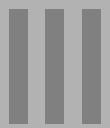


0D82:0100	B402	MOV	AH, 02
0D82:0102	B241	MOV	DL, 41
0D82:0104	CD21	INT	21
0D82:0106	CD20	INT	20
0D82:0108	69	DB	69
0D82:0109	64	DB	64
0D82:010A	61	DB	61
0D82:010B	64	DB	64
0D82:010C	65	DB	65
0D82:010D	206573	AND	[DI+73], AH
0D82:0110	7065	JO	0177
0D82:0112	63	DB	63
0D82:0113	69	DB	69
0D82:0114	66	DB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

Parte



Programação com emu8086

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

Capítulo

7

OPERAÇÕES ESSENCIAIS

Este capítulo mostra recursos e restrições da ferramenta de montagem e simulação Assembly **emu8086** (versão 4.08r). Traz informações sobre o processo de compilação de programas, execução (depuração) passo a passo do código de programa e o conjunto de instruções que formam a linguagem de programação de computadores Assembly 8086/8088..

7.1 Simulador e assembler 8086

O programa **Enhanced DEBUG** anteriormente demonstrado permite a construção de programas tanto em linguagem de máquina, por meio de *opcode* (códigos de operação) como em linguagem *Assembly*. No entanto, é uma ferramenta que possui algumas restrições operacionais, sendo uma delas exigir que se saiba em que parte da memória está se construindo os programas.

É fato que existem outras ferramentas que facilitam o trabalho de codificação e compilação dos programas, tais como: **TASM** (*Turbo Assembler da Borland*) e **MASM** (*Macro Assembler da Microsoft*) que são ferramentas comerciais, além de ferramentas baseadas em código livre como é o caso do programa **FASM** (*Flat Assembler - http://flat assembler.net*) e **NASM** (*The Netwide Assembler - http://www.nasm.us/*).

Pelo fato deste livro ser focado a leitores iniciantes cabe apresentar uma ferramenta com apelo mais didático e que facilite o entendimento deste tipo de programação, chamado **emu8086**. O programa **emu8086** é um simulador gráfico do processador 8086/8088 padrão shareware que também compila programas escritos em linguagem *Assembly*. Além de sua praticidade, possui boa documentação e pode ser adquirido a um baixo custo considerando o potencial que a ferramenta oferece e sua execução nos sistemas operacionais Microsoft Windows de 32 e 64 bits.

Efetue o carregamento para a memória do computador do programa **emu8086** como orientado no capítulo 3. Selecione na caixa de diálogo **welcome** o botão **New**. Na caixa de diálogo **choose code template** mantenha a seleção da opção **COM template** e acione o botão **OK**. A Figura 7.1 apresenta a tela inicial do programa.

Observe atentamente detalhes que já são conhecidos e estão presentes no trecho de código apresentado:

- ◆ Na linha **05** está a definição da diretiva¹ **org 100h** que estabelece o critério de criação de um programa simples com extensão **.COM**. A diretiva está grafada em tom azul-marinho e o valor associado em tom preto. A diretiva **org** é usada para marcar a posição inicial de carga do código objeto (WEBER, 2004, p. 239). O valor **100** é colocado no registrador de segmento **CS** quando o programa for executado. Anteriormente o acesso a esse endereço foi realizado por meio da instrução **A 0100** no programa **Enhanced DEBUG**.
- ◆ A linha **09** indica o uso do comando **RET** (ou **ret**) que tem por finalidade retornar o controle de execução do programa ao sistema operacional após seu encerramento.

Observação

A instrução **org 100h** pode ser substituída sem nenhum problema pela instrução **org 0x100**. O indicativo **0x** possui o mesmo efeito do indicativo **h**. A forma **org 0x100** será explorada neste texto.

¹ Uma diretiva é um comando usado para controlar a montagem de um programa objeto (WEBER, 2004, p. 239). É uma maneira de se passar ao programa montador opções que o programador deseja executar na ação de montagem de seu programa.

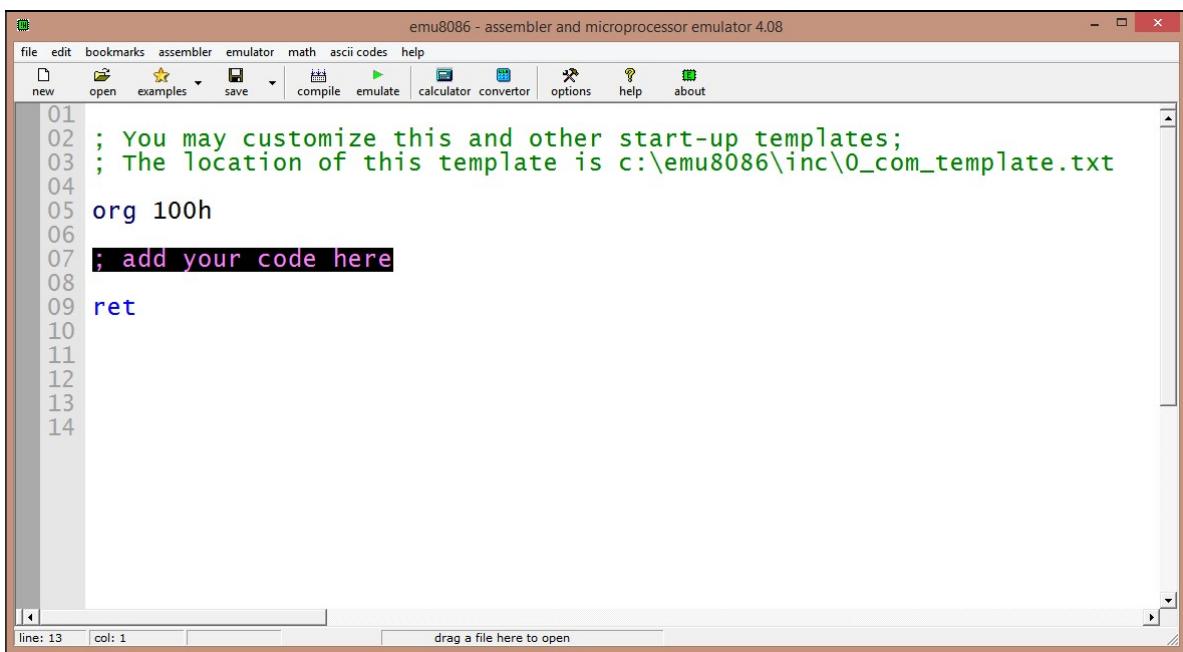


Figura 7.1 - Site da ferramenta emu8086.

O programa **emu8086** permite de maneira simples, a visualização gráfica de toda a estrutura da arquitetura de um microprocessador padrão Intel 8086. É possível visualizar diversos elementos de composição de um microprocessador como: *stack*, *alu*, *memória*, *variáveis*, *flags* entre outras possibilidades. Por ser esta ferramenta um simulador há algumas poucas instruções assembly 8086 que não são executadas no ambiente, mas nada que comprometa o entendimento e aprendizagem.

7.2 - Programa “Alo Mundo”

Em seguida será desenvolvido um pequeno programa que apresenta a mensagem “**Alo mundo**”, na tela do monitor de vídeo como o programa já apresentado no capítulo 5. Observe o código a seguir similar ao código que foi utilizado anteriormente no programa **Enhanced DEBUG**:

```
0D0B:0100 MOV AH,09
0D0B:0102 MOV DX,0200
0D0B:0105 INT 21
0D0B:0107 INT 20

0D0B:0200 41 6C 6F 20 6D 75 6E 64 6F 21 24
```

Observe em seguida o mesmo programa codificado no estilo utilizado pela ferramenta **emu8086**. Note a existência de algumas pequenas diferenças entre os dois códigos. Assim sendo, a partir da linha **04** escreva o programa seguinte:

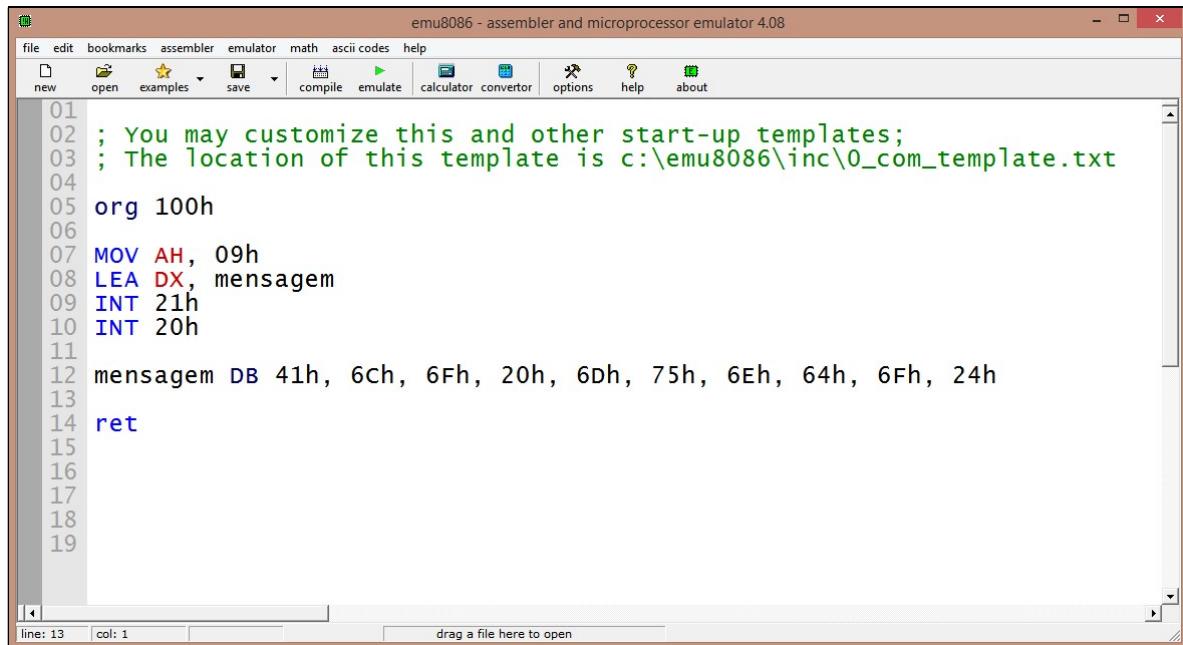
```
MOV AH, 09h
LEA DX, mensagem
INT 21h
INT 20h

mensagem DB 41h, 6Ch, 6Fh, 20h, 6Dh, 75h, 6Eh, 64h, 6Fh, 24h
```

As instruções da linguagem de programação de computadores *Assembly* podem ser escritas tanto com letras maiúsculas como minúsculas. Assim sendo, torna-se necessário definir um padrão de trabalho. Nesta obra as instruções sempre são escritas com caracteres maiúsculos.

A Figura 7.2 apresenta o código do programa na tela do editor do programa **emu8086**. Note a similaridade existente entre as duas formas de codificação. No primeiro programa a sequência de códigos hexadecimais da mensagem está

definida no endereço de memória **0200h**. Já no segundo programa a sequência de códigos hexadecimais para a escrita da mensagem está definida na variável **mensagem** (que será criada em algum endereço de memória desconhecido) por meio da diretiva **DB** (que será explanada no próximo capítulo). Outro detalhe do segundo programa é o uso da instrução **LEA DX, mensagem** no lugar da instrução **MOV DX,0200**. A instrução **MOV DX,0200** funciona no programa Enhanced DEBUG, mas não pode ser usada da mesma forma no programa **emu8086**, no qual essa operação é realizada pelo comando **LEA**.



```

emu8086 - assembler and microprocessor emulator 4.08
file edit bookmarks assembler emulator math ascii codes help
new open examples save compile emulate calculator convertor options help about
01
02 ; You may customize this and other start-up templates;
03 ; The location of this template is c:\emu8086\inc\0_com_template.txt
04
05 org 100h
06
07 MOV AH, 09h
08 LEA DX, mensagem
09 INT 21h
10 INT 20h
11
12 mensagem DB 41h, 6ch, 6Fh, 20h, 6Dh, 75h, 6Eh, 64h, 6Fh, 24h
13
14 ret
15
16
17
18
19

```

The screenshot shows the emu8086 software interface with the assembly code for "Hello World". The menu bar includes File, Edit, Bookmarks, Assembler, Emulator, Math, ASCII Codes, and Help. The toolbar has icons for New, Open, Examples, Save, Compile, Emulate, Calculator, Convertor, Options, Help, and About. The code window displays the assembly instructions from line 01 to 19. Line 01 is a comment. Lines 02-03 are another comment. Line 04 sets the origin to 100h. Lines 05-10 implement the DOS interrupt 21h to output the string. Line 12 defines the string "mensagem" with the values 41h, 6ch, 6Fh, 20h, 6Dh, 75h, 6Eh, 64h, 6Fh, 24h. Line 14 returns control to the operating system.

Figura 7.2 - Programa Alô mundo.

O comando **LEA** funciona com dois parâmetros: um **registrador** (representado por **DX**) e o **endereço** de memória (representado pela variável **mensagem**). Esta instrução não afeta nenhum registrador de estado e ocupa de 2 a 4 bytes de memória. No caso apresentado, o parâmetro *registrador* está definido pelo uso do registrador geral **DX** de 16 bits, e só é possível usar o comando **LEA** com registradores gerais que sejam de 16 bits; o parâmetro *endereço* representado pelo rótulo de identificação da variável **mensagem** indica o local da memória onde se encontra armazenada os caracteres da mensagem a ser apresentada.

O comando **LEA** obtém de forma automática o valor do deslocamento de endereço de memória a partir do local efetivo apontado, ou seja, o comando **LEA** calcula o valor efetivo do deslocamento de endereço a ser utilizado, ou seja, carrega em um registrador do tipo *word* (neste caso **DX**) o deslocamento de um operando (neste caso **mensagem**). Assim, o comando **LEA** carrega no **registrador** o endereço onde se encontra definido o conteúdo apontado em **endereço**. Apesar dessa facilidade, essa instrução é mais lenta em sua execução que a execução do comando **MOV**. Em seguida, grave o programa na pasta **Documentos** usando o comando de menu **file/save** com o nome **TESTEMSG1**.

Para fazer uso do comando **MOV** no lugar do comando **LEA** definido na linha 05 deve-se em seu lugar usar a instrução **MOV DX, OFFSET mensagem**, em que o registrador geral **DX** é carregado com o valor do deslocamento (instrução **OFFSET**) de endereço da posição de memória onde se encontra a variável **mensagem**. É como pedir para o comando **MOV** ir até o local da memória onde se encontra a variável **mensagem** por meio do comando **OFFSET**.

