3b/0 VIW	AND CMP	3-6
XCHG RMB, AL	86	Não possui	1
XCHG AL, RMB	86	Não possui	2-4
XCHG RMW, AX	87	Não possui	1
XCHG AX, RMW	87	Não possui	2-4
XCHG CX, AX	91	Não possui	2
XCHG DX, AX	92	Não possui	2
XCHG BX, AX	93	Não possui	2
XCHG SP, AX	94	Não possui	2
XCHG BP, AX	95	Não possui	2
XCHG SI, AX	96	Não possui	2
XCHG DI, AX	97	Não possui	2
XLATB	D7	Não possui	1
XOR RMB, AL	30	OR AND NOT NEG	2-4
XOR RMW, AX	31	OR AND NOT NEG	2-4
XOR AL, RMB	32	OR AND NOT NEG	2-4
XOR AX, RMW	33	OR AND NOT NEG	2-4
XOR AL, VIB	34 VIB	OR AND NOT NEG	3-4
XOR AX, VIW	35 VIW	OR AND NOT NEG	3-4
XOR RMW, VIW	81 GRP1/6 VIW	OR AND NOT NEG	3-6
XOR RMB, VIB	80 GRP1/6 VIB	OR AND NOT NEG	3-6
XOR RMW, VIB	83 GRP1/6 VIB	OR AND NOT NEG	3-6

Na coluna que apresenta o relacionamento entre as instruções, as indicações CMPS LODS MOVS SCAS STOS devem ser lidas como CMPSB, CMPSW, LODSB, LODSW, MOVSB, MOVSW, SCASB e SACASW.

Algumas instruções relacionadas possuem o mesmo valor de *opcode*. Isso ocorre com algumas instruções que pertencem ao mesmo grupo operacional. Destacam-se seis grupos operacionais de instruções, a saber: GRP1, GRP2, GRP3a, GRP3b, GRP4 e GRP5.

O grupo GRP1 relaciona-se às instruções ADD, OR, ADC, SBB, AND, SUB, XOR ou CMP representadas pelos *opcodes* 80, 81, 82 e 83. A definição de qual instrução estará associada ao *opcode* indicado depende do valor numérico após a barra, podendo ser 0, 1, 2, 3, 4, 5, 6 ou 7. Desta forma, /0 representa ADD, /1 representa OR, /2 representa ADC, /3 representa SBB, /4 representa AND, /5 representa SUB, /6 representa XOR e /7 representa CMP.

O grupo GRP2 relaciona-se às instruções ROL, ROR, RCL, RCR, SHL/SAL, SHR ou SAR representadas pelos opcodes D0, D1, D2 e D3. A definição de qual instrução estará associada ao *opcode* indicado depende do valor numérico após a barra, podendo ser 0, 1, 2, 3, 4, 5 ou 7. Desta forma, /0 representa ROL, /1 representa ROR, /3 representa RCL, /4 representa SHL, /5 representa SHR e /7 representa SAR. A operação /6 é nula para esse grupo.

Os grupos GRP3a e GRP3b relacionam-se às instruções TEST, NOT, NEG, MUL, IMUL, DIV ou IDIV representadas, respectivamente, pelos opcodes F6 e F7. A definição de qual instrução estará associada ao *opcode* indicado depende do valor numérico após a barra, podendo ser 0, 2, 3, 4, 5, 6 ou 7. Desta forma, /0 representa TEST, /2 representa NOT, /3 representa NEG, /4 representa MUL, /5 representa IMUL, /6 representa DIV e /7 representa IDIV. A operação /1 é mula para esses grupos.

O grupo GRP4 relaciona-se ao uso das instruções INC e DEC representadas pelo *opcode* FE. A definição de qual instrução estará associada ao opcode indicado depende do valor numérico após a barra, podendo ser 0 ou 1. Desta forma, /0 representa INC e /1 representa DEC. As demais operações são nulas para esse grupo.

O grupo GRP5 relaciona-se ao uso das instruções INC, DEC, CALL (1), CALL (2), JMP (1), JMP (2) ou PUSH representadas pelo *opcode* FF. A definição de qual instrução estará associada ao *opcode* indicado depende do valor numérico após a barra, podendo ser 0, 1, 2, 3, 4, 5, ou 6. Desta forma, /0 representa INC, /1 representa DEC, /2 representa CALL como modo 1, /3 representa CALL como modo 2, /4 representa JMP como modo 1, /5 representa JMP como modo 2 e /6 representa PUSH. A operação /7 é nula para esse grupo.

Observe em seguida o mapa de *opcodes* dividido em parte 1, parte 2 e parte de extensão. Para interpretação do mapa, veja primeiramente a linha e depois a coluna.

Mapa de opcodes - Parte 1								
	0	1	2	3	4	5	6	7
0	ADD RMB,AL	ADD RMW,AX	ADD AL,RMB	ADD AX,RMW	ADD AL,VIB	ADD AX,VIW	PUSH ES	POP ES
1	ACD RMB,AL	ADC RMW,AX	ADC AL,RMB	ADC AX,RMW	ADC AL,VIB	ADC AX,VIW	PUSH SS	POP SS
2	AND RMB,AL	AND RMW,AX	AND AL,RMB	AND AX,RMW	AND AL,VIB	AND AX,VIW	SEG. ES	DAA

Mapa de opcodes - Parte 1								
	0	1	2	3	4	5	6	7
3	XOR RMB,AL	XOR RMW,AX	XOR AL,RMB	XOR AX,RMW	XOR AL,VIB	XOR AX,VIW	SEG. SS	AAA
4	INC AX	INC CX	INC DX	INC BX	INC SP	INC BP	INC SI	INC DI
5	PUSH AX	PUSH CX	PUSH DX	PUSH BX	PUSH SP	PUSH BP	PUSH SI	PUSH DI
6	-	-	-	-	-	-	-	-
7	JO NOME	JNP NOME JNO NOME	JB NOME JC NOME JNAE NOME	JAE ROT08 JNB NOME	JE NOME JZ NOME	JNE NOME JNZ NOME	JBE NOME JNA NOME	JA NOME JNBE NOME
8	GRP1	GRP1	GRP1	GRP1	TEST RMB,AL	TEST RMW,AX	XCHG RMB,AL XCHG AL,RMB	XCHG RMW,AX XCHG AX,RMW
9	NOP	XCHG CX AX	XCHG DX AX	XCHG BX AX	XCHG SP AX	XCHG BP AX	XCHG SI AX	XCHG DI AX

Mapa de opcodes - Parte 1								
	0	1	2	3	4	5	6	7
A	MOV AL,MB	MOV AX,MW	MOV MB,AL	MOV MW,AX	MOVSB	MOVSW	CMPSB	CMPSW
B	MOV AL,VIB	MOV CL,VIB	MOV DL,VIB	MOV BL,VIB	MOV AH,VIB	MOV CH,VIB	MOV DH,VIB	MOV BH,VIB
C	-	-	RET VIW	RET	LES AX,PW	LDS AX, PW	MOV RMB,VIB	MOV RMW,VIW
D	GRP2	GRP2	GRP2	GRP2	AAM	AAD	-	XLATB
E	LOOPNZ NOME LOPPNE NOME	LOOPZ NOME LOOPE NOME	LOOP NOME	JCXZ NOME	IN AL,VIB	IN AX,VIB	OUT VIB,AL	OUT VIB,AX
F	LOCK	-	REP REPNE REPNZ	REPE REPZ	HLT	CMC	GRP3a	GRP3b

Mapa de opcodes - Parte 2								
	8	9	A	B	C	D	E	F
0	OR RMB,AL	OR RMW,AX	OR AL,RMB	OR AX,RMW	OR AL,VIB	OR AX,VIW	PUSH CS	-

Mapa de opcodes - Parte 2								
1	SBB RMB,AL	SBB RMW,AX	SBB AL,RMB	SBB AX,RMW	SBB AL,VIB	SBB AX,VIW	PUSH DS	POP DS
2	SUB RMB,AL	SUB RMW,AX	SUB AL,RMB	SUB AX,RMW	SUB AL,VIB	SUB AX,VIW	SEG. CS	DAS
3	CMP RMB,AL	CMP RMW,AX	CMP AL,RMB	CMP AX,RMW	CMP AL,VIB	CMP AX,VIW	SEG. DS	AAS
4	DEC AX	DEC CX	DEC DX	DEC BX	DEC SP	DEC BP	DEC SI	DEC DI
5	POP AX	POP CX	POP DX	POP BX	POP SP	POP BP	POP SI	POP DI
6	-	-	-	-	-	-	-	-
7	JS NOME	JNS NOME	JP NOME JPE NOME	JPO NOME	JL NOME JNGE NOME	JGE NOME JNL NOME	JLE NOME JNG NOME	JG NOME JNLE NOME

Mapa de opcodes - Parte 2								
8	MOV RMB,AL	MOV RMW,AX	MOV AL,RMB	MOV AX,RMW	MOV RW,ES	LEA AX,RMW	MOV ES,RW	POP RMW
9	CBW	CWD	CALL PTR16:16	WAIT	PUSHF	POPF	SAHF	LAHF
A	TEST AL,VIB	TEST AX,VIW	STOSB	STOSW	LODSB	LODSW	SCASB	SCASW
B	MOV AX,VIW	MOV CX,VIW	MOV DX,VIW	MOV BX,VIW	MOV SP,VIW	MOV BP,VIW	MOV SI,VIW	MOV DI,VIW
C	-	-	REFF VIW	RETF	INT3	INT VIB	INTO	IRET
D	-	-	-	-	-	-	-	-
E	CALL NOME	JMP NOME	JMP PTR16:16	JMP NOME	IN AL,DX	IN AX,DX	OUT DX,AL	OUT DX,AX

