Linguagem Assembly

Roteiro Nº 01

Fundação Universidade Federal de Rondônia, Núcleo de Ciência e Tecnologia, Departamento de Engenharia - DEE
Curso de Bacharelado em Engenharia Elétrica - Disciplina de Sistemas Microprocessados
Elaboração: Ivan S. de Oliveira - Revisão: Prof. M.Sc. Ciro Egoavil
Laboratório de Sistemas Microprocessados

I. Introdução

O Termo *Assembly* significa montagem, ou seja, linguagem de montagem (Assembly Language), a qual é utilizada para programar em baixo nível, sendo necessário montar o programa dentro do processador. *Assembly* não é uma linguagem de máquina, mas é a linguagem de programação que está mais próxima disso.

A linguagem de máquina é a utilizada por um microprocessador para controlar as funções de um computador digital. A linguagem de máquina só aceita e manipula informações numéricas expressas em notação de códigos binários, os quais matematicamente representam os estados de tensão alta "1"ou tensão baixa "0"para os circuitos eletrônicos de um computador.

A linguagem de programação de computador *Assembly* possui uma estrutura sintática particular. Ela é formada por um conjunto de instruções que, por meio de códigos mais legíveis, representam as instruções de código de máquina. As instruções da linguagem *Assembly* são conhecidas pelo nome de *mnemônicos* (lê-se menemônicos). É muito mais fácil olhar para um *mnemônicos* e lembrar o que ele faz do seu equivalente em código binário ou hexadecimal, legível apenas pelo microprocessador (CPU) do computador.

O termo *Assembler* significa o programa montador, ou seja, é o programa utilizado para compilar um programa escrito em linguagem de montagem, tornando-o executável em um computador. *Assembler* é basicamente o ambiente de programação. É a ferramenta responsável por traduzir o programa-fonte (escrito em linguagem *Assembly*) para o programa objeto (programa em código de máquina) a ser interpretado por um processador. O programa *Assembler* é uma ferramenta que tem características semelhantes em alguns aspectos aos compiladores para uma determinada linguagem de alto nível.

II. Objetivo

O objetivo principal é realizar uma introdução e apresentação básica da linguagem e programação *Assembly* 8086 levando em consideração o uso de uma computador padrão IBM-PC dotado de qualquer tipo de microprocessador Intel ou AMD. As ferramentas utilizadas serão o **DEBUG** (modo MS-DOS) e o **Emu8086** voltadas para o padrão 8086.

III. Material Utilizado

- Computador padrão IBM-PC;
- **DEBUG**;
- **Emu8086**;

IV. Programação com DEBUG

O sistema operacional MS-DOS, assim como o ambiente gráfico de trabalho Windows, possui no diretório de sistemas um programa denominado **DEBUG** (lê-se debâgui).

O programa **DEBUG** é uma ferramenta básica que possibilita a manipulação de dados e dos registradores de memória em linguagem de máquina e também dá suporte à ação de comandos em modo *Assembly*, além de outros recursos. É uma ferramenta simples, mas bastante poderosa, pois tem a capacidade de permitir a criação de pequenos programas em linguagem de máquina.

Para iniciar a execução do programa, posicione-se na linha de comando do MS-DOS e faça a chamada pelo nome **DEBUG**, acionando em seguida a tecla **<Enter>**. A tela terá aparência semelhante a mostrada na Figura 1.

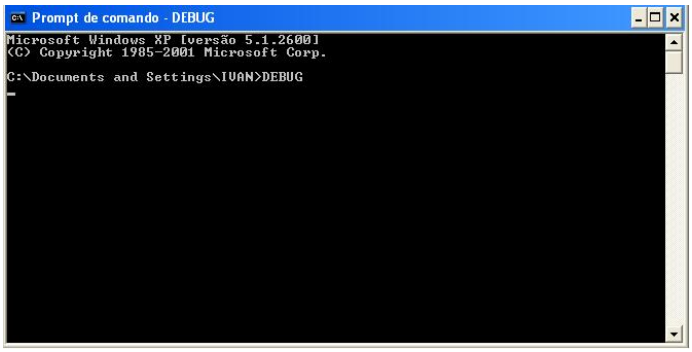

Figura 1. Aparência da tela com o programa DEBUG em operação.