O comando **OFFSET** é uma diretiva que obtém o endereço relativo, ou seja, o deslocamento da variável indicada no segmento. O comando **MOV DX, OFFSET mensagem** esta armazenando no registrador **DX** o offset do endereço onde a variável **mensagem** se encontra.

Observe a seguir o código do programa com essa alteração e também sua forma de apresentação na tela do programa **emu8086**, Figura 7.3. Grave o programa na pasta **Documentos** com o nome **TESTEMSG2** usando para tanto o comando de menu **file/save as...** .

```

01 ; You may customize this and other start-up templates;
02 ; The location of this template is c:\emu8086\inc\0_com_template.txt
03
04 org 100h
05
06 MOV AH, 09h
07 MOV DX, OFFSET mensagem
08 INT 21h
09 INT 20h
10
11 mensagem DB 41h, 6ch, 6Fh, 20h, 6dh, 75h, 6Eh, 64h, 6Fh, 24h
12
13
14 ret
15
16
17
18
19

```

Figura 7.3 - Programa Alô mundo com MOV DX, OFFSET mensagem.

Você sabia que o comando **MOV**:

- ◆ não afeta nenhum dos registradores de estado (*flags*);
- ◆ que as movimentações entre a origem e o destino devem ser do mesmo tipo de dado;
- ◆ que o registrador de apontamento **IP** não pode ser usado;
- ◆ que o registrador de segmento **CS** não pode ser usado como destino, mas pode ser definido como origem;
- ◆ que nenhum operador de segmento pode receber um dado de forma imediata;
- ◆ que as definições de origem e destino não podem ser ambos indicados como memória.

O comando **MOV** pode ser utilizado para efetuar movimentação:

- ◆ de dado imediato para a memória;
- ◆ de dado imediato para registrador;
- ◆ de registrador para registrador;
- ◆ de memória/registrador para registrador/memória;
- ◆ de memória ou registrador para registrador de segmento;
- ◆ de registrador de segmento para memória ou registrador;
- ◆ de acumulador/memória para memória/acumulador.

Assim sendo, observe alguns exemplos de utilização do comando **MOV**.

- ◆ De dado imediato para memória:

MOV byte ptr [0800h], ABh - movimenta o word **ABh** para o deslocamento de memória **0800h**. A indicação **byte ptr** refere-se ao uso de um byte na memória;

MOV word ptr [AX+ABCDh], CS - movimenta o word de **CS** para o deslocamento de memória **[AX+ABCDh]**. A indicação **word ptr** refere-se ao uso de um word na memória.

- ◆ De dado imediato para registrador:

MOV AX, 1357h - movimenta o valor hexadecimal **1357** para o registrador geral **AX**;

MOV CX, 10101010b - movimenta o valor binário **10101010** para o registrador geral **CX**.

- ◆ De registrador para registrador:
MOV AX, BX - movimenta o valor do registrador **BX** para o registrador **AX**;
MOV SI, AX - movimenta o valor do registrador **AX** para o registrador **SI**.
- ◆ De memória/registrador para registrador/memória:
MOV AL, [BX] - movimenta o conteúdo armazenado no deslocamento **[BX]** para o registrador **AL**;
MOV AL, [BX + SI] - movimenta o conteúdo armazenado no deslocamento **[BX + SI]** para o registrador **AL**.
- ◆ De memória ou registrador para registrador de segmento:
MOV ES, AX - movimenta o valor do registrador **AX** para o registrador **ES**;
MOV DS, [SI] - movimenta o conteúdo do deslocamento **[SI]** para o registrador **DS**.
- ◆ De registrador de segmento para memória ou registrador:
MOV AX, CS - movimenta o valor do registrador **CS** para o registrador **AX**;
MOV [0700h], DS - movimenta o conteúdo do registrador **DS** para o deslocamento de memória **0700h**.
- ◆ De acumulador/memória para memória/acumulador:
MOV [SI], AH - movimenta o byte armazenado no registrador **AH** para o deslocamento **[SI]**;
MOV AH, [BX+04h] - movimenta o conteúdo do deslocamento **[BX+04h]** para o registrador **AH**;
MOV DESLOC, AL - movimenta o valor do registrador **AL** para o deslocamento identificado pelo rótulo **DESLOC**;
MOV AL, DESLOC - movimenta o conteúdo do deslocamento identificado pelo rótulo **DESLOC** para o registrador **AL**.

Outro detalhe a ser observado é que no programa **emu8086** (como também é comum em outros programas montadores) não é necessário preocupar-se com o local onde a sequência de caracteres deve ser armazenada na memória. Esse trabalho fica a cargo da ferramenta de montagem (do programa Assembler).

A partir deste momento o programa pode ser compilado dentro da ferramenta **emu8086** ou apenas executado passo a passo. Se for compilado, será criado, neste caso, o arquivo de programa com a extensão **.COM**. Se for executado passo a passo, é possível acompanhar a execução de cada linha e também o processo de execução das instruções.

Abra o programa **TESTEMSG1** com o comando de menu **file/open**. Essa ação fecha o programa aberto, no caso o programa **TESTEMSG2** e coloca em uso o programa carregado. Atente para a instrução **MOV AH, 09h** definida na linha **04**, como foi indicado na Figura 6.6. Essa instrução está movimentando o valor **09h** para o registrador **AH** (parte mais alta do registrador geral **AX**). O código **09h** é usado quando se deseja apresentar um *string* armazenado no registrador geral **DX** (como na instrução da linha **05** que movimenta para o registrador geral **DX** o endereço de memória no qual se encontra o conteúdo da variável **mensagem** por meio da instrução **LEA DX, mensagem**). A partir do uso da interrupção **21h**, comandada pela linha **06** por meio da instrução **INT 21h**, ocorre a apresentação da mensagem no monitor de vídeo.

A interrupção **21h** é um recurso que estabelece o controle do serviço de entrada e saída operacionalizado pelo MS-DOS (entenda-se sistema operacional). A definição da ação de uma entrada ou da ação de uma saída dessa interrupção depende do valor armazenado sempre no registrador **AH**. Alguns códigos válidos para operacionalizar a interrupção **21h** são:

- ◆ **AH = 01h** - efetua a leitura com eco de um caractere a partir do periférico padrão para a entrada de dados conectado ao computador, representado pelo teclado. O caractere informado é armazenado no registrador **AL**. Se não houver nenhum caractere armazenado no *buffer* do teclado, a função espera até que alguma tecla seja acionada.
- ◆ **AH = 02h** - faz a escrita de um caractere a partir do periférico padrão para a saída de dados conectado ao computador, representado pelo monitor de vídeo. O caractere a ser escrito é armazenado no registrador **DL** após a execução de **AL = DL**.

- ◆ **AH = 05h** - efetua a escrita de um caractere a partir do periférico padrão para a saída de dados conectado ao computador, representado pela impressora. O caractere a ser escrito é armazenado no registrador **DL** após a execução de **AL = DL**.
- ◆ **AH = 06h** - efetua a entrada ou a saída a partir do teclado ou monitor de vídeo conforme o caso. Para efetivar uma saída, o registrador **DL** deve possuir um valor entre **00h..FEh** (em decimal 0..254) que representa o código ASCII do caractere a ser utilizado em **AL = DL** quando do retorno desta ação. Para efetivar a entrada, o registrador **DL** deve possuir o valor **FFh** (em decimal 255) e neste caso a entrada retorna o registrador **ZF** setado em 1 caso não se tenha nenhum caractere disponível no buffer do teclado e o registrador **AL** esteja com valor **00h**. O registrador **ZF** fica com seu valor em 0 quando existir algum caractere armazenado no buffer do teclado. Neste caso o registrador **AL** conterá a entrada e o buffer de teclado será limpo.
- ◆ **AH = 07h** - realiza a leitura sem eco de um caractere a partir do periférico padrão para a entrada de dados conectado ao computador, representado pelo teclado. O caractere informado é armazenado no registrador **AL**. Se não houver nenhum caractere armazenado no buffer do teclado, a função espera até que alguma tecla seja acionada.
- ◆ **AH = 09h** - efetua a escrita de uma sequência de caracteres a partir do periférico padrão para a saída de dados conectado ao computador, representado pela impressora. O caractere a ser escrito é armazenado no registrador **DS:DX**.
- ◆ **AH = 0Ah** - faz a leitura de uma sequência de caracteres (*string*) armazenando no registrador **DS:DX**. O primeiro byte representa o tamanho do buffer e o segundo byte representa o número de caracteres efetivamente lidos. Essa função não coloca ao final do *string* o caractere \$, necessário para a identificação do final de uma sequência de caracteres. Para fazer a impressão dessa entrada com **INT 21** e **AH = 09h**, é necessário acrescentar ao final da sequência de caracteres o caractere \$ e iniciar a apresentação a partir do endereço de memória **DS:DX+2**.
- ◆ **AH = 2Ah** - obtém a data do sistema, tendo como retorno o ano de 1980 até 2099 no registrador **CX**, o mês no registrador **DH**, o dia no registrador **DL**, o dia da semana no registrador **AL**, sendo o valor **00h** o domingo, o valor **01h** a segunda-feira e assim por diante.
- ◆ **AH = 2Ch** - obtém a hora do sistema, tendo como retorno a hora no registrador **CH**, o minuto no registrador **CL**, o segundo no registrador **DH** e o centésimo de minuto no registrador **DL:DX**.

Além dos valores apresentados, há também outras funções usadas com a interrupção **INT 21**.

A interrupção **INT 20** finaliza o programa, retornando o controle ao sistema operacional.

Com o objetivo de demonstrar o real uso da ferramenta **emu8086**, ou seja, executar um programa de baixo nível em modo passo a passo, acione o comando de menu **assembler/compile and load in emulator**, ou utilize o botão **emulat-e** da barra de ferramentas do programa, ou ainda utilize a tecla de função **<F5>**. A Figura 7.4 mostra um exemplo da tela de depuração do programa **original source code** com a janela **emulator: TESTEMSG1.asm_** sobre a janela de edição do programa.

Nesta etapa do processo utilize a tecla de função **<F8>** para ver a execução de cada linha do programa. A primeira linha do programa indicada é a de código **MOV AH, 09h**. Neste momento, acione a tecla **<F8>** e observe que a linha de código **LEA DX, mensagem**, Figura 7.5, da janela **original source code** é indicada como **MOV DX, 00109h** em operação na janela **emulator: TESTEMSG1.com_**, Figura 7.6.

Note que na janela **emulator: TESTEMSG1.com_**, Figura 7.6, a execução do comando **LEA DX, mensagem** é indicada como sendo **MOV DX, 00109h** (semelhante ao que ocorreu no programa **Enhanced DEBUG**). Essa janela apresenta três áreas de trabalho importantes:

- ◆ A parte à esquerda assinalada como **registers** (abaixo do botão **Load**) mostra o estado dos registradores e como estes estão se comportando à medida que é possível avançar o programa passo a passo por meio da tecla de função **<F8>**.
- ◆ A parte central exibe a execução do programa em linguagem de máquina por meio da indicação do endereço real de memória, que são os valores numéricos hexadecimais com cinco dígitos definidos na primeira coluna (na Figura 7.5 esse recurso está assinalado nas linhas **07100** e **07101**), os valores em hexadecimal na segunda coluna referente ao código de operação (Opcode) em execução (indicados na Figura 7.5 por **B4** e **09**, em que o valor hexadecimal **09** será movimentado para o registrador **AH**, sendo o valor hexadecimal **B4** o Opcode referente à ação da instrução **MOV AH**).

- ◆ A parte à direita mostra a execução linha a linha do programa em linguagem Assembly.

A Figura 7.6 apresenta algumas informações. Na área da direita é indicada a execução da instrução **MOV DX, 00109h** referente ao código-fonte **LEA DX, mensagem**. As áreas do centro e da direita da tela **emulate: TESTEMSG1.com_** são sinalizadas na parte superior das áreas central e da direita com os valores das posições de memória no formato **segmento:deslocamento**. O endereço físico **00109h** é o local onde se inicia o *string* da mensagem a ser apresentada.

Na sequência pressione a tecla <F8> três vezes e observe a mensagem “**Alo mundo**”, como indica a Figura 7.7. Continue acionando a tecla de função <F8> até que a mensagem de término seja apresentada, como na Figura 7.8. Para finalizar o processo por completo, acione o botão **OK**. Depois feche a janela **emulator screen (80 x 25 chars)** com o botão X no canto superior direito de sua barra de título.

Na janela **emulator: TESTEMSG1.com_** acione o comando de menu **file/close the emulator** para voltar ao editor de código-fonte do programa.

```

edit: C:\Users\Augusto Manzano\Documents\TESTEMSG1.asm
file edit bookmarks assembler emulator math ascii codes help
new open examples save compile emulate calculator convertor options help about
01 ; You may customize this and other start-up templates;
02 ; The location of this template is c:\emu8086\inc\0_com_template.txt
03
04 org 100h
05
06
07 MOV AH, 09h
08 LEA DX, mensagem
09 INT 21h
10 INT 20h
11
12 mensagem DB 41h, 6ch, 6Fh, 20h,
13
14 ret
15
16
17
18
19

```

emulator: TESTEMSG1.com_

Registers	0700:0100	0700:0100
AX	00 00	07100: B4 180
BX	00 00	07101: 09 009
CX	00 14	07102: BA 186
DX	00 00	07103: 09 009
CS	0700	07104: 01 001
IP	0100	07105: CD 205
SS	0700	07106: 21 033
SP	FFFE	07107: CD 205
BP	0000	07108: 20 032
SI	0000	07109: 41 065
DI	0000	0710A: 6C 108
DS	0700	0710B: 6F 111
ES	0700	0710C: 60 109

step delay ms: 0

Load reload step back single step run

original source code

drag a file here to open

Figura 7.4 - Programa Alô mundo em execução.

```

01 ; You may customize this and other start-up templates;
02 ; The location of this template is c:\emu8086\inc\0_com_template.txt
03
04 org 100h
05
06
07 MOV AH, 09h
08 LEA DX, mensagem
09 INT 21h
10 INT 20h
11
12 mensagem DB 41h, 6ch, 6Fh, 20h,
13
14 ret
15
16
17
18
19

```

Figura 7.5 - Original source code.