Mapa de opcodes - Parte 2								
F	CLC	STC	CLI	STI	CLD	STD	GRP4	GRP5

Mapa de opcodes - Parte de Extensão								
	0	1	2	3	4	5	6	7
GRP1	ADD	OR	ADC	SBB	AND	SUB	XOR	CMP
GRP2	ROL	ROR	RCL	RCR	SHL/SAL	SHR	-	SAR
GRP3a	TEST	-	NOT	NEG	MUL	IMUL	DIV	IDIV
GRP3b	TEST	-	NOT	NEG	MUL	IMUL	DIV	IDIV
GRP4	INC	DEC	-	-	-	-	-	-
GRP5	INC	DEC	CALL	CALL	JMP	JMP	PUSH	-

Para a criação deste apêndice, foram utilizadas informações dos manuais dos microprocessadores da Intel e da AMD que constam na bibliografia deste trabalho, além da obra *The 8086/8088 Primer* de Sthephen P. Morse.

```
0D82:0100 B402          MOV     AH,02
0D82:0102 B241          MOV     DL,41
0D82:0104 CD21          INT     21
0D82:0106 CD20          INT     20
0D82:0108 69            DB      69
```

Apêndice B

ENHANCED DEBUG

Este apêndice abrange detalhes operacionais gerais do utilitário de depuração, montagem e desmontagem *Enhanced DEBUG* usado em algumas partes desta obra.

O programa Microsoft DEBUG e Enhanced DEBUG caracteriza-se por ser um utilitário de depuração (permite verificar o código de programa em baixo nível), montagem (permite usar o utilitário para escrever um programa em baixo nível) e desmontagem (permite olhar internamente o código de um programa em baixo nível). É possível com esta ferramenta fazer alterações e ajustes em um código de programa compilado em baixo nível, tanto "**.COM**", como "**.EXE**".

Para a demonstração dos comandos existentes no utilitário *Enhanced DEBUG* seguem alguns arquivos a serem criados como material de apoio a algumas operações. Observe, respectivamente, os arquivos em formato texto indicados nas figuras A.1, A.2 e A.3 contendo os códigos básicos de dois programas e um texto descritivo definidos como suporte a algumas operações exemplificadas junto ao utilitário DEBUG. Lembre-se de deixar esses arquivos no mesmo local onde o programa DEBUG se encontra.

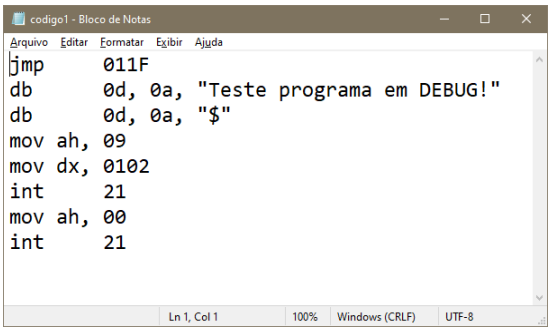

Figura A.1 - Código Assembly com uso de dados em formato byte.

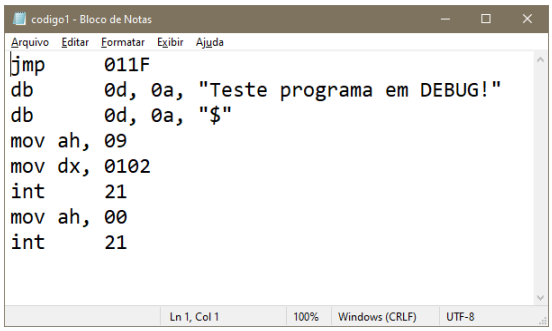

Figura A.2 - Código Assembly com uso de dados em formato word.

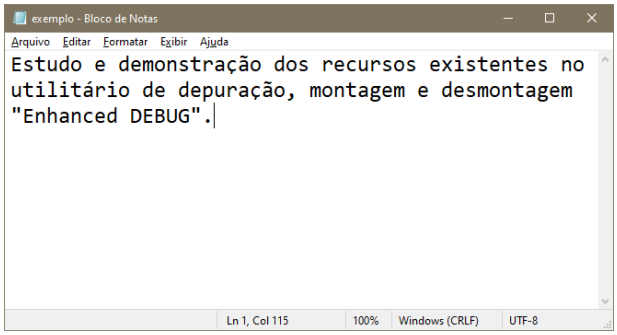

Figura A.3 - Texto com mensagem simples "exemplo.txt".

O programa *Enhanced DEBUG* pode ser executado a partir do *prompt* do sistema operacional com algumas definições iniciais:

```
debug [[unidade:] [caminho] nome do arquivo [parâmetros do arquivo]]
```

onde os parâmetros indicados como,

```
[unidade:] [caminho] nome do arquivo
```

são a localização do nome do arquivo executável que se deseja verificar a partir da unidade e caminho opcionalmente indicados, tendo possivelmente a indicação

```
[parâmetros do arquivo]
```

como sendo qualquer eventual informação de linha de comando exigida pelo arquivo executável a ser verificado.

Como comentado, dependendo de como se faz a chamada do programa tem-se uma diferente ocupação de memória. Veja a seguir duas situações comportamentais do programa *Enhanced DEBUG* carregado com e sem o uso de parâmetro a partir de seu modo padrão de operação.

Veja os detalhes a partir da figura A.4 para o uso da execução dos programas de depuração sem o uso de parâmetro, a partir da orientação indicada por Sedory (2017):

- Aloca os 64 KB do primeiro segmento de memória que esteja livre para uso, neste caso, representado pelo endereço de segmento 06AF;
- Os registradores de segmentos DS, ES, SS e CS são definidos com o valor do endereço do segmento de memória alocado com os 64 KB. Desta forma. DS=ES=SS=CS="posição do segmento", ou seja, 06AF;
- O ponteiro de instrução (IP) fica definido inicialmente, sempre por padrão, na posição de deslocamento 0100;
- O ponteiro de pilha SP fica definido como FFFE;
- Os registradores AX, BX, CX, DX, BP, SI e DI são sinalizados como zero;
- Os bits dos registradores de flags OF, DF SF, ZF, AF, PE e CF são zerados respectivamente com os valores NV, UP, PL, NZ, NA, PO e NC, exceção ao flag de interrupção IF configurado com valor 1 a partir do valor EI.

```
D:\>debug
-r
AX=0000 BX=0000 CX=0000 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=06AF ES=06AF SS=06AF CS=06AF IP=0100 NV UP EI PL NZ NA PO NC
06AF:0100 C3                RET
-
```

Figura A.4 - Programa "Enhanced DEBUG" carregado sem o uso de parâmetro.

Veja os detalhes a partir da figura A.5 para o uso da execução dos programas de depuração com o uso de parâmetro a partir de um programa ".COM", a partir da orientação indicada por Sedory (2017):

- Aloca pelo menos 64 KB do primeiro segmento de memória que esteja livre para depurar programas ou examinar arquivos especificados na linha de comando, neste caso, representado pelo endereço de segmento 06B4;
- Os registradores de segmentos DS, ES, SS e CS são definidos com o valor do endereço do segmento de memória alocado com os 64 KB. Desta forma. DS=ES=SS=CS="posição do segmento", ou seja, 06B4. Para arquivos maiores que 64KB, será necessário definir diferentes valores de segmento para acessar todos os bytes carregados na memória além dos primeiros 64 KB;
- O ponteiro de instrução (IP) fica definido inicialmente, sempre por padrão, na posição de deslocamento 0100;
- O ponteiro de pilha SP fica definido como FFFE;
- Os registradores BX, DX, BP, SI e DI são sinalizados como zero;
- Os registradores AX, CX ficam com valores diferentes de zero. O valor de CX (e por vezes em BX também) representa o tamanho em hexadecimal do arquivo carregado na memória. Para arquivos com menos de 64 KB (256 bytes), apenas CX é usado. Por exemplo: Se for carregado um arquivo de 360.247 bytes, então BX será 0005 e CX será 7F37 (57F37h = 360.247d). Se for carregado um arquivo com exatamente 65.536 bytes esses registradores serão BX com 0001 e CX com 0000. No entanto, devido ao deslocamento automático de carga no

endereço 0100h os últimos 256 bytes do arquivo serão carregados no início do próximo segmento de 64 KB. O valor de AX caracteriza por ser o valor inicial do programa passado como parâmetro carregado e apontado no endereço 0100h, que não vem ao caso.

- Os bits dos registradores de flags OF, DF SF, ZF, AF, PE e CF são zerados respectivamente com os valores NV, UP, PL, NZ, NA, PO e NC, exceção ao flag de interrupção IF configurado com valor 1 a partir do valor EI.

```
D:\>debug teste.com
-r
AX=FFFF BX=0000 CX=0061 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=06B4 ES=06B4 SS=06B4 CS=06B4 IP=0100 NV UP EI PL NZ NA PO NC
06B4:0100 B80300            MOV     AX,0003
-
```

Figura A.5 - Programa "Enhanced DEBUG" carregado com o uso de parâmetro.