Para sair do programa **DEBUG**, basta informar no seu *prompt* de trabalho o código **Q** (*quit*).

O programa **DEBUG** tem uma série de comandos que facilitam as operações em memória. Para ter uma idéia dos comandos disponíveis, acione no *prompt* do programa o comando **?** (ponto de interrogação), o qual apresenta a lista completa de comandos, conforme a Figura 2.

A. Registradores

A linguagem Assembly utiliza registradores para armazenar em memória os valores que serão manipulados por um programa. O registrador está intimamente relacionado com a estrutura do microprocessador em uso. Para cada tipo de processador existe uma forma peculiar de tratar este conceito.

Para visualizar a estrutura de registradores do processador do computar em uso, é necessário informar no *prompt* do

Figura 2. Lista de comandos do programa DEBUG.

programa **DEBUG** o comando **R** (register). Ao acionar o comando **R**, aparece uma listagem como a indicada a seguir:

```
-R
AX=0000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0CE1 ES=0CE1 SS=0CE1 CS=0CE1 IP=0100 NV UP EI PL NZ NA PO NC
0D0B:0100 69          JO      69
```

A primeira linha será idêntica em qualquer computador, mas é bem provável que na segunda e terceira linhas sejam apresentados dados com valores diferentes. A diferença encontrada nos valores se refere à quantidade de memória do computador em uso.

Os registradores apresentados são divididos em:

- Registradores Gerais;
- Registradores de Segmento;
- Registradores de Deslocamento ou Apontamento;
- Registradores de Estado;

1) ***Registradores gerais*** *::*

- **AX** = Accumulator extended;
- **BX** = Base extended;
- **AX** = Counter extended;
- **AX** = Data extended;

Os registradores gerais têm 16 bits de dados, e cada registrador pode ser dividido em duas partes, cada uma com 8 bits. Por exemplo, o registrador geral **AX** de 16 bits poder ser dividido no registrador de 8 bits **AH** (Accumulator Hight) e o registrador de 8 bits **AL** (Accumulator Low).

2) ***Registradores de Segmento*** *::*

- **CS** = Code Segment;
- **DS** = Data Segment;
- **ES** = Extra Data Segment;
- **SS** = Stack Segment;

Os registradores de segmento têm 16 bits e são utilizados para acessar uma determinada área de memória denominada *offset* (segmento), ou seja, são utilizados para auxiliar o microprocessador a encontrar o caminho pela memória do computador. Eles não podem ser divididos em registradores de 8 bits.

3) ***Registradores de Deslocamento ou Apontamento****::*

- **SI** = Source Index;
- **DI** = Destination Index;
- **SP** = Stack Index;
- **BP** = Base Pointer;
- **IP** = Instruction Pointer;

O registrador de deslocamento ou apontamento **IP**, conhecido como apontador da próxima instrução, possui o valor de deslocamento do código da próxima instrução a ser executada. Os registradores de apontamento **SI** e **DI** são índices de tabela. O registrador de apontamento **SI** faz a leitura e o registrador **DI** a escrita de uma tabela. Os registradores de apontamento **SP** e **BP** permitem acesso à pilha de programa (memória para armazenamento de dados). A pilha possibilita armazenar dados na memória, sem utilizar registradores gerais.