Registers	0700:0100	0700:0100
AX	00 00	07100: B4 180
BX	00 00	07101: 09 009
CX	00 14	07102: BA 186
DX	00 00	07103: 09 009
CS	0700	07104: 01 001
IP	0100	07105: CD 205
SS	0700	07106: 21 033
SP	FFFE	07107: CD 205
BP	0000	07108: 20 032
SI	0000	07109: 41 065
DI	0000	0710A: 6C 108
DS	0700	0710B: 6F 111
ES	0700	0710C: 60 109

step delay ms: 0

Load reload step back single step run

Figura 7.6 - Janela emulator: TESTEMSG1.com_.

O programa atual será agora gravado com outro nome para compilá-lo e assim gerar o arquivo executável. Execute o comando de menu **file/save as...** e informe o nome **MENSAGEM1** para que seja gravado o arquivo de programa com o nome **MENSAGEM1.asm**. Depois execute o comando de menu **assembler/compile** que apresenta a caixa de diálogo **Salvar como**. Neste momento acione o botão **Salvar** para gravar o programa **MENSAGEM1.com** na pasta **Documentos** e observe a apresentação da caixa de diálogo **compiler status**, Figura 7.9.

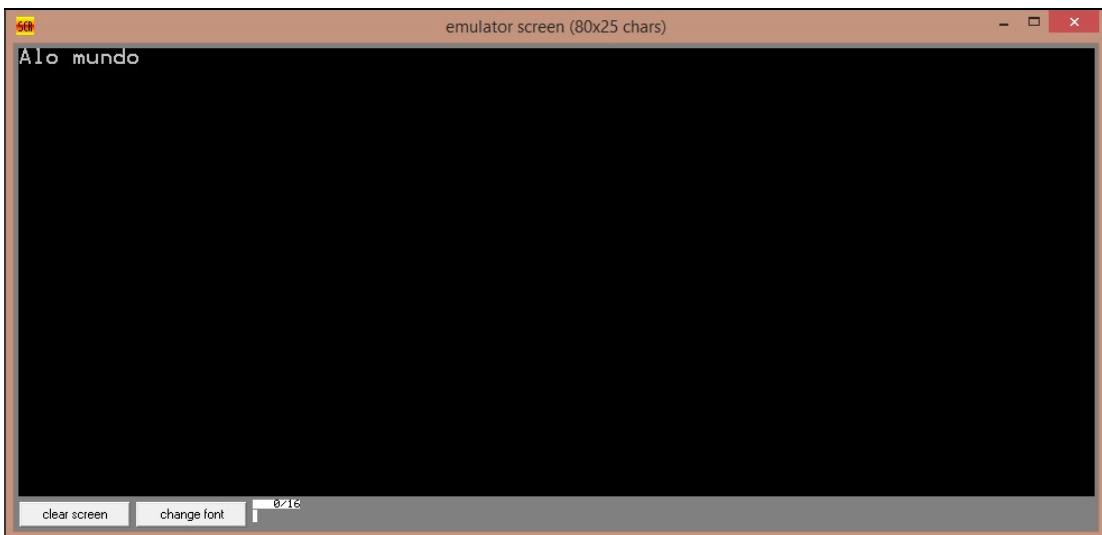


Figura 7.7 - Apresentação da saída do programa.

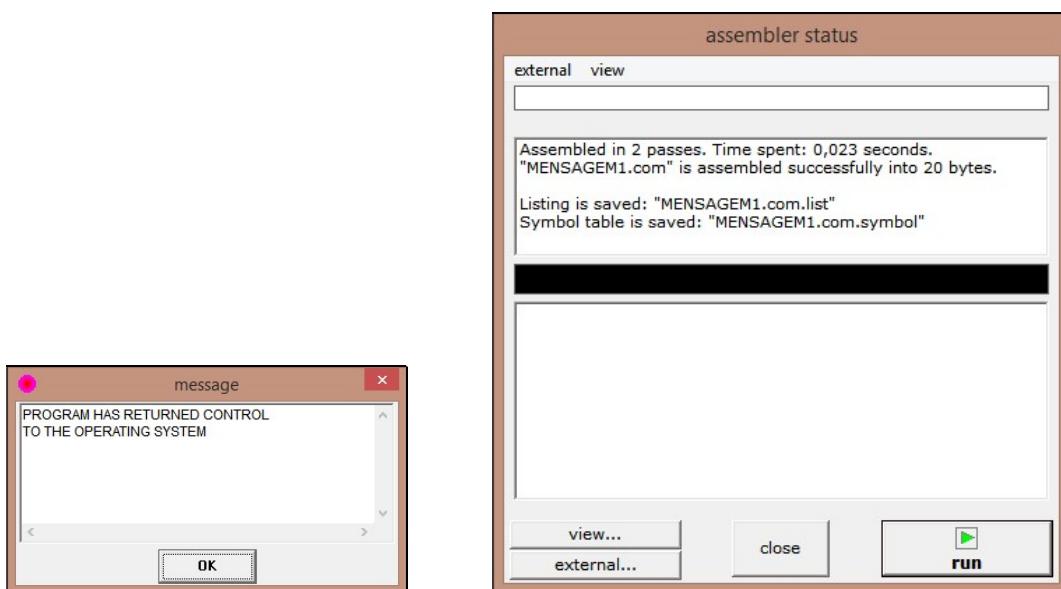


Figura 7.8 - Término da execução do programa.

Figura 7.9 - Caixa de diálogo Assembler status. Compiler Status.

A caixa de diálogo traz algumas informações a respeito do processo de compilação do programa. Em seguida, acione o botão **close** para fechar a caixa de diálogo **compiler status** e retornar ao editor.

O programa anterior desenvolvido na ferramenta **emu8086** fez uso de um pequeno conjunto de instruções similares as mesmas instruções usadas no programa **Enhanced DEBUG**. Neste conjunto de instruções foi utilizada certa estrutura sintática na escrita das instruções em código de linguagem **Assembly** para microprocessadores Intel (ou AMD) destinados a computadores pertencentes a família IBM-PC. Cabe neste ponto da apresentação da linguagem **Assembly 8086/8088** descrever a estrutura de escrita de suas instruções que obedecem a seguinte forma:

[rótulo:] mnemônico [argumento1][, argumento2][, argumento3] [;comentário]

onde

- ◆ **rótulo** é um identificador opcional seguido de dois pontos usado para indicar principalmente nas ferramentas de **assembler** pontos de retorno quando do uso de instruções de desvios.
- ◆ **mnemônico** é o uso da instrução **assembly** para a definição de alguma ação pretendida.

- ◆ **argumento1**, **argumento2** ou **argumento3** são operandos opcionais usados de zero até três argumentos dependendo do código operacional (*opcode*) em uso para a definição de valores ou literais.
- ◆ **Comentário** é a definição de uma linha explicativa opcional da instrução sendo implementada.

Quando são usados dois operandos em uma instrução aritmética ou lógica, o operando posicionado a esquerda representa o destino e o operando da direita representa a origem. Por exemplo:

```
carrega_registrador: MOV AX, 50h ; move o valor hexa 50 no registrador AX
```

A linha de instrução identificada com o rótulo **carrega_registrador**: efetua a movimentação com o mnemônico **MOV** do valor **50h** para o registrador **AX**. O comentário que faz uma explicação sobre a ação da instrução está definida após o uso do símbolo de ponto-e-vírgula.

O programa **emu8086** possui um simulador do programa **DEBUG** original da Microsoft que pode ser utilizado quando um programa está em execução passo a passo. Assim sendo, carregue para a memória o programa **TESTEMSG1.asm** gravado na pastas **Documentos** a partir do uso do comando de menu **file/open...**. Acione no menu principal do programa o comando **assembler/compile and load in emulator**.

Considerando apenas o uso da janela **emulator: TESTEMSG1.com** apresentada selecionada na parte inferior o botão **debug** para ser apresentada a janela **debug log - debug.exe emulation** como indicado na Figura 7.10.

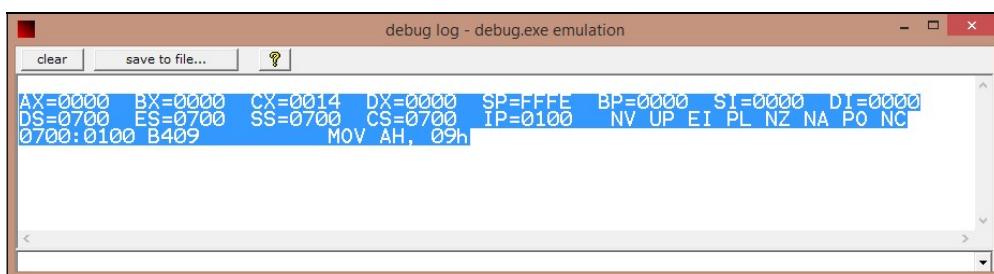


Figura 7.10 - Janela debug log – debug.exe emulation.

A janela **debug log - debug.exe emulation** é formada por três áreas operacionais, sendo: uma pequena barra de ação com os botões: **clear** (limpa o conteúdo da área de display apresentado no simulador de **debug**), **save to file...** (salva o conteúdo da área de display do emulador em um arquivo texto com o mesmo nome do programa em operação) e **interrogação** (mostra o modo de ajuda com os comandos aceitos pelo simulador de **debug**); uma área de display contendo a apresentação do estado dos registradores internos de um microprocessador padrão 8086; a última área é definida pelo campo de comando existente abaixo da área de display que permite a entrada de alguns comandos similares aos existentes nas várias versões dos programas **DEBUG**.

Os comandos aceitos pelo simulador **DEBUG** na terceira área de acesso são similares aos comandos dos programas **DEBUG** estando disponível um pequeno conjunto deles, destacando-se apenas os comandos: **C** (*compare*), **I** (*input*), **O** (*output*), **D** (*dump*), **T** (*trace*), **P** (*proceed*) e **R** (*register*). Desse pequeno conjunto os comandos conhecidos anteriormente são **D**, **T**, **P** e **R**.

Para um pequeno teste acione o botão **clear** na área de ação da janela para limpar a área de display e entre na área do campo de comando o código de ação **R** para ver os registradores de um microprocessador padrão 8086. As figuras 7.11 e 7.12 mostram respectivamente o uso do comando **R** e do resultado apresentado.

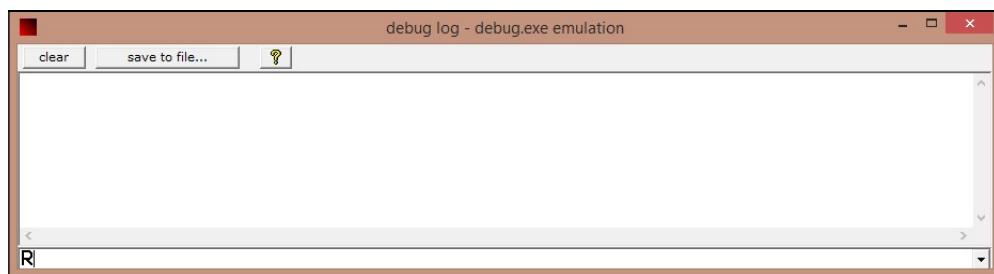


Figura 7.11 - Janela debug log com definição de uso do comando “R”.

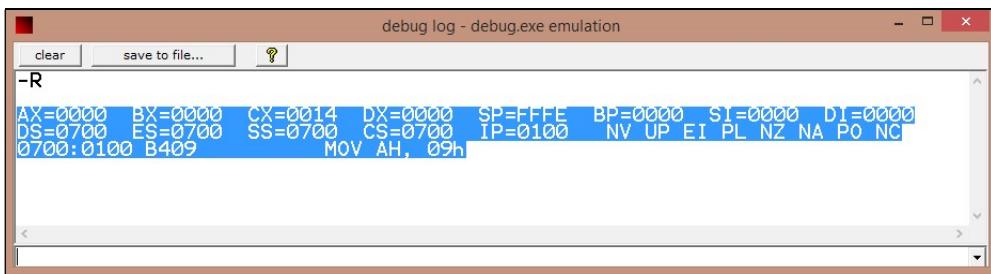


Figura 7.12 - Janela debug log com resultado apresentado após comando “R”.

Na janela **debug log - debug.exe emulation** é possível executar passo a passo o programa da memória de duas formas: uma utilizando-se o botão **single run** existente na janela **emulator: TESTEMSG1.com_**; outra se utilizando o código de ação **P** definido no campo de comando apresentado na janela **debug log - debug.exe emulation**.

Neste momento, acione uma vez o botão **single run** da janela **emulator: TESTEMSG1.com_** e note a execução da linha de instrução **MOV AH, 09h** para a linha de instrução **MOV DX, 00109h** como mostra a Figura 7.13.

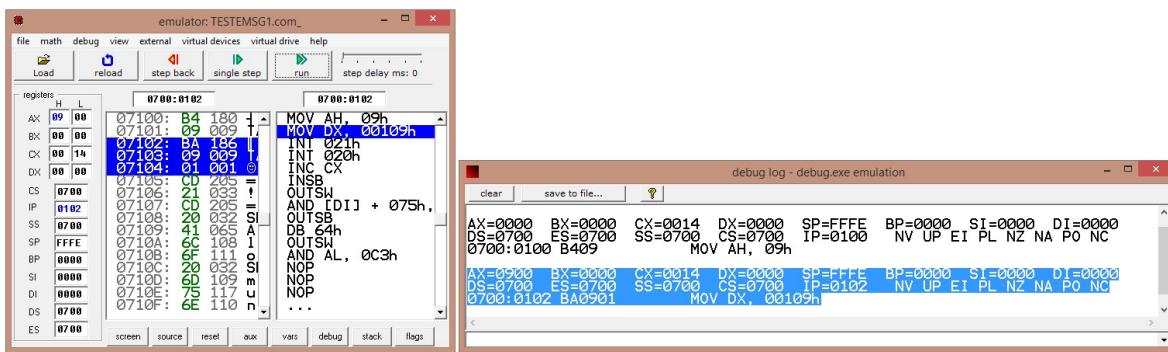


Figura 7.13 - Ação passo a passo após instrução **MOV AH, 09h**.

A próxima ação de execução para a instrução **MOV DX, 00109h** será efetuada com o uso do código de ação **P** definido no campo de comando da janela **debug log - debug.exe emulation**. Assim sendo, entre o comando **P** e acione a tecla **<Enter>**. Ao fazer esta ação o efeito ocorre tanto na janela **debug log - debug.exe emulation** como na janela **emulator: TESTEMSG1.com_** como pode ser constatado na figura 7.14.