A obtenção da relação de comandos do programa é obtida a qualquer momento a partir do uso do comando "**?**" (sinal de interrogação) no *prompt* de comando do programa. As figuras A.7 e A.8 apresentam a relação dos comandos existentes no utilitário *Enhanced DEBUG* (MASLOCH, 2021 & SEDORY, 2017).

```
assemble        A [address]
compare         C range address
dump            D[B|W|D] [range]
dump MCB chain  DM
enter           E address [list]
fill            F range list
go              G [=address] [breakpoints]
hex add/sub     H value1 value2
input           I[W|D] port
load file       L [address]
load sectors    L address drive sector count
move            M range address
name            N [[drive:][path]filename [arglist]]
output          O[W|D] port value
proceed         P [=address] [count]
proceed return  PR
quit            Q
register        R [register [value]]
FPU register    RN
toggle 386 regs RX
search          S range list
trace           T [=address] [count]
trace mode      TM [0|1]
unassemble      U [range]
[more]
```

Figura A.7 - Relação de comandos do utilitário Enhanced DEBUG (1/2).

```
dump MCB chain  DM
enter           E address [list]
fill            F range list
go              G [=address] [breakpoints]
hex add/sub     H value1 value2
input           I[W|D] port
load file       L [address]
load sectors    L address drive sector count
move            M range address
name            N [[drive:][path]filename [arglist]]
output          O[W|D] port value
proceed         P [=address] [count]
proceed return  PR
quit            Q
register        R [register [value]]
FPU register    RN
toggle 386 regs RX
search          S range list
trace           T [=address] [count]
trace mode      TM [0|1]
unassemble      U [range]
view flip       V
write file      W [address]
write sectors   W address drive sector count
-
```

Figura A.8 - Relação de comandos do utilitário Enhanced DEBUG (2/2).

Observe que alguns parâmetros são definidos entre colchetes "**[**" e "**]**", o que indica que são parâmetros opcionais. Veja a seguir o significado de cada parâmetro dentro do contexto de operação do programa.

PARÂMETRO	SIGNIFICADO
address (endereço)	Endereço de memória expresso com valor numérico hexadecimal.
arglist (lista de argumentos)	Parâmetros a serem informados a um programa carregado com o DEBUG.
breakpoints (pontos de interrupção)	Definição de um ponto de parada para a efetivação de testes e acompanhamentos de programas.
count (número)	Definição do número de setores consecutivos cujo conteúdo será carregado.
drive (unidade)	Indicação da unidade de disco a ser utilizada.
list (lista)	Sequência de bytes hexadecimais separados por espaço ou dados no formato ASCII incluídos em cadeias definidas com aspas simples ou inglesas.
path (caminho)	Indicação da pasta ou diretório de acesso em disco.
port (porta)	Definição de porta lógica a ser utilizada em alguma operação.
range (faixa)	Definição de um segmento de memória a partir de dois valores separados por espaço ou, em alguns casos, a partir de um valor inicial.
register (registrador)	Indicação de um registrador a ser utilizado.
sector (setor)	Indicação em hexadecimal de um setor de disco a ser utilizado para a operação.
value (número)	Definição para uso de um valor numérico no formato hexadecimal.

Além da ajuda interno, obtido por meio do comando "**?**", é possível obter informações adicionais de auxílio do programa no *prompt* do sistema antes de fazer seu carregamento em memória. Para tanto, use após o de *prompt* "**C:>**" a instrução "**debug /?**" e observe o resultado apresentado, como indicado na figura A.9.

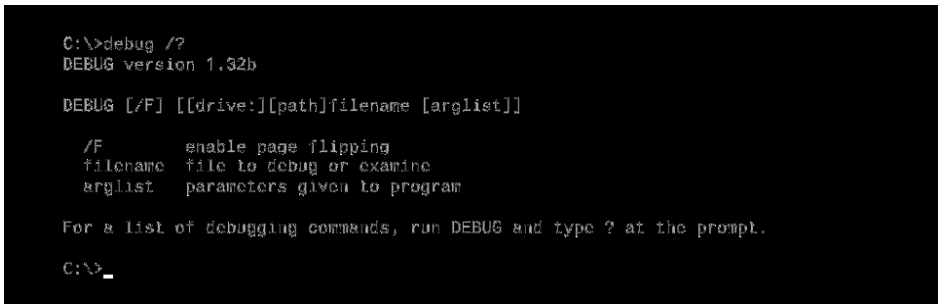

Figura A.9 - Apresentação de informações adicionais do utilitário Enhanced DEBUG.

O uso do parâmetro "**/F**" coloca o programa em execução na página 1 do vídeo, sem este parâmetro o programa entra na página 0. O uso do argumento "**filename**" caracteriza a chamada do DEBUG com a indicação de um arquivo a ser verificado ou modificado e o parâmetro "**arglist**" caracteriza-se por ser a indicação de eventuais parâmetros que o programa carregado em "**filename**" venha a fazer uso. Desta forma, são possíveis de uso algumas combinações, como:

```
C:> debug /?
C:> debug /f
C:> debug programa
C:> debug programa /arg
C:> debug /f programa
C:> debug /f programa /arg
C:> debug c:\teste\programa
C:> debug c:\teste\programa /arq
C:> debug /f c:\teste\programa
C:> debug /f c:\teste\programa /arg
```

A seguir são comentados e apresentados em ordem alfabética os comandos operacionalizados. Atenta para as indicações de textos em negrito e normal. Em texto negrito estará a indicação das entradas a serem realizadas por você e em texto normal estará o que o ambiente mostra para você.

```
A [endereço] - assemble (montar)
```

Este comando é usado para definir instruções de códigos de um programa na memória, devendo-se iniciar sua ação, sempre a partir do endereço **0100** do segmento **CS** que estiver ativo, neste caso, **06AF**. Neste modo de operação devem ser indicadas instruções em linguagem Assembly compatível com microprocessadores Intel/AMD dentro dos limites de 16/32 bits. O comando "**A**" mantém ativo na memória o último endereço usado para a definição de alguma operação. Isto significa que este comando ao ser usado sucessivamente faz a colocação do ponteiro **IP** no próximo endereço de acesso após o endereço anteriormente utilizado. Neste sentido, este comando é semelhante ao comando "**D**" (Dump) que também "lembra" a localização de seu último uso. A montagem de um programa na memória é encerrada quando a tecla "**<Enter>**" é acionada em uma linha vazia.

Observação

Para a demonstração dos exemplos a seguir escreva diretamente cada código na posição indicada ou abra respectivamente os arquivos "**codigo1.txt**" e "**codigo2.txt**", selecione o texto com as teclas de atalho "**<Ctrl>+<a>**", depois execute o atalho "**<Ctrl>+<c>**" e dentro do programa DEBUG a partir da posição **0100** execute as teclas de atalho "**<Ctrl>+<v>**".

Exemplo

Observe o resultado da operação na figura A.10 utilizando-se estrutura de 16 bits de dados.

```
-a 0100
06AF:0100 jmp     011F
06AF:0102 db      0d, 0a, "Teste programa em DEBUG!"
06AF:011C db      0d, 0a, "$"
06AF:011F mov ah, 09
06AF:0121 mov dx, 0102
06AF:0124 int     21
06AF:0126 mov ah, 00
06AF:0128 int     21
06AF:012A
-g =0100

Teste programa em DEBUG!

Program terminated (0000)
-
```

Figura A.10 - Exemplo de codificação de programa em Assembly em modo 16 bits.

Veja o resultado da operação de uso do comando "**A**" na figura A.11 utilizando-se estrutura de 32 bits de dados. Para maiores referências consulte o comando "**G**".

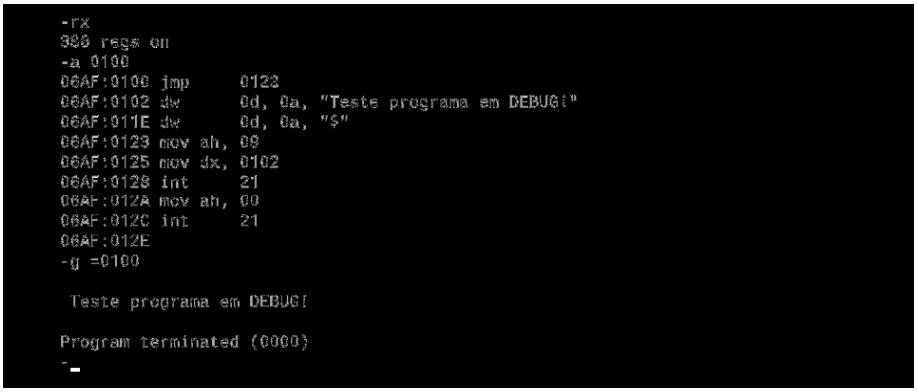

Figura A.11 - Exemplo de codificação de programa em Assembly em modo 32 bits.

```
C faixa de endereço – compare (comparar)
```

Este comando é usado para estabelecer a comparação dos bytes de dois blocos de memória. Sendo os conteúdos dos blocos iguais o comando mostra, na sequência, apenas seu *prompt*, mas se houver alguma diferença o programa apresenta os bytes que se diferem lado a lado com a indicação do local de memória onde encontra-se a diferença a partir do formato **endereço1 byte1 byte2 endereço2**. O parâmetro **faixa de endereço** (obrigatório) do comando "**C**" pode ser definido a partir de duas formas de uso: com a especificação dos valores de endereços inicial e final com **valor1**, **valor2** e **valor3**, em que **valor1** e **valor2** marcam o início e fim do endereço de origem e **valor3** marca o início do endereço de destino a serem comparados utilizando-se implicitamente a mesma distância definida entre **valor1** e **valor2**; ou com a especificação de um endereço inicial e seu comprimento de ação utilizando o argumento "**L**" que poder ser escrito tanto com caractere maiúsculo quanto caractere minúsculo.

Exemplos

Observe o resultado da operação na figura A.12.