4) ***Registrador de Estado****: :*

- **CF** = Carry Flag;
- **PF** = Paity Flag;
- **AF** = Auxiliar Flag;
- **ZF** = Zero Flag;
- **SF** = Signal Flag;
- **OF** = Overflow Flag;
- **TF** = Trap Flag;
- **IF** = Interrupt Flag;
- **DF** = Direction Flag;

O registrador de estado (*Flag*) tem 16 bits que agrupa um conjunto de flags de 1 bit, e cada flag define ou sinaliza um estado de comportamento particular do computador. Se o valor de cada bit estiver sinalizado como "1", indica que o flag em questão está "setado"(acionado), caso esteja sinalizado com o valor "0", significa que o flag não está "setado"(desabilitado).

B. Apresentação de Dados

Como exemplo, deseja-se apresentar o caractere **A** na tela do MS-DOS. Inicialmente, informe na linha de *prompt* do programa a seguinte instrução:

```
R AX <Enter>
```

A tela terá aparência como a ilustrada abaixo:

```
-R AX
AX 0000
:
```

Ao lado do símbolo de dois-pontos informe o valor hexadecimal **0200** e acione a tecla **<Enter>**. Posteriormente, insira o comando **R** e acione tecla **<Enter>** e observe que o registrador **AX** apresenta agora o valor **0200**.

```
-R
AX=0200 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0CE1 ES=0CE1 SS=0CE1 CS=0CE1 IP=0100 NV UP EI PL NZ NA PO NC
0D0B:0100 69          JO      69
```

Agora informe ao registrador **DX** o valor hexadecimal **0041** (valor do caractere **A** na tabela ASCII), como demonstrado a seguir:

```
R DX <Enter>
```

A tela terá aparência como a ilustrada abaixo:

```
-R DX
DX 0000
:
```

Informe o valor o valor **0041** e observe os registradores através do comando **R**:

```
-R
AX=0200 BX=0000 CX=0000 DX=0041 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0CE1 ES=0CE1 SS=0CE1 CS=0CE1 IP=0100 NV UP EI PL NZ NA PO NC
0D0B:0100 69         JO      69
```

Com os valores armazenados nos registradores, é necessário informar as interrupções **21** e **20**, que são responsáveis pela apresentação do caractere e pelo encerramento do programa, respectivamente.

Acione, agora, o comando **A** *assemble* fornecendo como ponto inicial de armazenamento de código o deslocamento **0100**, como indicado a seguir:

```
-A 0100 <Enter>
```

Automaticamente será apresentada a linha (segmento:deslocamento)em que o código deve ser definido. Nesta etapa forneça as seguintes linhas de código:

```
INT 21 <Enter>
INT 20 <Enter>
<Enter>
```

Ao informar a primeira linha de código e acionar a tecla **<Enter>**, é apresentado automaticamente o segundo endereço de deslocamento. Nesse ponto, ao entrar a segundo linha de código e acionar **<Enter>**, passa-se para o próximo segmento, que se nenhum dado for inserido e for acionada a tecla **<Enter>**, o modo de trabalho do comando **A** é encerrado. A sequência de comandos tem a seguinte aparência no **prompt**:

```
-A 0100
0CE1:0100 INT 21
0CE1:0102 INT 20
0CE1:0104
```

O endereço de seguimento **0CE1** é um valor escolhido pelo próprio programa **DEBUG** e provavelmente será diferente conforme o computador usado.

Para fazer um teste de execução do programa, utilize o comando **G** sem indicar nenhum endereço de deslocamento inicial.

```
-G
A
O programa terminou de forma normal
```

C. Movimentação de Dados

Para realizar a movimentação de dados nos registradores e entre os registradores, é necessário utilizar uma instrução *Assembly* denominada **MOV** (move). Antes de realizar a movimentação, deve-se informar ao registrador seus valores. Para começar, informe para o registrador geral **AX** o valor **1122** e para o registrador **DX** o valor **AABB**. Verifique com o comando **R** se os valores estão como os seguintes:

```
-R
AX=1122 BX=0000 CX=0000 DX=AABB SP=FFEE BP=0000 SI=0000 DI=0000
DS=0CE1 ES=0CE1 SS=0CE1 CS=0CE1 IP=0100 NV UP EI PL NZ NA PO NC
0D0B:0100 69         JO      69
```