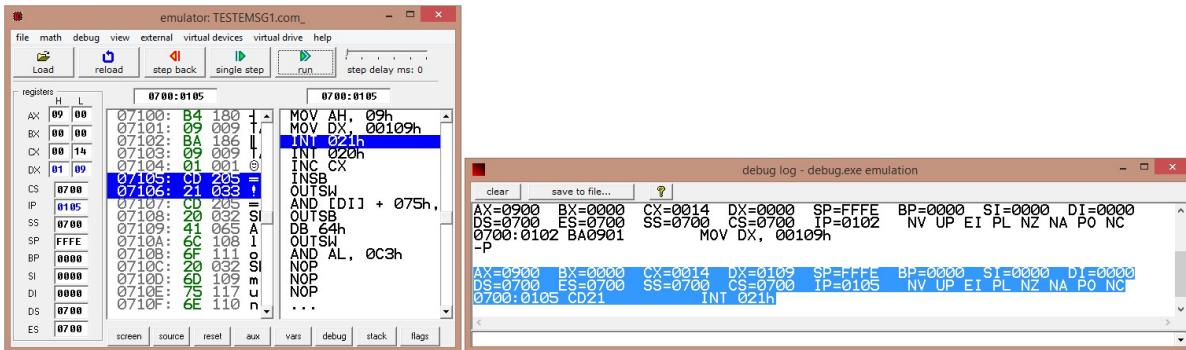
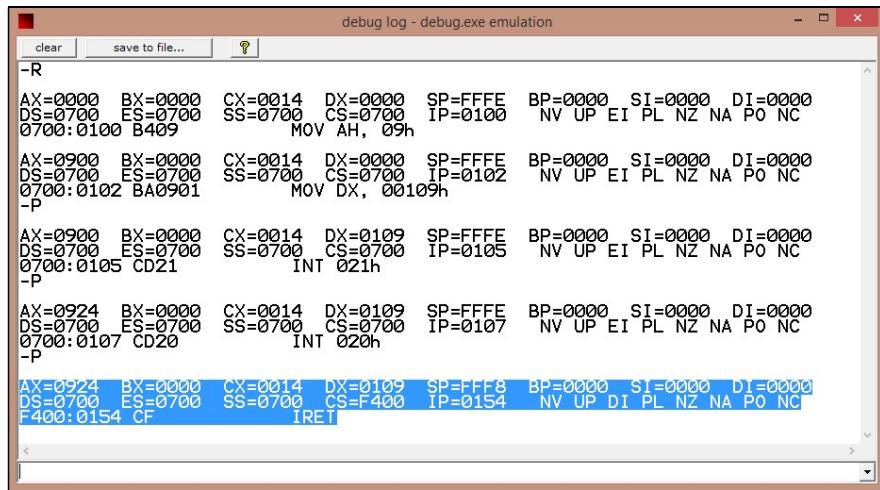


Figura 7.14 - Ação passo a passo após instrução **MOV DX, 00109h**.

A partir das duas ações anteriormente executadas percebe-se que a execução passo a passo de um programa pode ser realizada tanto pelo uso do botão **single run** como com o uso do comando **P**. A única diferença percebida é que quando a ação é efetivada pelo botão **single run** da janela **emulator: TESTEMSG1.com_** a apresentação do resultado na janela **debug log - debug.exe emulation** ocorre sequencialmente sem a indicação do uso, por exemplo, do comando **P**, como pode ser percebido junto a Figura 7.14 ao se indicar na janela **debug log - debug.exe emulation** o indicativo **-P**.

Como continuidade da ação de execução do programa efetue mais duas vezes o uso do código de ação **P** no campo de comando da janela **debug log - debug.exe emulation** para concluir a ação do programa. Atente para a apresentação

da tela de saída do programa com a resposta da operação na janela **emulator screen (80x25chars)** e a indicação de encerramento do programa com a janela **message** de forma similar as imagens das Figuras 7.7 e 7.8. A Figura 7.15 mostra a janela **debug log - debug.exe emulation** com todas as ações executadas.



```

debug log - debug.exe emulation
clear | save to file... | ?
-R
AX=0000 BX=0000 CX=0014 DX=0000 SP=FFFFE BP=0000 SI=0000 DI=0000
DS=0700 ES=0700 SS=0700 CS=0700 IP=0100 NV UP EI PL NZ NA PO NC
0700:0100 B409 MOV AH, 09h

AX=0900 BX=0000 CX=0014 DX=0000 SP=FFFFE BP=0000 SI=0000 DI=0000
DS=0700 ES=0700 SS=0700 CS=0700 IP=0102 NV UP EI PL NZ NA PO NC
0700:0102 BA0901 MOV DX, 00109h
-P

AX=0900 BX=0000 CX=0014 DX=0109 SP=FFFFE BP=0000 SI=0000 DI=0000
DS=0700 ES=0700 SS=0700 CS=0700 IP=0105 NV UP EI PL NZ NA PO NC
0700:0105 CD21 INT 021h
-P

AX=0924 BX=0000 CX=0014 DX=0109 SP=FFFFE BP=0000 SI=0000 DI=0000
DS=0700 ES=0700 SS=0700 CS=0700 IP=0107 NV UP EI PL NZ NA PO NC
0700:0107 CD20 INT 020h
-P

AX=0924 BX=0000 CX=0014 DX=0109 SP=FFF8 BP=0000 SI=0000 DI=0000
DS=0700 ES=0700 SS=0700 CS=F400 IP=0154 NV UP DI PL NZ NA PO NC
F400:0154 CF IRET

```

Figura 7.15 - Janela debug log - debug.exe emulation com todas as ações executadas.

Após a apresentação da caixa de diálogo **message** acione o botão **OK** e junto a janela **emulator**: TESTEMSG1.com_ selecione o comando **close the emulator** do menu **file** para voltar a tela de edição do programa **emu8086**.

Além dos recursos básicos já apresentados, o programa **emu8086** tem uma série de outros como um conversor de bases numéricas, que pode ser acionado pelo comando de menu **math/base converter** ou pelo botão **converter** da barra de ferramentas, Figura 7.16, e uma calculadora de expressões aritméticas que pode ser acionada pelo comando de menu **math/multi base calculator** ou pelo botão **Calculator** da barra de ferramentas, Figura 7.17.



Figura 7.16 - Conversor numérico.

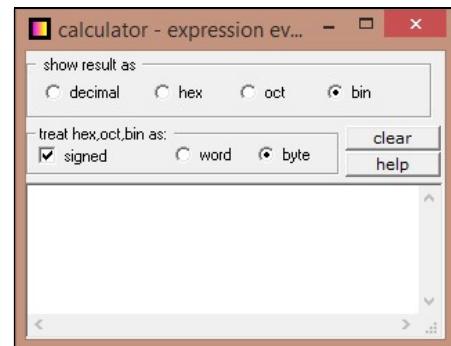


Figura 7.17 - Calculadora de expressões aritméticas.

A maior parte dos recursos do programa **emu8086** está disponível quando se executa um programa com a tecla de função **<F5>**, quando é apresentada a tela do programa **emulator**. Entre os vários recursos da janela **emulator** estão disponíveis no menu **view** boa parcela deles, tais como **extended value viewer** (Figura 7.18), **stack** (Figura 7.19), **variables** (Figura 7.20), **memory** (Figura 7.21), **arithmetic & logical unit/ALU** (Figura 7.22), **flags**, Figura 7.23, e **lexical flag analyser**, Figura 7.24. Os recursos **actual source** e **user screen** são exibidos automaticamente quando se usa o modo de execução do programa.

A Figura 7.19 mostra a tela **extended value viewer** com os valores atuais armazenados e existentes em um determinado registrador. Neste sentido, é possível visualizar o valor nas formas hexadecimal, binária, octal, decimal de 8 e 16 bits com ou sem sinal.

A Figura 7.20 apresenta a tela **stack** com os valores existentes na pilha. Observe o caractere **<** indicando o topo da pilha.

A Figura 7.21 exibe a tela **variables** com a lista de variáveis em uso no programa, bem como os valores iniciais encontrados nas variáveis relacionadas.

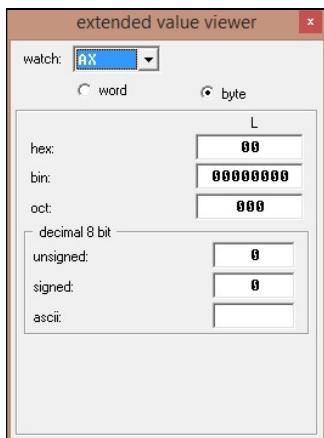


Figura 7.18 - Extended value viewer.

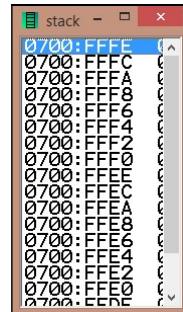


Figura 7.19 - Stack.

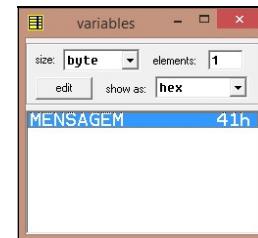


Figura 7.20 - Variables.

A Figura 7.21 mostra a tela **Memory** com a lista de valores externos armazenados nos endereços de memória a partir das coordenadas de segmento e deslocamento definidas na memória RAM (*Random Access Memory*).

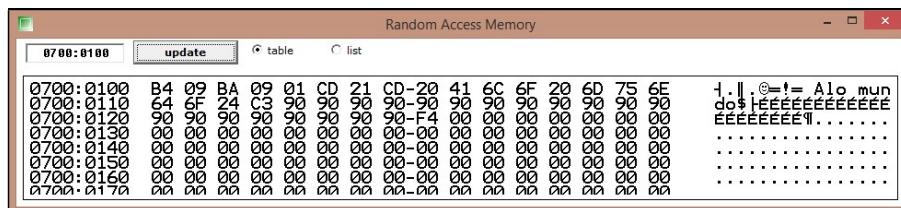


Figura 7.21 - Memory.

Na Figura 7.22 a tela **ALU - arithmetic & logical unit** mostra os dados existentes nas unidades lógica e aritmética.

Na Figura 7.23 a tela **flags** indica os valores do registrador de estado. Com o acionamento do botão **analyse** é apresentada a tela **lexical flag analyser**, Figura 7.24.

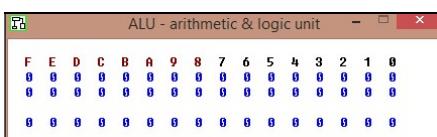


Figura 7.22 - ALU - arithmetic & logical unit.

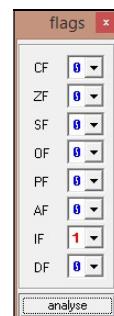


Figura 7.23 - Flags.

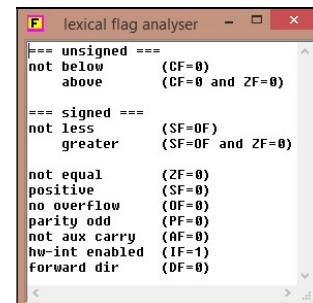


Figura 7.24 - Lexical flag analyser.

Na Figura 7.24 a tela **lexical flag analyser** indica na forma escrita os valores e o conteúdo de cada *flag* do registrador de estado.

7.3 - De volta ao programa Enhanced DEBUG

A partir da criação do programa **MENSAGEM1.com** é interessante visualizar seu código Assembly escrito no programa **emu8086** dentro do programa **Enhanced DEBUG**. Assim sendo, copie o arquivo **MENSAGEM1.com** da pasta **Documentos** para a pasta **DEBUGX** e efetue a chamada do ambiente **vDos**, mantendo o programa **emu8086** carregado e execute os seguintes comandos:

```
C:\>E: <Enter>
E:\>DEBUGX MENSAG~1.COM <Enter>
```

A indicação **MENSAG~1.COM** refere-se ao programa **MENSAGEM1.com**. A forma de referência **MENSAG~1** ocorre pelo fato do programa **vDos** reconhecer apenas e até os oito primeiros caracteres do nome de uma arquivo em modo MS-DOS, como o nome do programa **MENSAGEM1** possui mais de oito caracteres este é renomeado internamente e automaticamente pelo programa **vDos** como sendo **MENSAG~1**.

Assim que o programa **vDos** é carregado na memória juntamente do programa **MENSAG~1.COM** execute o comando a seguir que tem por finalidade decompilar o programa e apresentar seu código originalmente escrito ou um código perto da forma que o código original foi escrito:

```
U 0100 0112 <Enter>
```

Em seguida é apresentado o código do programa definido em memória, que pode ser visualizado a partir de duas partes.

A primeira parte representa o trecho do código em negrito do programa em si definido entre os endereços **0100** e **0107**:

07EF:0100 B409	MOV AH,09
07EF:0102 BA0901	MOV DX,0109
07EF:0105 CD21	INT 21
07EF:0107 CD20	INT 20

A segunda parte representa o trecho do código em negrito que representa a mensagem a ser escrita definida entre os endereços **0109** e **0112** (efetivamente no endereço **0111**):

07EF:0109 41	INC CX
07EF:010A 6C	INSB
07EF:010B 6F	OUTSW
07EF:010C 206D75	AND [DI+75],CH
07EF:010F 6E	OUTSB
07EF:0110 646F	FS:OUTSW
07EF:0112 24C3	AND AL,C3

Atente para o trecho grafado em negrito (coluna de *opcodes*), o qual indica os códigos ASCII em valores hexadecimais dos caracteres que compõem a mensagem a ser apresentada “**Alo mundo!**” definidas entre os endereços **0109** e **0111** que está anexo abaixo da última linha do trecho do código de apresentação da mensagem, identificado entre os endereços **0100** e **0107**.

Quando se utiliza uma ferramenta de montagem, um “*assemblador*”, não é necessário se preocupar com o ajuste de endereço de deslocamento inicial, pois os programas montadores fazem esse controle automaticamente como é o caso de uso da variável **mensagem** no programa escrito no programa **emu8086** acessado pela instrução **LEA DX, mensagem**. No entanto, programas como o **Enhanced DEBUG** necessitam que este trabalho seja efetuado manualmente, o que acaba exigindo certo grau de atenção e cuidado como mostra o uso do endereço **0109** acessado pela instrução **MOV DX, 0109**.