Figura A.12 - Exemplo de comparação de áreas de memória.

```
D [B|W|D] [faixa] [comprimento] – dump (despejar)
```

Este comando mostra (despeja) o conteúdo existente em determinada área de memória indicada no parâmetro **faixa**. O comando "**D**" é utilizado com todos os parâmetros opcionais, onde **faixa** permite definir a área de memória a ser apresentada podendo fazer uso dos indicativos **valor1** para determinar o endereço inicial e **valor2** para determinar opcionalmente o endereço final, **comprimento** estabelece o tamanho a ser apresentado a partir do endereço inicial indicado como **valor1**.

Observação

Para a demonstração dos exemplos a seguir execute na linha de comando do sistema operacional a chamada da instrução "**debug /f exemplo.txt**".

Exemplo

```
C:> debug /f exemplo.txt
```

Veja o resultado da operação na figura A.13. Perceba que os caracteres acentuados em português são omitidos por não serem caracteres ASCII puros.

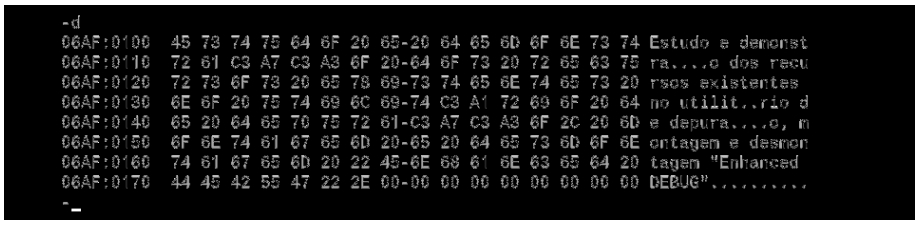

Figura A.13 - Exemplo de despejo de memória.

Observe outras formas de realização do despejo de dados da memória indicada na figura A.14.

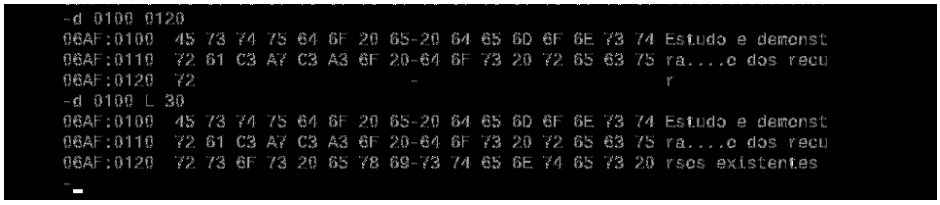

Figura A.14 - Outros exemplos de despejo de memória.

O comando "**D**" pode ser usado com as variantes "**B**" como comando "**DB**", "**W**" como comando "**DW**" e "**D**" como comando "**DD**" no sentido de apresentar o conteúdo de memória disposto na forma de *byte* "**B**", *word* "**W**" e *double word* "**D**" e alterar o modo de operação do DEBUG utilizando-se ou não os detalhes já comentados. Veja na figura A.15 exemplos dos resultados apresentados com "**DB**", "**DW**" e "**DD**".

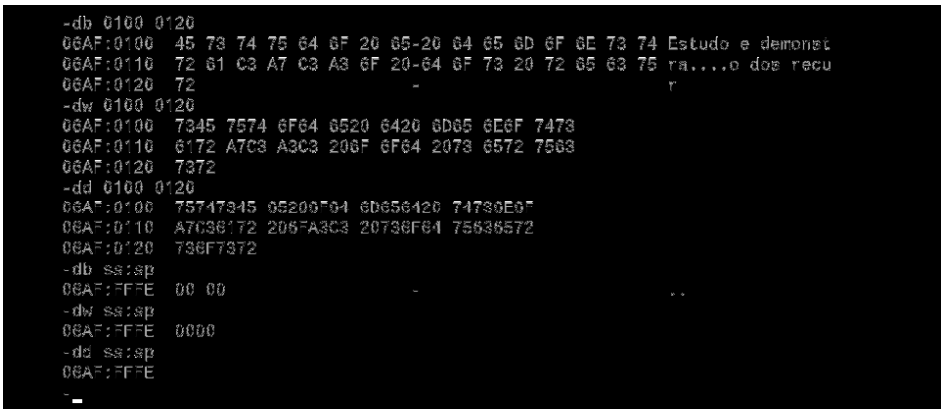

Figura A.15 - Exemplos de despejo de memória com "DB", "DW" e "DD".

Quando um dos comandos "**DB**", "**DW**" e "**DD**" é usado ocorre a mudança do modo de ação do ambiente DEBUG, ou seja, a partir do uso de um desses comandos o ambiente fica configurado ao modo solicitado até que outro comando dessa categoria seja executado. Para manter o desempenho em modo padrão garanta a execução do comando "**DB**".

```
DM – dump MCB chian (despejar cadeia MCB)
```

Este comando descarta a apresentação de uma cadeia MCB (*Memory Conflict Buffer* - Buffer de Conflito de Memória). O comando "**DM**" apresenta a lista PSP (*Process Segment Prefix* - Prefixo do Segmento de Processo) a partir do segmento de memória que esteja ativo até encontrar um MCB que não tenha uma letra de assinatura "M", "Z" ou o endereço MCB que extrapole o limite de 1 MB.

Exemplo

Observe o resultado da operação na figura A.16.

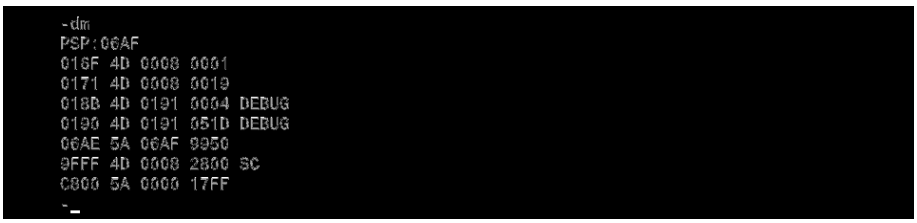

Figura A.16 - Exemplo de despejo MCB.

As colunas são as seguintes, a partir do indicativo da terceira linha após **PSP:06AF** considere o descrito a seguir para todas as linhas apresentadas.

- 1a coluna - **018B** - endereço do segmento do MCB.
- 2a coluna - **4D** - assinatura do MCB: "4D" para vincular "M" ao MCB e "5A" para vincular "Z" de outra forma.
- 3a coluna - **0191** - proprietário do MCB: valores abaixo de "50" são valores especiais do sistema, sendo "0" para um MCB não usado e "8" é o SC/SD/S proprietário do sistema MCB em uso; valores mais altos representam geralmente segmentos de processo (bloco de memória precedido por um MCB, que pertence ao bloco em si).
- 4a coluna - **0004** - quantidade de parágrafos do MCB: "0" indica bloco de memória vazio.
- 5a coluna - **DEBUG** - nome do proprietário do MCB, podendo ser grafado sem nome quando estão livres ou nomeados quando ocupados: MCB do sistema têm um nome de identificação com até duas letras, quando com mais letras o nome do MCB é obtido a partir dele mesmo tendo então até 8 letras.

```
E endereço [lista] - enter (entrada)
```

Este comando é usado para fazer a entrada de dados ou instruções em Assembly diretamente em endereços de memória indicados ou examinar o conteúdo de certa posição. O comando "**E**" opera com dois parâmetros, sendo o primeiro chamado **endereço** obrigatório e opcionalmente com um segundo chamado **lista**. Se o primeiro parâmetro for usado sem o segundo o comando entra em modo examinar/alterar e se ambos forem usados o comando opera diretamente no modo entrada de dados.

Exemplo

```
C:> debug /f exemplo.txt
```

Veja o resultado da operação na figura A.17 a partir da execução do comando "**d 0100 0105**" no *prompt* DEBUG. Atente para os valores apresentados **45** para a letra "**E**", **73** para a letra "**s**", **74** para a letra "**t**", **75** para a letra "**u**", **64** para a letra "**d**" e **6F** para a letra "**o**".

Figura A.17 - Área de despejo com palavra "Estudo" indicada.

A partir da palavra "**Exemplo**" armazenada entre a faixa de endereços de **0100** até **0105** é possível com o comando "**E**" fazer a alteração deste conteúdo, por exemplo, para "**Ensaio**". Para tanto, entre no *prompt* DEBUG a instrução "**e 0101 6e,73,61,69**" respectivamente para as letras "**n**", "**s**", "**a**", "**i**" e "**o**" (ou informe: **e 0101 "nsai"**) e em seguida execute a instrução "**d 0100 0105**" para ver o resultado indicado junto a figura A.18.

Figura A.18 - Resultado após o uso do comando "E".

Para examinar o conteúdo de memória com a possibilidade de realizar alteração use a instrução "**e 0101**". Neste caso é apresentado o valor da posição seguindo de um ponto. Neste momento, pode-se informar o novo valor para a posição e acionar a tecla "**<Enter>**" uma vez para registrar a alteração ou usar a tecla de "**<Espaço>**" para colocar o modo do comando no próximo endereço da memória para alterar ou não a "nova" posição. Caso efetue alguma alteração de forma incorreta use a tecla "**<menos>**" para voltar a uma posição e realizar sua alteração. Após realizar as ações desejadas basta acionar um último "**<Enter>**" para finalizar a operação.

Para alterar a palavra "**Ensaio**", tornando-a "**Estudo**" execute no *prompt* DEBUG a instrução "**e 0101**" e entre os detalhes indicados em negrito. As referências "**<Espaço>**" e "**<Enter>**" indicam que essas teclas devem ser acionadas durante a edição:

```
-e 0101
06FA:0101   6E.73 "<Espaço>" 73.74  "<Espaço>"  61.75  "<Espaço>"  69.64 "<Enter> 2x".
```

Na sequência execute, novamente, a instrução "**d 0100 0105**" e observe o resultado indicado junto a figura A.19.

Figura A.19 - Resultado após o uso do comando "E" a partir de posição definida.

```
F faixa lista - fill (preencher)
```

Este comando é usado para realizar o preenchimento de determinada área de memória com certo conjunto de bytes. Desta forma, uma das suas finalidades é realizar a limpeza de certa área de memória. O comando "**F**" usa dois parâmetros obrigatórios sendo o primeiro para a indicação do endereço de memória a ser tratado e o comprimento do preenchimento a ser executado (**faixa**) e o segundo (**lista**) a ser usado no preenchimento da área de memória.

Exemplo

```
C:> debug /f exemplo.txt
```

Inicialmente execute o comando "**d 0100**" para ver o conteúdo de parte do texto carregado e na sequência e efetue o uso das instruções "**f 0100 010F 'DEBUG '**" e " **d 0100**". Veja na figura A.20 a ocorrência no uso do comando "**F**".

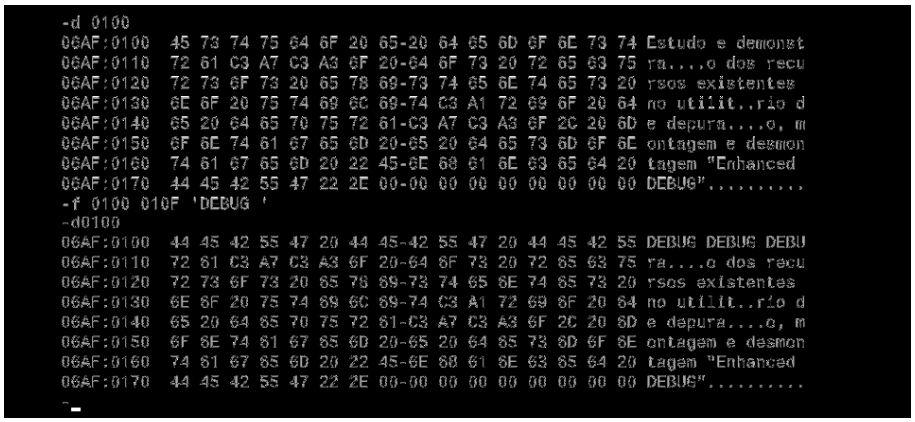

Figura A.20 - Resultado do preenchimento de dados com o comando "F".

Para realizar a limpeza dos dados de uma área de memória basta executar o uso do comando "**F**" o preenchimento desta área com o valor numérico "**00**" a partir da instrução "**f 0100 FFFF 00**". No entanto, o uso do comando "**F**" requer alguns cuidados pois se definido sobre uma área de memória dados que possua informações do programa DEBUG ou informações de uso do próprio sistema poderá inviabilizar sua ação ou gerar efeitos inesperados. Mantenha atenção e foco sobre isso.

```
G [[=endereço] [pontos de interrupção]] - go (ir)
```

Este comando é usado para executar um programa que esteja definido em memória ou usado para definir pontos de interrupção no código de um programa. O comando "**G**" usa dois parâmetros opcionais, sendo o primeiro a indicação do endereço onde o programa se inicia na memória e o segundo a definição do ponto de interrupção (**HALT**) até onde o programa se estende na memória. O segundo parâmetro pode ser omitido quando o programa possui internamente em seu código alguma instrução de encerramento ou interrupção. É possível definir para o segundo parâmetro até dez pontos de interrupção a partir do uso do comando "**G**" separando os endereços de interrupção com espaços em branco. Quando um ponto de interrupção é definido ocorre a interrupção da execução do programa no endereço imediatamente antes ao ponto definido para sua verificação.

Exemplo

Para realizar um teste de execução no uso do comando "**G**" é necessário ter em memória o código de um programa definido para execução. Veja na figura A.21 um código de programa definido a partir do comando "**A**" e sua execução a partir do uso da instrução "**g =0100**".

Figura A.21 - Exemplo de execução de programa com comando "G".

O comando "**G**" pode ser usado sem parâmetros e neste caso apenas executa o código que estiver indicado no registrador "**CS**" a partir da posição indicada no registrador "**IP**", se usado com a indicação do endereço, por exemplo, como em "**0100**" inicia o programa nesta posição a partir do segmento "**CS**" e se usado, por exemplo, com o formato "**g 0100 012A**" inicia o programa na posição **0100** e encerra a execução do programa antes do ponto de interrupção estabelecido no endereço **012A**, ou seja, encerra o programa no endereço **0128**.

Quando forem definidos mais de um ponto de interrupção a partir do primeiro ponto de interrupção definido (como em **012A**) é possível fazer usos sucessivos do comando "**G**" para avançar até o próximo ponto de interrupção. No entanto, esses pontos de interrupção somente podem ser definidos nos endereços de memória que tenham códigos válidos para os *opcodes* Assembly sob risco das ações de parada definidas deixarem de surtir efeito na operação.

```
H valor1 valor2 – hex add/sub (adição/subtração hexadecimal)
```

Este comando é usado para realizar simultaneamente as operações de adição e subtração de dois valores numéricos hexadecimais. O comando "**H**" usa dois parâmetros obrigatórios representados por **valor1** e **valor2**. Esses parâmetros podem ser operacionalizados com valores de até quatro dígitos. O resultado da diferença ocorre a partir do **valor2** sobre o **valor1** e utiliza o efeito de *complemento de dois* para representar valores que estejam acima do limite máximo permitido.

Exemplo

Observe na figura A.22 alguns exemplos de operações utilizando-se valores numéricos.

Figura A.22 - Resultados de adições e subtração com o comando "H".

```
I [W|D] porta - input (porta de entrada)
```

Este comando é usado para realizar entrada, ou seja a leitura, de uma porta x86 e mostrar seu conteúdo. O comando "**I**" usa um parâmetro obrigatório que deve ser indicado a partir de uma porta de 16 bits especificada entre os valores **0000** até **FFFF**. Os comandos "**IW**" e "**ID**" fazem a inserção de valores no formato **word** ou **dword** (*double-word*). Este comando permite ler as portas de entrada e aos códigos escritos nos endereços das portas de saída.

```
L [endereço] [[unidade] [setor] [número]] - load file (carrega arquivo)
```

Este comando é usado para realizar o carregamento de um arquivo especificado como argumento do programa DEBUG ou usado após o comando "**N**". É possível com o comando "**L**" carregar os setores especificados de um disco na memória sem levar em consideração o sistema de arquivos em uso. O argumento **endereço** é o local da memória onde os dados são copiados (use apenas quatro dígitos para mantê-los dentro da área de memória alocada), o argumento **unidade** refere-se a um valor de mapeado da unidade de disco do computador, como: **0** para **A:**, **1** para **B:**, **2** para **C:** e assim por diante, o argumento **setor** conta de 0 até o maior setor do volume existente e **número** refere-se ao número de setores copiados para a memória.

Quando o comando "**L**" é usado sem parâmetros, o arquivo especificado na linha de comando é carregado na memória, começando automaticamente na posição de deslocamento **0100** do segmento "**CS**" em uso desde que antecipadamente tenha sido usado o comando "**N**", se usado com o parâmetro **endereço** é possível carregar para a memória o arquivo ou o conteúdo dos setores especificados memória, se usado todos os parâmetros é possível carregar o conteúdo de setores específicos do disco em vez de carregar um arquivo.

Exemplo

Veja na figura A.23 o exemplo do carregamento do arquivo "**exemplo.txt**" a partir da execução do programa DEBUG sem a passagem deste arquivo como argumento do programa a partir da instrução "**C:> debug /f**".

Figura A.23 - Carregamento do arquivo "exemplo.txt".

Para o carregamento de setores de disco pode-se fazer uso de uma instrução de chamada semelhante a seguinte:

```
-l 0100 2 0 1
```

Em que o conteúdo solicitado é carregado no endereço **0100** do setor **0** da unidade **2** (disco **C:**) com a quantidade de setores definida com **1**. O comando "**L**" usa os valores dos registradores "**DX**" e "**CX**" para que nestes estejam definidos o tamanho dos dados carregados no formato de 32 bits. Caso a leitura não seja possível será apresentada a mensagem de erro "**Unknown command reading drive**" com a indicação da unidade de disco lida.

```
M faixa endereço – move (movimentar)
```

Este comando é usado para copiar o conteúdo de certo intervalo de endereço de memória para outro intervalo de endereço de memória dentro do limite de 64 KB operacionalizado pelo programa DEBUG. Antes de escrever certo conteúdo na área de destino o programa DEBUG armazena inicialmente os bytes da origem com o comando "**M**" para efetivar sua ação. É importante tomar cuidado de não se fazer a especificação de áreas de memória fora da abrangência de deslocamento dentro do segmento de memória em uso. Os parâmetros **faixa** (endereço de origem: local inicial e final) e **endereço** (local inicial de destino) são obrigatórios.