Na sequência encerre a execução dos programas **Enhanced DEBUG** e **vDos** e de volta ao programa **emu8086** acione a tecla de função **<F5>**. Na janela **emulator: TESTEMSG1.com** selecione no menu **view** a opção **listing** e será apresentada um relatório detalhado do código do programa grafado em *opcodes* e *assembly* como mostra a Figura 7.25.

O histórico apresentado mostra detalhes do código do programa em uso. Neste relatório há a indicação do número de linha usada no programa na coluna **[LINE]**, o local de memória onde o código do programa é montado identificado pela coluna **LOC:**, o código de máquina na forma de *opcode* indicado na coluna **MACHINE CODE** e o código escrito em *Assembly* identificada pela coluna **SOURCE**.

Além das informações apresentadas no relatório de um programa verificado no programa **emu8086** o relatório indica a data e hora em que o relatório foi processado.

```

TESTEMSG1.com_list - Bloco de notas
Arquivo Editar Formatar Egíbir Ajuda
EMU8086 GENERATED LISTING. MACHINE CODE <- SOURCE.
TESTEMSG1.com_ -- emu8086 assembler version: 4.08
[ 20/10/2018 -- 17:48:32 ]

=====
[LINE] LOC: MACHINE CODE SOURCE
=====

[ 1]   :
[ 2]   :
[ 3]   :
[ 4]   :
[ 5]   :
[ 6]   :
[ 7] 0100: B4 09 ; You may customize this and other start-up templates;
[ 8] 0102: BA 09 01 ; The location of this template is c:\emu8086\inc\0_com_template.txt
[ 9] 0105: CD 21 INT 21h
[10] 0107: CD 20 INT 20h
[11]   :
[12] 0109: 41 6C 6F 20 6D 75 6E 64 6F 24 mensagem DB 41h, 6Ch, 6Fh, 20h, 6Dh, 75h, 6Eh, 64h, 6Fh, 24h
[13]   :
[14] 0113: C3 ret

```

Figura 7.25 - Listagem de código de um programa em opcode e assembly.

O relatório apresentado com o comando **view/listing** da janela **emulator** é uma maneira de se ter em mãos um paralelo entre as formas de escrita de um programa em baixo nível codificado em linguagem de máquina e linguagem de montagem.

Um detalhe a ser observado junto ao relatório apresentado é a indicação da instrução da linha 8 onde o código de máquina **BA 09 01** equivalente a instrução em linguagem de montagem **LEA DX, mensagem** mostra que a variável **mensagem** se encontra definida no endereço de memória **0901**. No entanto, é pertinente salientar que a variável **mensagem** definida na linha 12 como **mensagem DB 41h, 6Ch, 6Fh, 20h, 6Dh, 75h, 6Eh, 64h, 6Fh, 24h** é indicada no trecho descrito em linguagem de máquina como **0109: 41 6C 6F 20 6D 75 6E 64 6F 24**. Isto posto, mostra que o endereço de memória **0901** indicado na linha 8 é identificado na listagem do programa na linha 12 como endereço **0109**.

Note que os valores são a mesma informação, mas mostrados de forma invertida **0901 x 0109**. Isto decorre da forma como o endereçamento de dados na memória é gerenciado pelo microprocessador. Observe que o endereço de memória **0109** onde se encontra a mensagem a ser apresentada requer para ser armazenado dois bytes de memória, sendo no byte mais significativo do registrador **DX (DH)** armazenado o valor **09** e no byte menos significativo do registrador **DX (DL)** armazenado o valor **01**.

O microprocessador necessita “conversar” com a memória para nela fazer o armazenamento de dados. Esta ação ocorre de forma reversa, isto é, um byte de baixa ordem é armazenado em certo endereço de memória e o outro byte de alta ordem é armazenado no próximo endereço de memória. Desta forma, para que o microprocessador possa operar o valor **0109** do registrador **DX** na memória os faz transferindo primeiro o valor **09** para certo endereço de memória e em seguida transfere o valor **01** para o próximo endereço de memória, dai vem o significado da indicação dos valores **0901** da linha 8 estar definido na linha 12 como o valor **0109**. Assim que o microprocessador obtém os dados do endereço de memória a ser utilizado no registrador os inverte novamente.

7.4 - Depuração com uso da pilha

Para um teste com o uso da pilha, considere um programa com uso de chamadas de procedimento semelhante a um exemplo utilizado anteriormente quando do uso do programa **Enhanced DEBUG**.

No programa **emu8086** execute o comando de menu **file/new/com template**. A partir da linha **07** informe a sequência de código seguinte de forma que fique semelhante à Figura 7.26, mantenha a instrução **RET** do código seguinte e retire a instrução **ret** apresentada automaticamente no programa:

```

MOV AX, 000Ah
MOV BX, 000Bh
PUSH AX
PUSH BX
CALL procedimento
INC AX

```

```

INC BX
CALL procedimento
POP BX
POP AX
INT 20h

procedimento: MOV AX, 0001h
              MOV BX, 0002h
              INC AX
              INC BX
              RET

```

Em seguida grave o programa com o nome **ROTINA** e acione a tecla de função <F5> e na medida em que a janela **original source code** vá sendo apresentada feche-a sucessivamente. Por meio da janela **emulator: ROTINA.com_**, execute os comandos de menu **view/log and debug.exe emulation** e **view/stack** que respectivamente apresentarão as janelas **debug log - debug.com emulator** como mostra a Figura 7.27 e **stack** como mostra a Figura 7.28 (que mostra esta janela ajustada com seu tamanho para a direita e para baixo).

The screenshot shows the emu8086 assembly editor window. The title bar reads "edit: C:\Users\Augusto Manzano\Documents\ROTINA.asm". The menu bar includes "file", "edit", "bookmarks", "assembler", "emulator", "math", "ascii codes", and "help". Below the menu is a toolbar with icons for "new", "open", "examples", "save", "compile", "emulate", "calculator", "converter", "options", "help", and "about". The main text area contains the assembly code for ROTINA.asm, with line numbers from 01 to 24. The code includes instructions like INC, CALL, MOV, POP, INT, and a procedure definition. The code is color-coded for syntax highlighting. At the bottom of the editor, there are status bars for "line: 24" and "col: 1", and a message "drag a file here to open".

Figura 7.26 - Programa escrito no ambiente de programação emu8086.

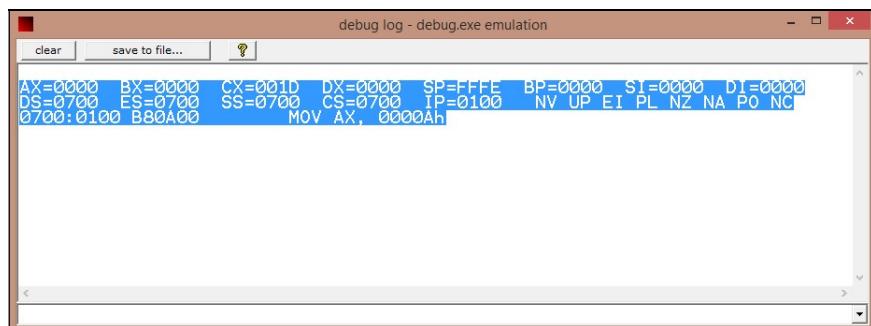


Figura 7.27 - Visualização do modo log and debug.exe emulation.

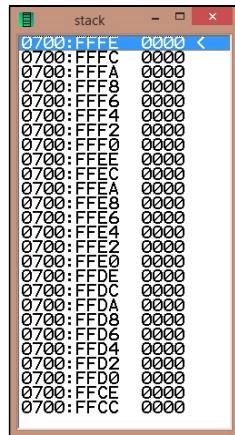


Figura 7.28 - Visualização do modo stack.

Na sequência vá acionando a tecla <F8> e observe atentamente as informações sendo apresentadas nas janelas apresentadas. Inicialmente a Figura 7.29 mostra as janelas **emulator: ROTINA.com_**, **debug log - debug.com emulator** e **stack** em conjunto antes da execução de qualquer ação.

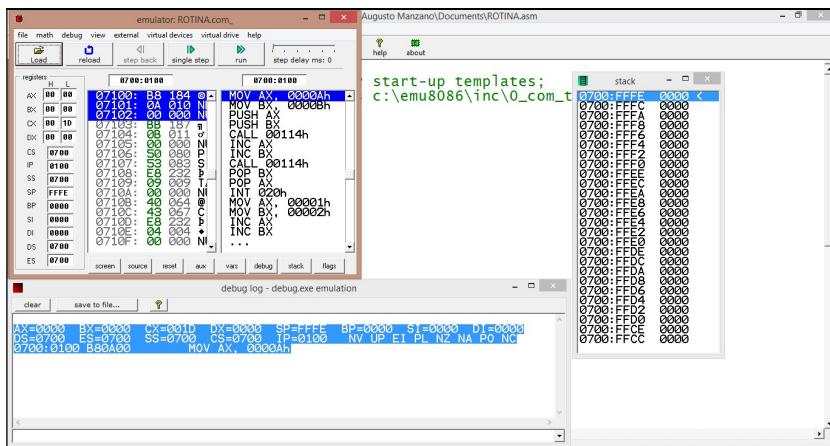


Figura 7.29 - Janelas de ação passo a passo do programa ROTINA.

A primeira ação a ser executada é a movimentação do valor **000A** para o registrador **AX**. Assim sendo acione a tecla <F8> e observe o resultado ocorrido como mostra a Figura 7.30.

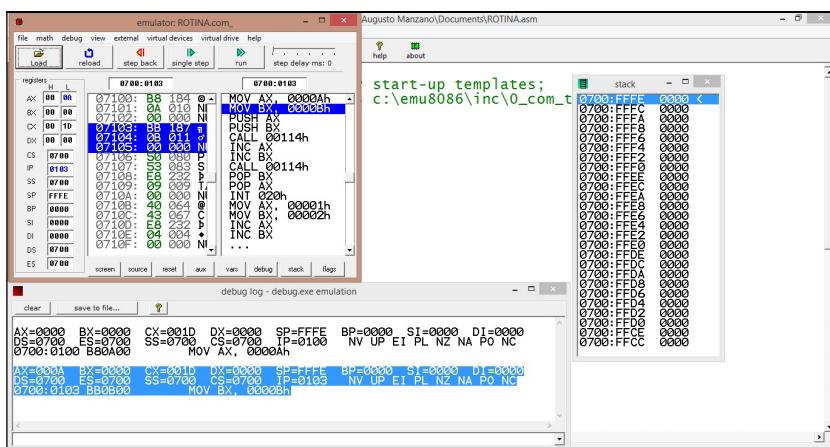


Figura 7.30 - Resultado da execução da instrução **MOV AX, 0000Ah**.

Observe a apresentação do valor **0A** no registrador **AL** e a mudança do valor do registrador **IP** de **0100** para **0103**. Na sequência acione a tecla de <F8> e observe o resultado ocorrido como mostra a Figura 7.31.

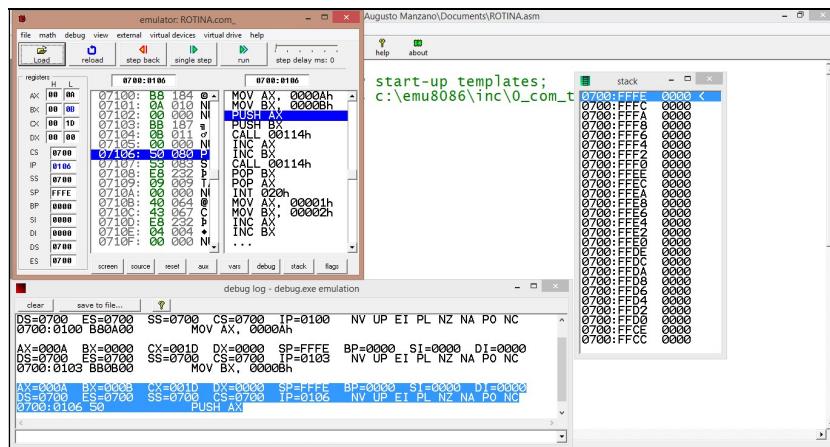


Figura 7.31 - Resultado da execução da instrução MOV AX, 0000Bh.

Observe a apresentação do valor **0B** no registrador **BL** e a mudança do valor do registrador **IP** de **0103** para **0106**. Na sequência acione a tecla de <F8> e note o que ocorre com o valor do registrador **SP** que marca o valor **FFFE** e passa a marcar o valor **FFFC** como indica a Figura 7.32.

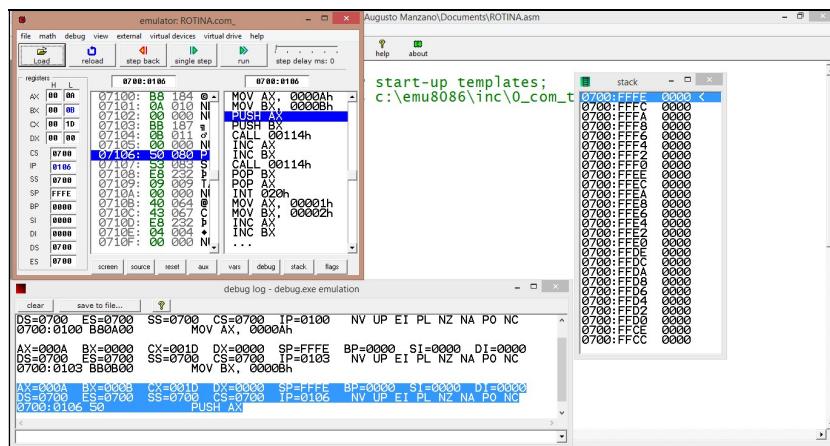


Figura 7.32 - Resultado da execução da instrução PUSH AX.

Observe a apresentação do valor **FFFC** no registrador **SP**, a mudança do valor do registrador **IP** de **0106** para **0107** e a mudança na pilha (janela **stack**) do valor **0000** para **000A** definido no endereço **0700:FFFC**. Na sequência acione a tecla de <F8> e note as mudanças comentadas a seguir como indica a Figura 7.33.