Exemplo

```
C:> debug /f exemplo.txt
```

Veja na figura A.24 o exemplo da efetivação de cópia do conteúdo da área de memória de **0100** até **0106** para o endereço **0200**.

```
-m 0100 0106 0200
-d 0100
06AF:0100  45 73 74 75 64 6F 20 65-20 64 65 6D 6F 6E 73 74 Estudo e demonst
06AF:0110  72 61 C3 A7 C3 A3 6F 20-64 6F 73 20 72 65 63 75 ra....o dos recu
06AF:0120  72 73 6F 73 20 65 78 69-73 74 65 6E 74 65 73 20 rsos existentes
06AF:0130  6E 6F 20 75 74 69 6C 69-74 C3 A1 72 69 6F 20 64 no utilit..rio d
06AF:0140  65 20 64 65 70 75 72 61-C3 A7 C3 A3 6F 2C 20 6D e depura....o, m
06AF:0150  6F 6E 74 61 67 65 6D 20-65 20 64 65 73 6D 6F 6E ontagem e desmon
06AF:0160  74 61 67 65 6D 20 22 45-6E 68 61 6E 63 65 64 20 tagem "Enhanced
06AF:0170  44 45 42 55 47 22 2E 00-00 00 00 00 00 00 00 00 DEBUG".........
-d 0200
06AF:0200  45 73 74 75 64 6F 20 00-00 00 00 00 00 00 00 00 Estudo .........
06AF:0210  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
06AF:0220  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
06AF:0230  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
06AF:0240  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
06AF:0250  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
06AF:0260  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
06AF:0270  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
-_
```

Figura A.23 - Movimentação (cópia) de bytes em memória.

```
N [[unidade:] [caminho]] arquivo [lista de argumentos] – name (nome)
```

Este comando é usado para estabelecer em memória o nome do arquivo a ser lido com o comando "**L**" ou escrito com o comando "**W**". O comando "**N**" usa um parâmetro obrigatório chamado **arquivo** que se refere ao nome de referência de um arquivo no sistema operacional, o qual poderá ou não fazer uso do parâmetro **lista de argumentos**. Além desses parâmetros há opcionalmente a indicação de **unidade** referente a unidade de disco a ser usada e **caminho** para indicar o local onde o arquivo será ou é referenciado.

Exemplo

```
C:> debug /f
```

Observe na figura A.25 o uso do comando "**N**" para o carregamento do arquivo "**exemplo.txt**" para a memória. Atente para os detalhes indicados na figura.

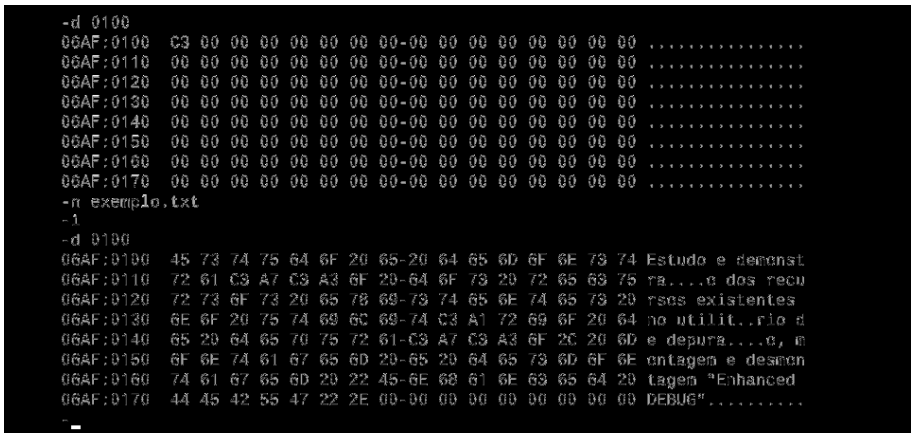

Figura A.25 - Carregamento do arquivo "exemplo.txt".

```
O [W|D] porta valor - output (porta de saída)
```

Este comando é usado para realizar o envio, ou seja a saída, de um valor a uma porta x86. O comando "**O**" usa dois parâmetros obrigatórios que devem ser indicados a partir de um número entre **0000** e **FFFF** para produzir um byte para uma porta de 16 bits, tamém especificada, entre os valores **0000** até **FFFF**. Os comandos "**OW**" e "**OD**" produzem valores no formato **word** ou **dword** (*double-word*).

```
P [=endereço] [numero] - proceed (continuar)
```

Este comando é usado para realizar a continuidade de execução de um trecho de programa que esteja sendo operado no modo passo a passo semelhantemente ao uso do comando "**T**" após o uso do comando "**G**". O comando "**P**" executa de uma só vez as instruções de um bloco de programa (sub-rotina CALL), diferentemente do comando "**T**" que faz esta ação linha a linha não importando o formato do programa. O comando "**P**" faz uso de dois parâmetros opcionais alternativamente: o parâmetro **=endereço** permite estabelecer de qual endereço de deslocamento o programa será executado; o parâmetro **número** especifica a quantidade de vezes que o comando "**P**" poderá ser executado.

```
PR - proceed return (prossiga retorno)
```

Este comando é usado para realizar um salto para fora da sub-rotina atual, passando pelo próximo RET, RETF ou IRET que não esteja aninhado sem parar. O comando "**PR**" não faz uso de nenhum parâmetro.

```
Q - quit (sair)
```

Este comando é usado para encerrar a execução do programa DEBUG e retornar o controle ao sistema operacional. O comando "**Q**" não faz uso de nenhum parâmetro.

```
R [registro [valor]] - register (registrador)
```

Este comando é usado para dar acesso aos registradores de memória tanto na apresentação de seus conteúdos como para alteração dos mesmos. O comando "**R**" pode trabalhar com um parâmetro opcional chamado **registro** que pode por sua vez fazer uso ou não de um parâmetro chamado **valor**. O comando "**R**" sem a definição de parâmetro apresenta o estado dos registros gerais atuais, exibidos como valores de 16 ou 32 bits (dependendo do uso do comando "**RX**"), e o conteúdo dos demais registradores. Se usado com o comando o nome de um registrador específico este apresentará o conteúdo do registrador e se usado o parâmetro com o nome do registrador e um valor fará a inserção do valor indicado no registrador apontado.

Exemplo

```
C:> debug /f
```

Observe na figura A.26 o uso do comando "**R**" a partir de algumas ocorrências.

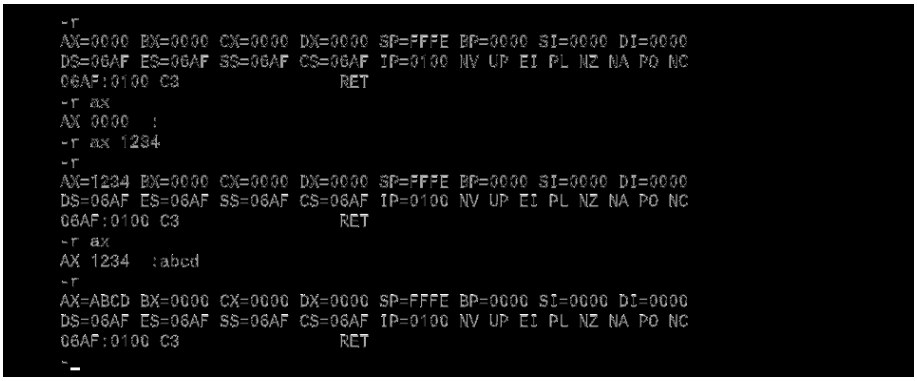

Figura A.26 - Resultados operacionais do uso do comando "R".

```
RF [lista de sinalizadores] - register flag (sinalizadores: registrador de estado)
```

Este comando é usado para apresentar ou modificar os valores dos sinalizadores do registrado de estado. O comando "**RF**" não é apresentado no modo ajuda com o uso do comando "**?**" mas é um dos comandos mais importantes do programa DEBUG. Este comando é operado apenas com valores de 16 bits e seu uso sem parâmetro apresenta a relação dos *flags* (sinalizadores) com a sinalização de seus bits em uma única linha e permite modificação dos valores de estado de cada bit ou de todos os bits do registrador. Os bits do registrador de estado que podem ser apresentados e/ou alterados são: **OF**, **DF**, **IF**, **SF**, **ZF**, **AF**, **PF**, **CF** que respectivamente poderão estar configurados com o valor "**1**" representados pelos rótulos **NV**, **UP**, **DI**, **PL**, **NZ**, **NA**, **PO** e **NC** ou configurados com o valor "**0**" com os rótulos **OV**, **DN**, **EI**, **NG**, **ZR**, **AC**, **PE** e **CY**. Veja resumidamente as definições desses valores no quadro e figura A.27.

Flag Register (Registrador de Estado)								
Flag	**OF**	**DF**	**IF**	**SF**	**ZF**	**AF**	**PF**	**CF**
0	*NV*	*UP*	*DI*	*PL*	*NZ*	*NA*	*PO*	*NC*
1	*OV*	*DN*	*EI*	*NG*	*ZR*	*AC*	*PE*	*CY*

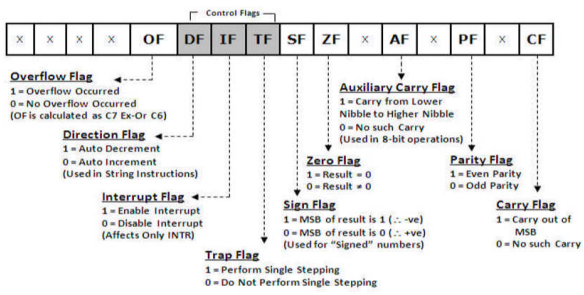

Figura A.27 - Flag Register: Robotic Eletronics

https://roboticelectronics.in/wp-content/cache/all/flag-register-in-8086/index.html

Exemplo

```
C:> debug /f
```

Observe na figura A.28 o uso do comando "**RF**" a partir de algumas ocorrências.