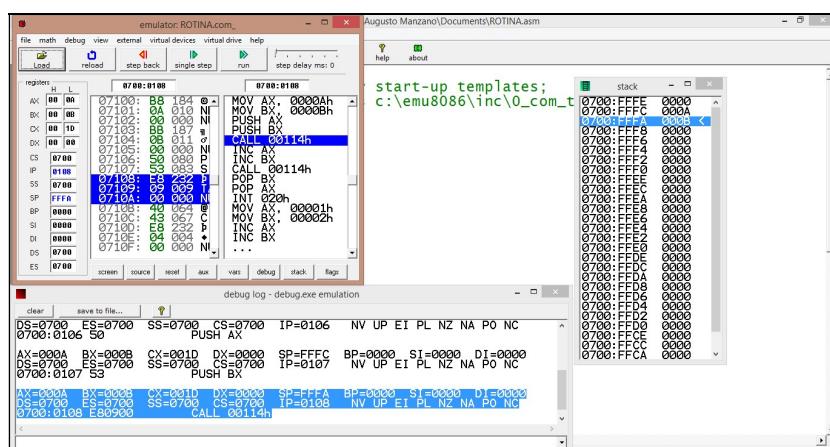


Figura 7.33 - Resultado da execução da instrução CALL 00114h.

A janela **stack** mostra os valores armazenados na pilha, sendo o valor **000A** no endereço de pilha **0700:FFF8** e o valor **000B** no endereço **0700:000A**. Já a janela **debug log - debug.exe emulation** mostra a mudança do valor do registrador **SP** de **FFF8** para **FFFA** e do registrador **IP** com o valor **0108**, além de indicar a execução da próxima instrução **CALL 00114h** equivalente a instrução **CALL procedimento** definido na linha de código **11**. O endereço **00114h** é o local onde se encontra o código do procedimento a ser executado.

O próximo passo a ser executado é a chamada do código do procedimento definido. Assim sendo, acione a tecla de **<F8>** e note as mudanças comentadas a seguir como indica a Figura 7.34.

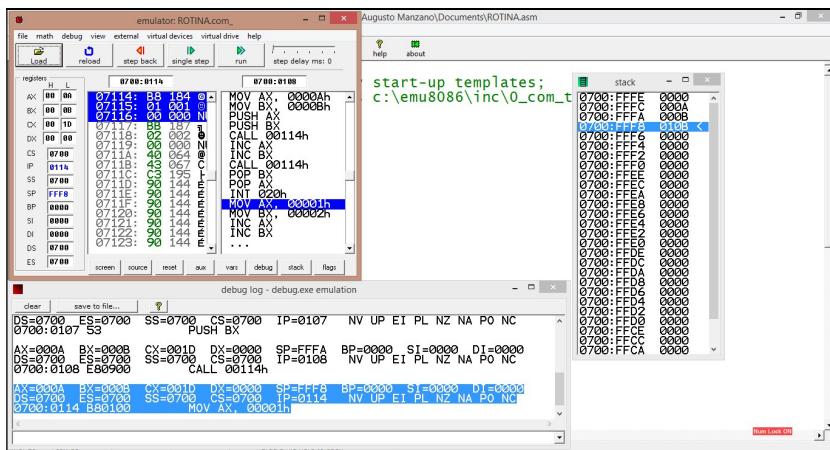


Figura 7.34 - Resultado da execução da instrução **MOV AX, 00001**.

Note as mudanças ocorridas nos registradores **SP** de **FFF8** para **FFFA** e **IP** de **0108** para **0114** e o armazenamento do valor **010B** na posição **0700:FFF8** da pilha que é o valor de retorno para uso do registrador **IP** após a execução do comando **RET** no retorno do procedimento executado. Acione a tecla de **<F8>** e note as mudanças comentadas a seguir como indica a Figura 7.35.

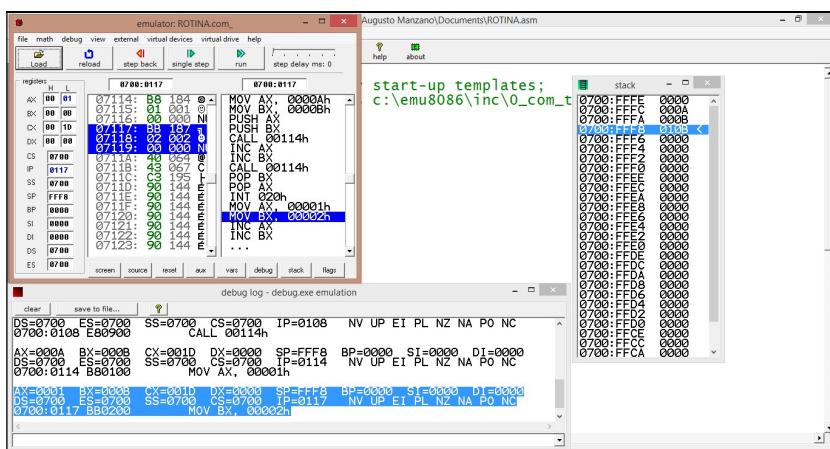


Figura 7.35 - Resultado da execução da instrução **MOV BX, 00002**.

Note que neste momento o registrador **SP** mantém foco sob o valor **FFF8** que não foi alterado, pois guarda o valor do endereço de retorno que será usado pelo comando **RET** para retornar o fluxo de execução do programa para a próxima linha após a chamada do procedimento. A alteração ocorre apenas no registrador **AX** que é alterado de **000A** para **0001** e no registrador **IP** de **0114** para **0117**. Acione em seguida a tecla de **<F8>** e note as mudanças comentadas a seguir como indica a Figura 7.36.

Observe a mudança do valor do registrador **BX** que passa a ser **0002** e do registrador **IP** de **0117** para **011A**. A Figura 7.36 apresenta a execução da próxima instrução **INC AX** que acrescentará o valor **1** ao valor existente no registrador **AX**, ou seja, passará a ser **0002**. Nesta etapa acione a tecla de função **<F8>** por duas vezes de forma que o registrador **AX** fique com o valor **0002** e o registrador **BX** fique com o valor **0003** como indica a Figura 7.37.

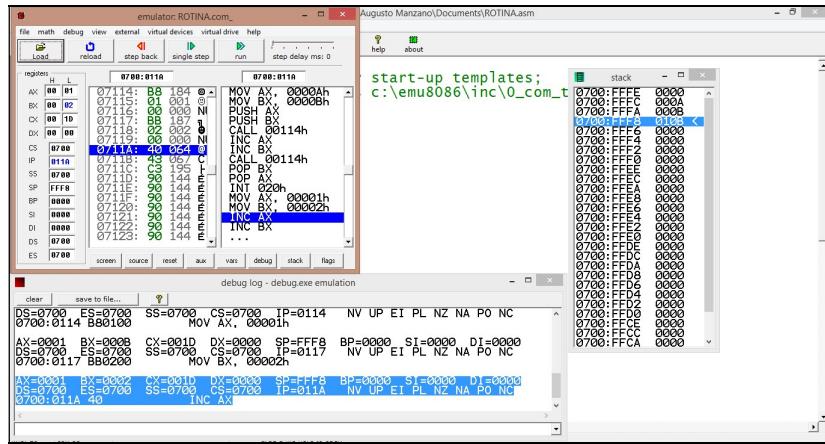


Figura 7.36 - Resultado da execução da instrução INC AX.

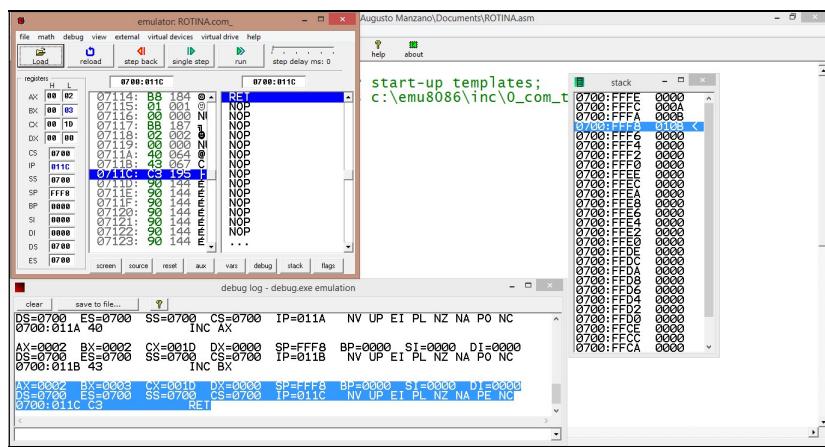


Figura 7.37 - Resultado da execução da instrução RET.

Após as últimas duas execuções os registradores **AX** e **BX** estão respectivamente os valores **0002** e **0003**. Na sequência acione o **<F8>** e observe a execução do comando **RET**. O programa retorna para o endereço **0700:010B**, sendo o valor **010B** obtido a partir da posição **0700:FFF8** da pilha, como demonstrado na Figura 7.38.

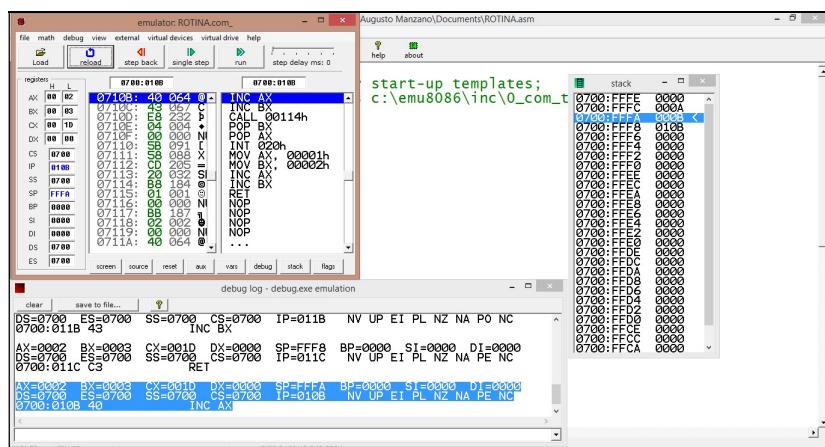


Figura 7.38 - Resultado da execução após uso da instrução RET.

Neste momento acione novamente por duas vezes a tecla de função **<F8>** e observe os registradores **AX** e **BX** assumirem respectivamente os valores **0003** e **0004** como mostra a Figura 7.39.

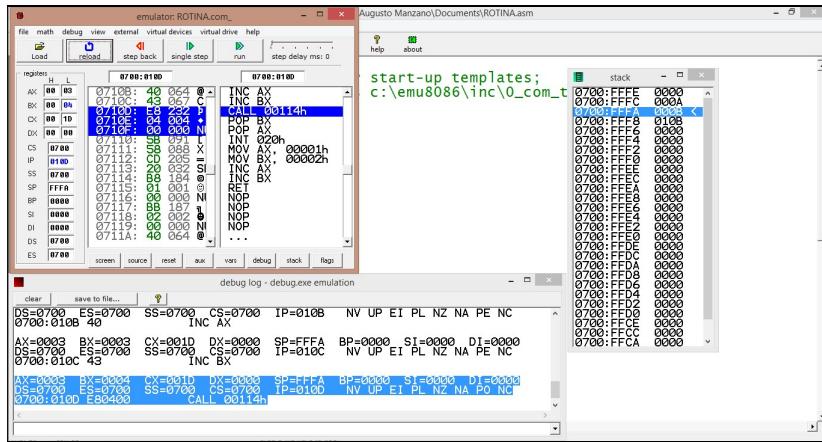


Figura 7.39 - Resultado da execução após os dois últimos incrementos.

A próxima etapa chama novamente o procedimento situado no endereço **00114h**. Assim sendo, acione **<F8>** e observe que o endereço da pilha **0700:FFF8** armazena agora o valor **0110** (antes armazenava o valor **010B**) que é o endereço de retorno a ser executado quando do uso do comando **RET**. A Figura 7.40 mostra essas últimas mudanças.

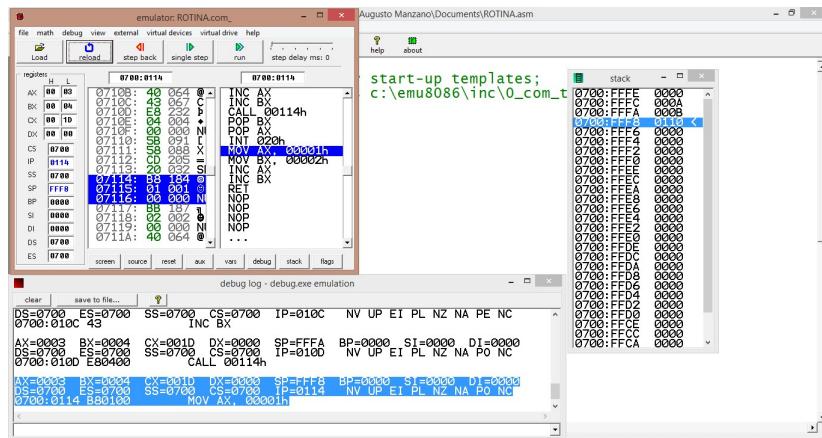


Figura 7.40 - Resultado da execução da segunda chamada do procedimento.

Os dois próximos passos substituem respectivamente os valores dos registradores **AX** e **BX** para **0001** e **0002**. Assim sendo, acione **<F8>** mais duas vezes e observe o resultado indicado na Figura 7.41.

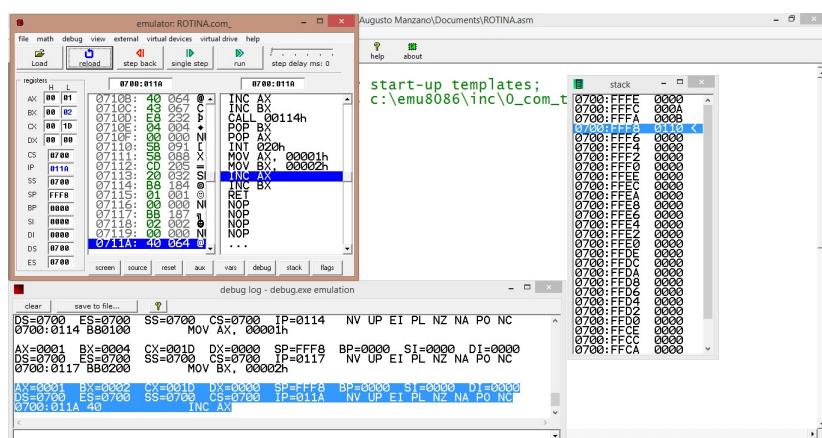


Figura 7.41 - Resultado da execução da definição de valores em AX e BX.

Os dois próximos passos são os incrementos do valor **1** nos valores existentes dos registradores **AX** e **BX** de forma que passem a possuir os valores **0002** e **0003**. Assim sendo, acione **<F8>** duas vezes e observe a Figura 7.42.