Figura A.28 - Resultados operacionais do uso do comando "RF".

A figura A.26 apresenta a alteração do estado do registrador **OF** e **IF** com seus valores alterados de **NV** (**0**) para **OV** (**1**) em **OF** e de **EI** (**1**) para **DI** (**0**) em **IF**.

As alterações de valores nos registradores de sinalização podem ser efetivadas com ou sem o uso de espaços em branco após o *prompt* DEBUG, como por exemplo "**ovei**" como mostra a figura A.29.

Figura A.29 - Alteração de registradores sem uso de espaço em branco entre os flags.

```
RN - FPU register (apresenta o conteúdo do registrador FPU)
```

Este comando é usado para apresentar o estado do registrador de FPU (Floating-Point Unit – Unidade de Ponto Flutuante) que permite ao computador fazer uso de valores de ponto flutuante (valore do conjunto de números reais). O registrador FPU possui um vetor de memória com oito posições que podem ser acessadas como uma pilha: a posição "**ST0**" refere-se ao registrador que está no topo da pilha e "**ST7**" refere-se ao registrador no final da pilha. Os valores numéricos de ponto flutuante operacionalizados no registrador FPU possui, para reduzir erros de arredondamento, o cumprimento de 80 bits. O comando "**RN**" não faz uso de nenhum parâmetro.

Exemplo

```
C:> debug /f
```

Observe na figura A.30 o uso do comando "**RN**" a partir de algumas ocorrências.

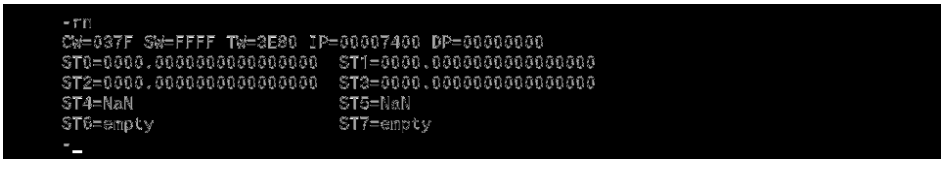

Figura A.30 - Apresentação conteúdo do registrador "FPU" com o comando "RN".

```
RX - toggle 386 regs (alternar registrador 386)
```

Este comando é usado para alternar os modos de operação do programa DEBUG entre os formatos 16/32 bits. O comando "**RX**" não faz uso de nenhum parâmetro.

Exemplo

```
C:> debug /f
```

Observe na figura A.31 o uso do comando "**RX**".

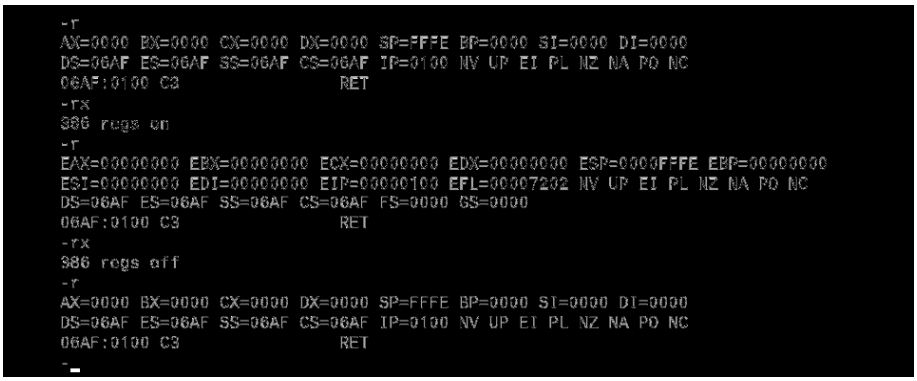

Figura A.31 - Alteração dos modos 16/32 bits.

```
S faixa lista - search (procurar)
```

Este comando é usado para efetuar na memória a pesquisa de uma cadeia de bytes. O comando "**S**" faz uso obrigatório de dois parâmetros, sendo **faixa** para especificar a área de memória a ser pesquisada e **lista** para determinar a cadeia de bytes a ser pesquisada. O comando pode resultar duas formas de saída: a primeira quando nada é encontrado e mostra apenas o próximo *prompt* DEBUG ou quando encontra o conteúdo pesquisado e mostra o endereço inicial de onde a sequência é iniciada.

Exemplo

```
C:> debug /f exemplo.txt
```

Observe na figura A.32 o uso do comando "**S**" com algumas variações.

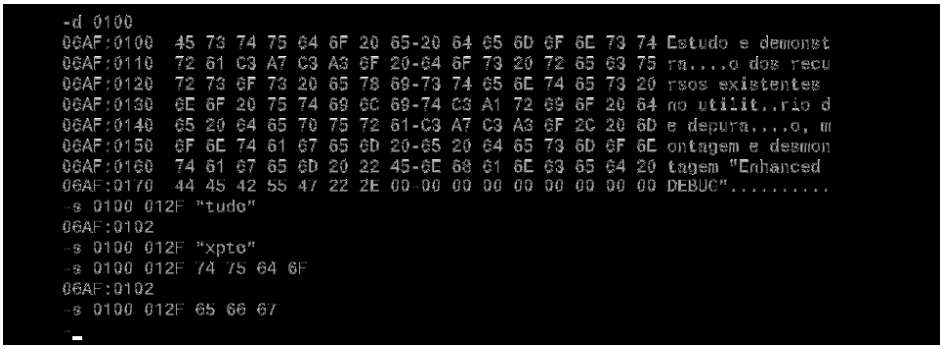

Figura A.32 - Alteração dos modos 16/32 bits.

```
T [=endereço] [número] - trace (rastrear)
```

Este comando é usado para rastrear as instruções de um programa disposto na memória, uma de cada vez, passo a passo. O comando "**T**" pode fazer uso de dois parâmetros opcionais, sendo **endereço** a indicação do ponto de origem que se deseja rastrear e **número** é a indicação de quantas vezes se deseja executar passo a passo o rastreamento. Se o comando "**T**" isoladamente rastreará apenas a instrução do código de programa que estiver apontada a partir dos valores definidos nos registradores "**CS**" e "**IP**" e fará a interrupção de sua ação apresentando os valores existentes nos registradores, além de indicar a próxima instrução a ser rastreada. Se usado com seus parâmetros permite definir o

ponto de início do rastreamento (apenas o primeiro parâmetro) ou com o segundo parâmetro realizar a partir do endereço apontado um certo número de repetições do comando.

Exemplo

```
C:> debug /f
```

Observe na figura A.33 o uso do comando "**T**". Atente para a definição de um código de programa que permite fazer uso do comando algumas vezes. Veja que a cada execução do comando "**T**" o registrador **IP** tem seu valor alterado apontando para a próxima linha a ser executada. A primeira execução do comando "**T**" salta o programa para a linha **011F** conforme indicado na instrução "**jmp 011F**" na linha **0100**. As duas próximas linhas representam os dados armazenados na memória e que são tratados no programa.

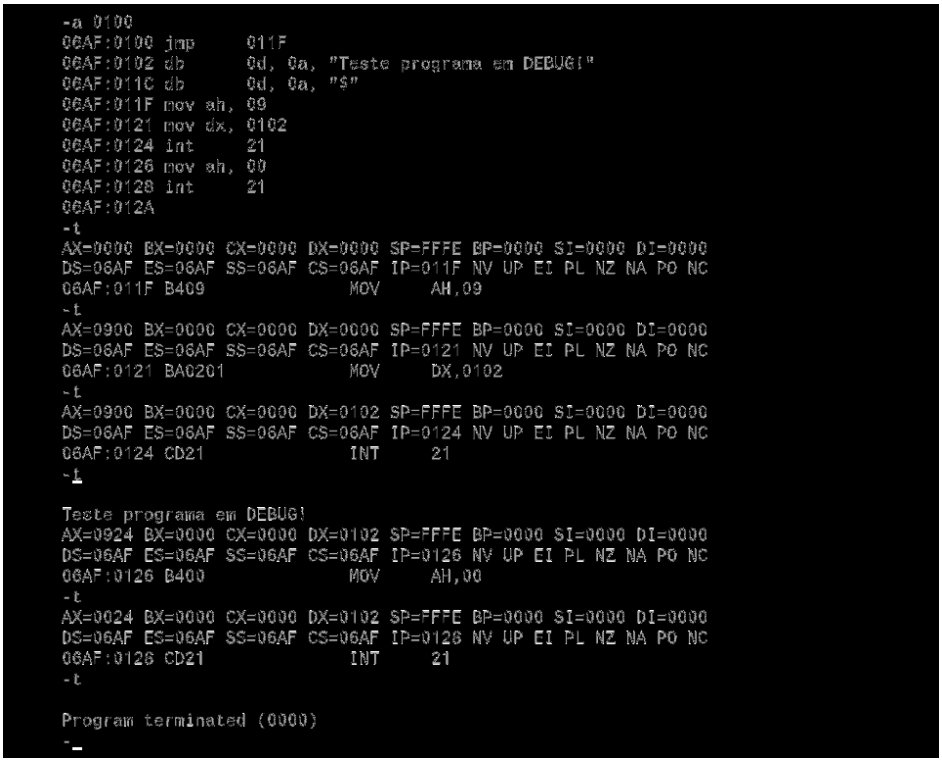

Figura A.33 - Execução passo a passo com comando "T".