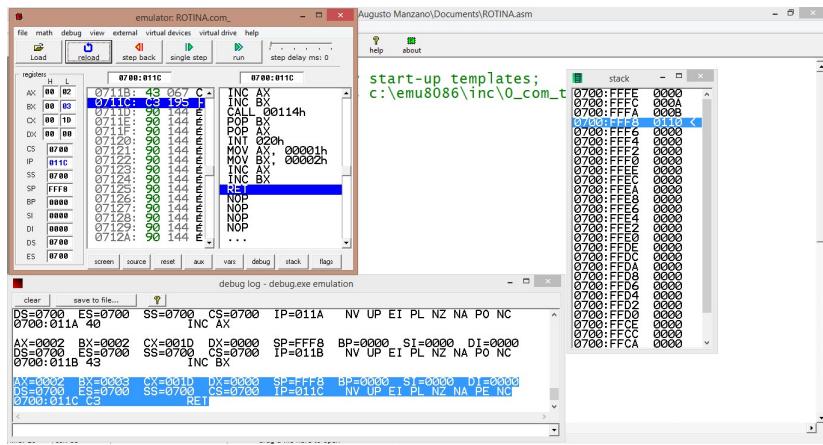


Figura 7.42 - Resultado da execução da definição de valores em AX e BX após incrementos.

O próximo passo é a execução do comando **RET** que retornará o fluxo de operação do programa para o endereço **0110** que é a primeira instrução após a segunda chamada de ação do procedimento. Assim sendo, acione <F8> e observe que a marcação de posição da janela **stack** aponta o endereço **0700:FFFA** que indica o acesso ao valor **000B** que será recuperado pela instrução **POP BX**. A Figura 7.43 mostra este momento.

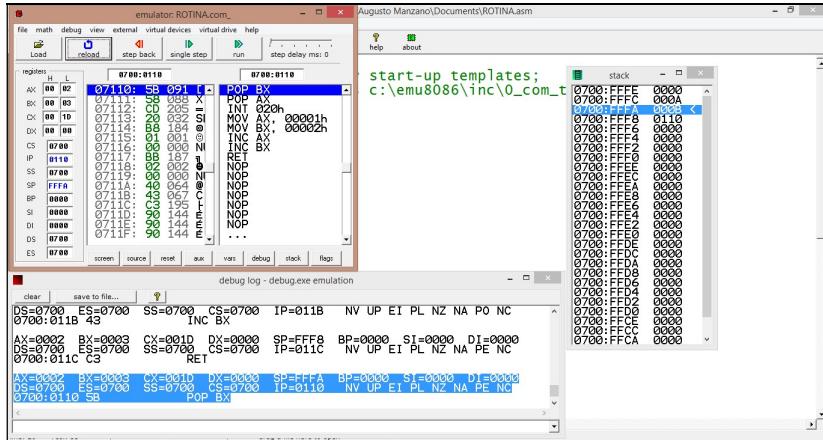


Figura 7.43 - Indicação do valor armazenado na pilha a ser recuperado.

Ao ser acionada a tecla de função <F8> ocorrerá a recuperação do valor **000B** para o registrador **BX**. Note esta ocorrência junto a Figura 7.44 e observe a mudança dos valores definidos no registrador **IP** e no registrador **SP** que mostra o endereço de recuperação dos valores armazenados na pilha.

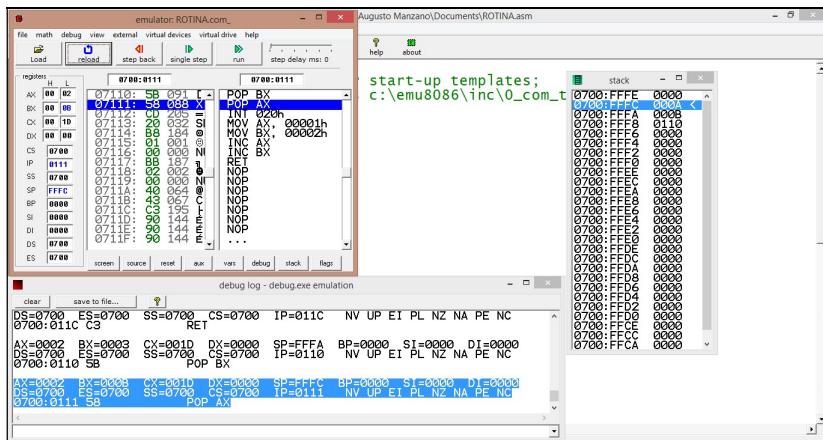


Figura 7.44 - Indicação da recuperação do valor 000B armazenado na pilha.

Na sequência acione **<F8>** para recuperar o valor **000A** para o registrador **AX** e em seguida acione mais duas vezes **<F8>** para encerrar a execução passo a passo do programa. Acione o botão **OK** e feche as janelas abertas.

7.5 - Instruções da linguagem de montagem 8086

O microprocessador 8086/8088 (da Intel e seus compatíveis, como os da AMD) tem um conjunto de instruções que possibilita o total controle de um computador. O conjunto de instruções do microprocessador 8086/8088 pode ser categorizado nos seguintes tipos, de acordo com instruções encontradas no programa **emu8086**:

- ◆ Transferência de dados para entrada e saída (IN, OUT);
- ◆ Transferência de dados gerais (MOV, POP, POPA, PUSH, PUSHA, XCHG, XLATB);
- ◆ Transferência de dados relacionada a endereços de segmento (LEA, LDS e LES);
- ◆ Transferência de dados para controle de registradores de estado (LAHF, SAHF, PUSHF e POPF);
- ◆ Aritmética para adição (AAA, ADD, ADC, DAA e INC);
- ◆ Aritmética para subtração (AAS, DAS, CMP, SBB, SUB, DEC e NEG);
- ◆ Aritmética para multiplicação (AAM, MUL e IMUL);
- ◆ Aritmética para divisão (AAD, CBW, CWD, DIV e IDIV);
- ◆ Manipulação de bit para operação lógica (AND, NOT, OR, TEST e XOR);
- ◆ Manipulação de bit para deslocamento (SHL/SAL, SHR e SAR);
- ◆ Manipulação de bit para rotação (RCL, RCR, ROL e ROR);
- ◆ Manipulação de sequências de caracteres (CMPSB, CMPSW, LODSB, LODSW, MOVSB, MOVSW, REP, REPE, REPNE, REPZ, REPNZ, SCASB, SCASW, STOSB e STOSW);
- ◆ Transferência de programa com instrução incondicional (CALL, JMP, RET e RETF);
- ◆ Transferência de programa com instrução condicional (JA, JAE, JB, JBE, JC, JE, JG, JGE, JL, JLE, JNA, JNAE, JNB, JNBE, JNC, JNG, JNGE, JNE, JNL, JNLE, JNO, JNP, JPO, JS, JZ, JNS, JNZ, JO, JP e JPE);
- ◆ Transferência de programa com instrução de controle iterativo (JCXZ, LOOP, LOOPE, LOPPNE, LOPPNZ e LOOPZ);
- ◆ Transferência de programa com instrução de interrupção (INT, INTO e IRET);
- ◆ Instruções de controle do processador em operações de estado (CLC, CLD, CLI, CMC, STC, STD e STI);
- ◆ Não classificada (NOP).

Toda linguagem de programação é composta por um conjunto de instruções, as quais, de forma geral, podem ser formadas por agrupamentos de comandos, diretivas, parâmetros e funções internas, que permitem a um programador "conversar" com um computador por meio do programa que desenvolve.

As instruções de uma linguagem formam o conjunto de palavras reservadas da linguagem. No caso da linguagem *Assembly*, o conjunto de palavras reservadas recebe o nome de *mneumônico* ("menemônicos"). Uma palavra reservada não pode ser usada para qualquer tipo de operação que não seja aquela para a qual foi projetada.

A linguagem *Assembly* para a manipulação apenas do microprocessador padrão 8086/8088 de acordo com a documentação dos manuais da Intel possui um conjunto de 113 mneumônicos (instruções), chegando a um total de 124 instruções se considerar o uso dos microprocessadores 80186/80188.

O programa **emu8086** faz uso de 116 instruções, em que são omitidas as instruções BOUND, ESC, HLT, LOCK e WAIT da série 80186/10188, exceto a instrução PUSHA; faz-se a inserção da instrução RETF não existente nos processadores 8086/8088 (80186/80188), mas encontrada nos microprocessadores de 32 e 64 bits. Efetua o desmembramento das instruções encontradas nos processadores acima do 80286: CMPS (como CMPSB e MPSW), LODS (como LODSB e LODSW), MOVS (MOVSB e MOVSW), SCAS (SCASB e SCASW), STOS (STOSB e STOSW) e o uso da instrução XLAT como XLATB, como mostra a Tabela 7.1.

Tabela 7.1 - Comandos encontrados no programa emu8086

Mnemônico	Significado	Sintaxe
AAA	ASCII Adjust for Addition	AAA
AAD	ASCII Adjust for Division	
AAM	ASCII Adjust for Multiply	AAM
AAS	ASCII Adjust for Subtraction	AAS
ADC	Add with Carry	ADC destino, origem
ADD	ADDition	ADD destino, origem
AND	AND logical	AND destino, origem
CALL	CALL procedure	CALL nome_procedimento
CBW	Convert Byte to Word	CBW
CLC	Clear Carry	CLC
CLD	Clear Direction flag	CLD
CLI	Clear Interrupt-enable flag	CLI
CMC	CoMplement Carry flag	CMC
CMP	CoMPare	CMP destino, origem
CMPSB	CoMPare String Byte	CMPSB texto_destino, texto_origem
CMPSW	CoMPare String Word	CMPSW texto_destino, texto_origem
CWD	Convert Word to Doubleword	CWD
DAA	Decimal Adjust for Addition	DAA
DAS	Decimal Adjust for Subtraction	DAS
DEC	DECrement	DEC
DIV	DIVide	DIV origem
HLT	HaLT	HLT
IDIV	Integer DIVide	IDIV origem
IMUL	Integer MULTiply	IMUL origem
IN	Input from port	IN acumulador, porta
INC	INCrement	INC destino
INT	INTerrupt	INT tipo_interrupção
INTO	INTerrupt on Overflow	INTO
IRET	Interrupt RETurn	IRET
JA	Jump on Above	JA
JAE	Jump on Above or Equal	JAE
JB	Jump on Below	JB
JBE	Jump on Below or Equal	JBE
JC	Jump on Carry	JC
JCXZ	Jump if CX register Zero	JCXZ rótulo
JE	Jump on Equal	JE
JG	Jump on Greater	JG
JGE	Jump on Greater or Equal	JGE
JL	Jump on Less	JL
JLE	Jump on Less or Equal	JLE
JMP	JuMP unconditionally	JMP rótulo
JNA	Jump on Not Above	JNA
JNAE	Jump on Not Above or Equal	JNAE

Mnemônico	Significado	Sintaxe
JNB	Jump on Not Below	JNB
JNBE	Jump on Not Below or Equal	JNBE
JNC	Jump on Not Carry	JNC
JNE	Jump on Not Equal	JNE
JNG	Jump on Not Greater	JNG
JNGE	Jump on Not Greater or Equal	JNGE
JNL	Jump on Not Less	JNL
JNLE	Jump on Not Less or Equal	JNLE
JNO	Jump on Not Overflow	JNO
JNP	Jump on Not Parity	JNP
JNS	Jump on Not Sign	JNS
JNZ	Jump on Not Zero	JNZ
JO	Jump on Overflow	JO
JP	Jump on Parity	JP
JPE	Jump on Parity Equal	JPE
JPO	Jump on Parity Odd	JPO
JS	Jump on Sign	JS
JZ	Jump on Zero	JZ
LAHF	Load register AH from Flags	LAHF
LDS	Load pointer using DS	LDS destino, origem
LEA	Load Effective Address	LEA destino, origem
LES	Load pointer using ES	LES destino, origem
LODSB	LOAD String Byte	LODSB texto_origem
LODSW	LOAD String Word	LODSW texto_origem
LOOP	LOOP	LOOP rótulo
LOOPE	LOOP while Equal	LOOPE rótulo
LOOPNE	LOOP while Not Equal	LOOPNE rótulo
LOOPNZ	LOOP while Not Zero	LOOPNZ rótulo
LOOPZ	LOOP while Zero	LOOPZ rótulo
MOV	MOVE (byte ou word)	MOV destino, origem
MOVSB	MOVE String Byte	MOVSB texto_destino, texto_origem
MOVSW	MOVE String Word	MOVSW texto_destino, texto_origem
MUL	MULTiply	MUL origem
NEG	NEGate	NEG destino
NOP	No Operation	NOP
NOT	logical NOT	NOT destino
OR	logical OR	OR destino, origem
OUT	OUTput	OR porta, acumulador
POP	POP	POP destino
POPA	POP All registers	POPA
POPF	POP Flags	POPF
PUSH	PUSH	PUSH origem
PUSHA	PUSH All registers	PUSHA
PUSHF	PUSH Flags	PUSHF

Mnemônico	Significado	Sintaxe
RCL	Rotate through Carry Left	RCL destino, contador
RCR	Rotate through Carry Right	RCR destino, contador
REP	REPeat	REP
REPE	REPeat while Equal	REPE
REPNE	REPeat while Not Equal	REPNE
REPNZ	REPeat while Not Zero	REPNZ
REPZ	REPeat while Zero	REPZ
RET	RETurn	RET valor_opcional_pop
RETF	RETurn from Far procedure	RETF
ROL	ROtate Left	ROL destino, contador
ROR	ROtate Right	ROR destino, contador
SAHF	Store register AH into Flags	SAHF
SAL	SHift Arithmetic Left	SAL destino, contador
SAR	Shift Arithmetic Right	SAR destino, contador
SBB	SuBtract with Borrow	SBB destino, origem
SCASB	SCAn String Byte	SCASB texto_destino
SCASW	SCAn String Word	SCASW texto_destino
SHL	SHifit logical Left	SHL destino, contador
SHR	SHift logical Right	SHR destino, origem
STC	SeT Carry	STC
STD	SeT Direction flag	STD
STI	SeT Interrupt enable flag	STI
STOSB	STOre String (byte ou word)	STOSB texto_destino
STOSW	STOre String (byte ou word)	STOSW texto_destino
SUB	SUBtract	SUB destino, origem
TEST	TEST	TEST destino, origem
XCHG	Exchange	XCHG destino, origem
XLATB	Translate table look-up	XLATB tabela
XOR	eXclusive OR	XOR destino, origem

A primeira coluna da tabela descreve o nome da instrução, a segunda coluna mostra sinalizando com caracteres maiúsculos o significado da instrução e a terceira coluna apresenta de forma resumida a sintaxe da instrução.

Note junto a terceira coluna algumas instruções que possuem em sua sintaxe a definição de elementos complementares, como *destino*, *origem*, *acumulador*, *porta*, *texto_destino*, *texto_origem*, *nome_procedimento*, *tipo_interrupção*, *rótulo*, *contador* e *tabela*, *valor_opcional_pop*, os quais definem o tipo de operação que pode ser efetuado com uma determinada instrução.

A Tabela 7.2 apresenta e descreve o significado de cada um dos elementos apontados no parágrafo anterior.

Tabela 7.2 - Elementos complementares ao uso dos comandos

Elemento	Usado em operações	Descrição
origem	de transferência de dados, aritméticas, de manipulação de bit	Um registro, um endereço de memória ou um valor definido que será usado na operação, mas que não alterará a ação da instrução.
destino	de transferência de dados, de manipulação de bit	Um registro ou endereço de memória que contém o dado que será operacionalizado pela instrução.
acumulador	com as instruções in e out	Transferência de dado do tipo Word para o registrador AX e de dado do tipo byte para o registrador AL.
porta	com as instruções in e out	Especificação de um número de 0 a 255 na definição da porta de entrada ou saída a ser utilizada. O valor fornecido ocorre de forma direta no registrador DX.
texto_destino	de manipulação de texto	Nome do <i>string</i> definido em memória por meio do endereçamento do registrador DI que será substituído pelo resultado da operação.
texto_origem	de manipulação de texto	Nome do <i>string</i> definido em memória por meio do endereçamento do registrador SI que será usado na operação de substituição, mas não será alterado pela operação.
nome_procedimento	de sub-rotinas	Definição do nome de uma sub-rotina que será chamada dentro do programa.
tipo_interrupção	com a instrução int	Valor numérico entre 0 e 255 utilizado para definir a interrupção do sistema a ser utilizada.
rótulo	de condições, transferências, de controle iterativo	Definição de um nome de identificação que será usado como ponto de desvio para a ação pretendida.
contador	de deslocamento de rotações	Especificação de um valor numérico para operações de deslocamento e de rotação definidas com o registrador CS em valores entre 0 e 255.

Elemento	Usado em operações	Descrição
tabela	com a instrução xlatb	Nome do endereço de memória usado na tradução de uma tabela usada com o registrador BX.
valor_opcional_pop	com a instrução ret	Definição do número de bytes que será usado nas operações de pilha.

Do conjunto de instruções apresentadas, algumas delas, quando executadas, alteram os valores dos registra-dores de flags. Assim sendo, segue tabela com indicação apenas das instruções que afetam o conjunto de registradores de flags. As instruções que não operam ou não afetam seus valores foram omitidas. Cabe salientar que o registrador TF não é afetado.

Os registradores de flags podem assumir um de quatro comportamentos possíveis, sendo o valor binário **1** (um) ou **0** (zero) quando são assim explicitamente definidos por alguma instrução em execução (sinalizado na tabela como **UM** - um e **ZR** - zero); podem assumir um valor que depende da instrução em execução e neste caso seu valor binário pode ser **1** ou **0** (sinalizado na tabela como **DP** - depende); podem assumir um valor indefinido e neste caso talvez ser o valor binário **1** ou **0**, dependendo da instrução em execução (sinalizado na tabela como **ID** - independe) ou podem não assumir nenhum valor (sinalizado na tabela como **NV** - nenhum valor). Os registradores não afetados pela instrução em execução estão sinalizados com a identificação **NA** - não afetado. A Tabela 7.3 descreve os flags afetados com o uso das instruções.

Tabela 7.3 - Instruções e flags afetados

Instrução	Flags afetados							
	CF	DF	ZF	SF	OF	PF	AF	IF
AAA	DP	NA	IN	IN	IN	IN	DP	NA
	CF	DF	ZF	SF	OF	PF	AF	IF
AAD	IN	NA	DP	DP	IN	DP	IN	NA
	CF	DF	ZF	SF	OF	PF	AF	IF
AAM	IN	NA	DP	DP	IN	DP	IN	NA
	CF	DF	ZF	SF	OF	PF	AF	IF
AAS	DP	NA	IN	IN	IN	IN	DP	NA
	CF	DF	ZF	SF	OF	PF	AF	IF
ADC	DP	NA	DP	DP	DP	DP	DP	NA
	CF	DF	ZF	SF	OF	PF	AF	IF
ADD	DP	NA	DP	DP	DP	DP	DP	NA
	CF	DF	ZF	SF	OF	PF	AF	IF
AND	ZR	NA	DP	DP	ZR	DP	IN	NA
	CF	DF	ZF	SF	OF	PF	AF	IF
CLC	ZR	NA						
	CF	DF	ZF	SF	OF	PF	AF	IF
CLD	NA	ZR	NA	NA	NA	NA	NA	NA
	CF	DF	ZF	SF	OF	PF	AF	IF
CLI	NA	NA	NA	NA	NA	NA	NA	ZR
	CF	DF	ZF	SF	OF	PF	AF	IF
CMC	DP	NA						
	CF	DF	ZF	SF	OF	PF	AF	IF

Instrução	Flags afetados							
CMP	CF	DF	ZF	SF	OF	PF	AF	IF
	DP	NA	DP	DP	DP	DP	DP	NA
CMPSB	CF	DF	ZF	SF	OF	PF	AF	IF
	DP	NA	DP	DP	DP	DP	DP	NA
CMPSW	CF	DF	ZF	SF	OF	PF	AF	IF
	DP	NA	DP	DP	DP	DP	DP	NA
DAA	CF	DF	ZF	SF	OF	PF	AF	IF
	DP	NA	DP	DP	DP	DP	DP	NA
DAS	CF	DF	ZF	SF	OF	PF	AF	IF
	DP	NA	DP	DP	DP	DP	DP	NA
DEC	CF	DF	ZF	SF	OF	PF	AF	IF
	NA	NA	DP	DP	DP	DP	DP	NA
DIV	CF	DF	ZF	SF	OF	PF	AF	IF
	IN	NA	IN	IN	IN	IN	IN	NA
IDIV	CF	DF	ZF	SF	OF	PF	AF	IF
	IN	NA	IN	IN	IN	IN	IN	NA
IMUL	CF	DF	ZF	SF	OF	PF	AF	IF
	DP	NA	IN	IN	DP	IN	IN	NA
INC	CF	DF	ZF	SF	OF	PF	AF	IF
	IN	NA	DP	DP	DP	DP	DP	NA
INT	CF	DF	ZF	SF	OF	PF	AF	IF
	NA	NA	NA	NA	NA	NA	NA	ZR
MUL	CF	DF	ZF	SF	OF	PF	AF	IF
	DP	NA	IN	IN	DP	IN	IN	NA
NEG	CF	DF	ZF	SF	OF	PF	AF	IF
	DP	NA	DP	DP	DP	DP	DP	NA
OR	CF	DF	ZF	SF	OF	PF	AF	IF
	ZR	NA	DP	DP	ZR	DP	IN	NA
RCL	CF	DF	ZF	SF	OF	PF	AF	IF
	DP	NA	NA	NA	DP	NA	NA	NA
RCR	CF	DF	ZF	SF	OF	PF	AF	IF
	DP	NA	NA	NA	DP	NA	NA	NA
REP	CF	DF	ZF	SF	OF	PF	AF	IF
	NA	NA	DP	NA	NA	NA	NA	NA
REPE	CF	DF	ZF	SF	OF	PF	AF	IF
	NA	NA	DP	NA	NA	NA	NA	NA
REPNE	CF	DF	ZF	SF	OF	PF	AF	IF
	NA	NA	DP	NA	NA	NA	NA	NA
REPNZ	CF	DF	ZF	SF	OF	PF	AF	IF
	NA	NA	DP	NA	NA	NA	NA	NA
REPZ	CF	DF	ZF	SF	OF	PF	AF	IF
	NA	NA	DP	NA	NA	NA	NA	NA

Instrução	Flags afetados							
	CF	DF	ZF	SF	OF	PF	AF	IF
ROL	DP	NA	NA	NA	DP	NA	NA	NA
	CF	DF	ZF	SF	OF	PF	AF	IF
ROR	DP	NA	NA	NA	DP	NA	NA	NA
	CF	DF	ZF	SF	OF	PF	AF	IF
SAHF	DP	NA	DP	DP	DP	DP	DP	NA
	CF	DF	ZF	SF	OF	PF	AF	IF
SAL	DP	NA	NA	NA	DP	NA	NA	NA
	CF	DF	ZF	SF	OF	PF	AF	IF
SAR	DP	NA	NA	NA	DP	NA	NA	NA
	CF	DF	ZF	SF	OF	PF	AF	IF
SBB	DP	NA	DP	DP	DP	DP	DP	NA
	CF	DF	ZF	SF	OF	PF	AF	IF
SCASB	DP	NA	DP	DP	DP	DP	DP	NA
	CF	DF	ZF	SF	OF	PF	AF	IF
SCASW	DP	NA	DP	DP	DP	DP	DP	NA
	CF	DF	ZF	SF	OF	PF	AF	IF
SHL	DP	NA	NA	NA	DP	NA	NA	NA
	CF	DF	ZF	SF	OF	PF	AF	IF
SHR	DP	NA	NA	NA	DP	NA	NA	NA
	CF	DF	ZF	SF	OF	PF	AF	IF
STC	UM	NA						
	CF	DF	ZF	SF	OF	PF	AF	IF
STD	NA	UM	NA	NA	NA	NA	NA	NA
	CF	DF	ZF	SF	OF	PF	AF	IF
STI	NA	NA	NA	NA	NA	NA	NA	UM
	CF	DF	ZF	SF	OF	PF	AF	IF
SUB	DP	NA	DP	DP	DP	DP	DP	NA
	CF	DF	ZF	SF	OF	PF	AF	IF
TEST	ZR	NA	DP	DP	ZR	DP	NA	NA
	CF	DF	ZF	SF	OF	PF	AF	IF
XOR	ZR	NA	DP	DP	ZR	DP	IN	NA
	CF	DF	ZF	SF	OF	PF	AF	IF

7.6 - Uso do modo ajuda

O estudo da linguagem Assembly para microcomputadores IBM-PC é sempre realizado a partir da edição 8086/8088 do microprocessador Intel. Uma vez que se tenha uma boa noção do funcionamento das instruções e da estrutura da arquitetura do microprocessador, fica mais fácil utilizar processadores de 32 e 64 bits.

O programa de simulação do microprocessador Intel 8086/8088 **emu8086** foi desenvolvido com objetivos didáticos. Ele possui uma série de recursos que o tornam uma ferramenta bastante prática para o uso escolar.

Este foi um dos principais fatores que incentivaram o uso dessa ferramenta para a elaboração desta obra, pois além da execução pausada das instruções Assembly, a ferramenta disponibiliza várias janelas que mostram informações sobre os registradores, flags, pilha, entre outros detalhes, como a existência de um *tutorial* muito bem elaborado. Esses detalhes vieram ao encontro dos objetivos deste trabalho.

Apesar do **emu8086** ser uma ferramenta prática e possuir diversos recursos, tem algumas pequenas limitações e restrições no uso de alguns recursos, como, por exemplo, não trabalhar com o código de função **08h** da interrupção **21h** para a entrada de dados sem eco de teclado, permitir apenas o uso das diretivas de definição de tipos de variáveis **DB** e **DW** (que ainda serão explanadas).

Para obter maiores detalhes a respeito de funcionalidades e restrições do uso da ferramenta, sugere-se uma leitura detalhada da documentação anexa à ferramenta. Execute no editor do programa (tela principal) o comando de menu **help/documentation and tutorial**, que apresenta o modo ajuda indicado na Figura 7.45.

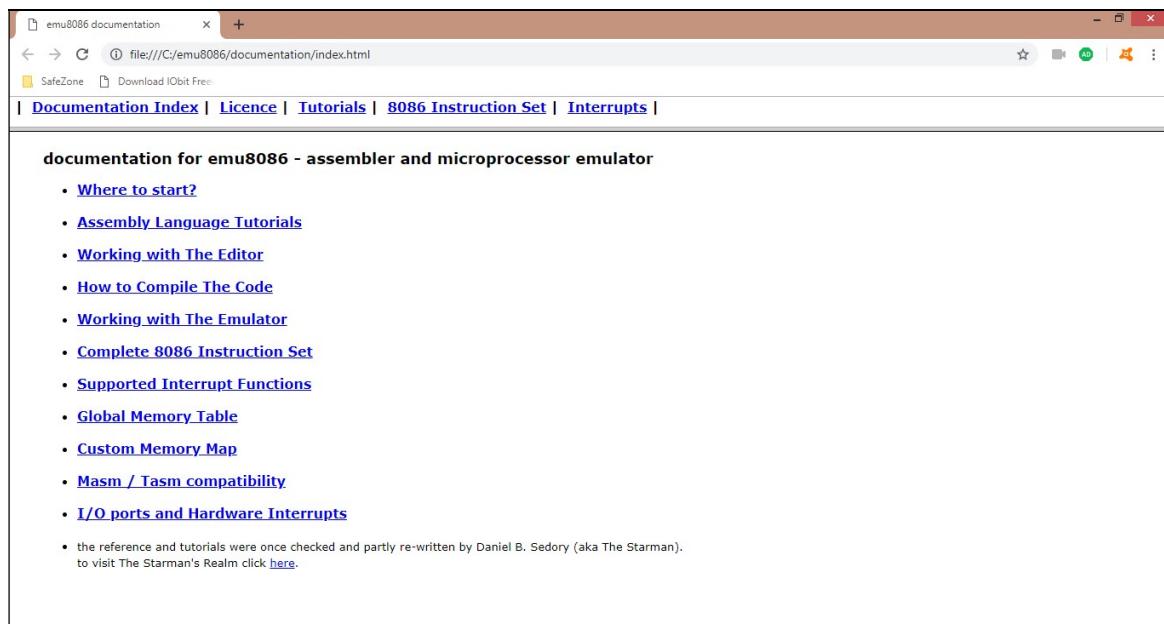


Figura 7.45 - Tela do modo ajuda.