Outra maneira de executar o programa e ver cada um dos conteúdos gerados na memória de uma única vez é por meio da instrução "**t -0100 5**".

```
TM [0|1] - trace mode (modo de rastreamento)
```

Este comando é usado para mostrar ou definir o modo de rastreamento do programa. São possíveis o uso de dois modos de operação com "**TM0**" e "**TM1**". Exceto a indicação dos modos de operação o comando "**TM**" não usa nenhum parâmetro. Os modos "**TM0**" e "**TM1**" afetam a forma como o comando "**T**" é operado. Se selecionado o modo "**TM0**" (modo padrão) executa rastreamento no estilo do programa *Enhanced DEBUG* e o modo "**TM1**" executa rastreamento no estilo *Microsoft DEBUG*.

```
U [faixa] - unassemble (desmontar)
```

Este comando é usado para desmontar as instruções de um programa na forma de código de máquina. O código desmontado tem a aparência semelhante a estrutura de um código montado. O comando "**U**" utiliza opcionalmente um

parâmetro chamado **faixa**. Caso o comando seja usado sem o parâmetro atuará automaticamente sobre o endereço de deslocamento **0100** desmontando 32 bytes, mas com a definição do parâmetro é possível definir o endereço inicial e final de atuação ou a definição do endereço inicial e seu cumprimento com o código "**L**".

Exemplo

```
C:> debug /f exemplo.txt
```

Observe na figura A.34 o uso do comando "**U**" sem a definição de parâmetro, na figura A.35 com definição de parâmetro de faixa inicial/cumprimento e na figura A.36 com definição de parâmetro de faixa inicial e final.

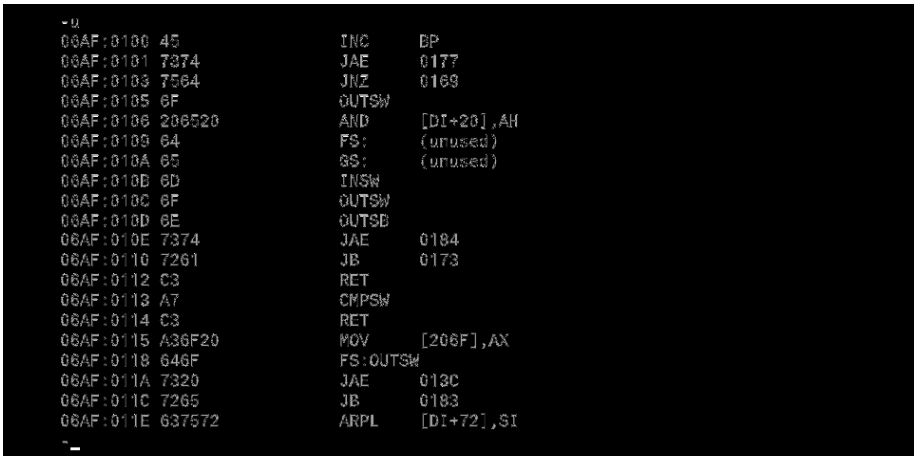

Figura A.34 - Execução do comando "U" sem parâmetro.

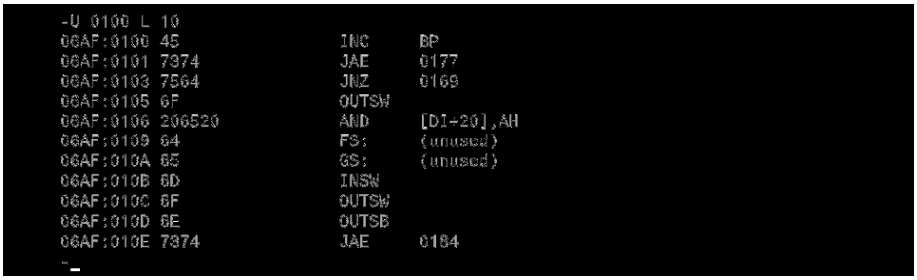

Figura A.35 - Execução do comando "U" com parâmetro de faixa inicial e cumprimento.

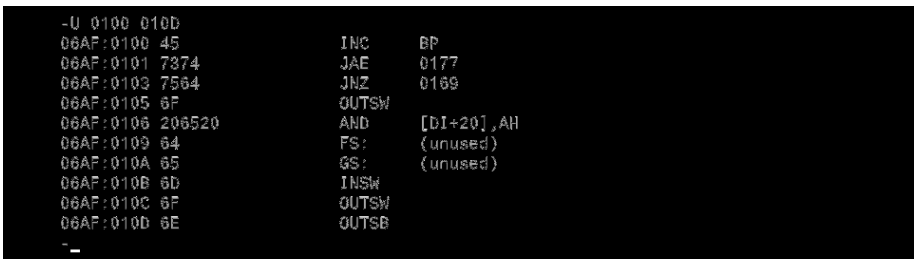

Figura A.36 - Execução do comando "U" com parâmetro de faixa inicial e final.

```
V - view flip (visualizar inversão)
```

Este comando quando em uso permite que a saída do programa ocorra na página de vídeo 0 (zero) quando o efeito de inversão de página no sistema é suportado. O comando "**V**" não possui nenhum parâmetro e seu efeito pode, dependendo de diversos fatores alheios ser imperceptível. Quando o argumento "**/f**" é usada no *prompt* do sistema na chamada do programa DEBUG o modo de operação em uso é a página 1 de vídeo, enquanto a saída do programa, geralmente, ocorrerá na página 0 de vídeo. Dentre os comandos existentes este é um que pode não parecer muito útil.

```
W [[endereço] [unidade setor número]] - write file/sectors (escrever arquivo/setores)
```

Este comando escreve em disco um arquivo ou setores específicos que estejam em memória. O comando "**W**" pode ser usado com ou sem parâmetros. Para usar o comando sem parâmetro é necessário que o arquivo a ser gravado tenha sido carregado inicialmente com o programa DEBUG ou que se utilize antes o comando "**N**". Caso haja a necessidade de gravar um conteúdo de memória é necessário considerar o número de bytes a serem gravados em um arquivo de disco definidos junto aos registradores gerais "**BX**" e "**CX**". O parâmetro **endereço** especifica o ponto inicial de memória a ser considerado na gravação, o parâmetro **unidade** especifica a unidade de disco usada para a gravação, o parâmetro **setor** especifica o endereço do primeiro setor a ser gravado e o parâmetro **número** especifica a quantidade de setores a ser usada na gravação. Para fazer uso do comando "**W**" é necessário tomar certos cuidados: caso tenha ocorrido antes de executar o comando "**W**" o uso dos comandos "**G**", "**T**", "**P**" e "**R**" é necessário redefinir, primeiro, os registradores gerais "**BX**" e "**CX**"; não use o comando para gravar arquivos nos formatos "**.EXE**" e "**.HEX**".

Exemplo

`C:> debug /f exemplo.txt`

Observe na figura A.37 o uso do comando "**W**" sem a definição de parâmetro.

```
-d 0100 0105
06AF:0100  45 73 74 75 64 6F      -                       Estudo
-e 0101 6e,73,61,69
-d 0100 0105
06AF:0100  45 6E 73 61 69 6F      -                       Ensaio
-w
Writing 0077 bytes
-q

D:\>debug
-n exemplo.txt
-l
-d 0100 0105
06AF:0100  45 6E 73 61 69 6F      -                       Ensaio
-_
```

Figura A.37 - Execução do comando "W" sem parâmetro após alteração de arquivo.

Observe na figura A.38 o uso do comando "**W**" com a definição de parâmetro gravando o conteúdo de memória do arquivo "**exemplo.txt**" em um arquivo chamado "**teste.txt**" a partir do endereço "**CS:0100**". A figura A.39 mostra o resultado da gravação no arquivo "**TESTE**".

```
D:\>debug exemplo.txt
-d 0100
06AF:0100  45 6E 73 61 69 6F 20 65-20 64 65 6D 6F 6E 73 74 Ensaio e demonst
06AF:0110  72 61 C3 A7 C3 A3 6F 20-64 6F 73 20 72 65 63 75 ra....o dos recu
06AF:0120  72 73 6F 73 20 65 78 69-73 74 65 6E 74 65 73 20 rsos existentes
06AF:0130  6E 6F 20 75 74 69 6C 69-74 C3 A1 72 69 6F 20 64 no utilit..rio d
06AF:0140  65 20 64 65 70 75 72 61-C3 A7 C3 A3 6F 2C 20 6D e depura....o, m
06AF:0150  6F 6E 74 61 67 65 6D 20-65 20 64 65 73 6D 6F 6E ontagem e desmon
06AF:0160  74 61 67 65 6D 20 22 45-6E 68 61 6E 63 65 64 20 tagem "Enhanced
06AF:0170  44 45 42 55 47 22 2E 00-00 00 00 00 00 00 00 00 DEBUG".........
-n teste.txt
-w cs:0100
Writing 0077 bytes
-_
```

Figura A.38 - Execução do comando "W" com parâmetro no endereço "CS:0100".

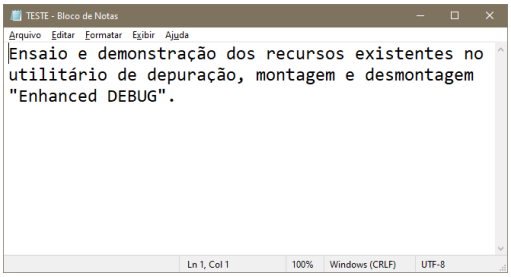

Figura A.39 - Arquivo gerado após o uso da instrução "w CS:0100".

O comando "**W**" pode ser usado de forma mais completa como "**w cs:0100 2 3A 3F**" e neste caso far-se-á a gravação dos dados em memória iniciando-se no endereço "**cs:0100**" na unidade de disco "**C:**" começando no setor lógico "**3A**" por "**3F**" setores.