Como exemplo será realizada a movimentação da parte menos significativa do registrador **DX** para a parte mais significativa do operador **AX**. Na linha de *prompt* do programa **DEBUG** execute as seguintes linhas:

```
-A 0100
0CE1:0100 MOV AH,DL
0CE1:0102
```

Execute o comando **T** para visualizar o estado atual dos registradores durante a execução do programa, como indicado a seguir:

```
-T
AX=BB22 BX=0000 CX=0000 DX=AABB SP=FFEE BP=0000 SI=0000 DI=0000
DS=0CE1 ES=0CE1 SS=0CE1 CS=0CE1 IP=0100 NV UP EI PL NZ NA PO NC
0D0B:0100 69         JO      69
```

O valor hexadecimal **BB** armazenado na parte menos significativa do registrador **DL** pertencente ao registrador geral **DX** foi copiado para o registrador mais significativo do registrador geral **AX**. Para encerrar o programa, execute o comando **G** e será apresentada a mensagem: **O programa terminou de forma normal**.

O programa escrito anteriormente para apresentar um único caractere na tela, pode agora ser escrito utilizando o comando **MOV**, conforme as linhas de código indicadas abaixo:

```
-A 0100
0CE1:0100 MOV AH,02
0CE1:0102 MOV DL,41
0CE1:0104 INT 21
0CE1:0106 INT 20
0CE1:0108
```

Execute o comando **G** e a tela terá aparência conforme abaixo:

```
-G
A
O programa terminou de forma normal
```

D. Apresentar Sequência de Dados

Para apresentar na tela uma sequência de caractere na tela do monitor de vídeo, é necessário codificar cada caractere do *string* em seu equivalente hexadecimal.

Para apresentar a mensagem **ALO MUNDO!**, por exemplo, deve converter os caracteres para o equivalente hexadecimal de acordo com a tabela ASCII. No caso da mensagem pretendida os valores hexadeciamis são:

A	L	O		M	U	N	D	O	!
41	4C	4F	20	4D	55	4E	44	4F	21

È preciso colocar os códigos do *string* na memória, para isso execute o comando **E**, e informe o endereço de deslocamento a partir de **0200**.

Informe cada *byte* da tabela anterior separando-os por espaços em branco. Ao final acrescente o código hexadecimal **24** que representada o símbolo **$**, o qual define para o programa **DEBUG** o fim do *string*. A seguir é apresentada a aparência da tela do programa **DEBUG** após a entrada dos valores hexadecimais da mensagem **ALO MUNDO!**.

```
-E 0200 41 4C 4F 20 4D 55 4E 44 4F 21 24
```

É necessário, agora, informar para o registrador **DX** o endereço de deslocamento em que se encontra a sequência de caracteres. Neste caso, o endereço é **0200**. É necessário também definir o valor hexadecimal **09** no registrador mais significativo **AH**, para que um *string* seja impresso. Execute o comando **A 0100** e as seguintes linhas de código:

```
-A 0100
0CE1:0100 MOV AH,09
0CE1:0102 MOV DX,0200
0CE1:0105 INT 21
0CE1:0107 INT 20
0CE1:0109
```

No código anterior não está sendo utilizado o código **02** para o registrador mais significativo **AH**, mas sim o código **09**. O código **02** aciona o recurso de impressão de apenas um caractere, enquanto o código **09** aciona o serviço de apresentação de uma sequência de caracteres. Execute o comando **G** para que a mensagem seja apresentada, como indicado a seguir:

```
-G
ALO MUNDO!
O programa terminou de forma normal
```

O comando **U** permite visualizar a sequência de instruções de um programa. A acione este comando e visualize as instruções inseridas anteriormente, como mostrado a seguir:

```
-U
0CE1:0100 B409          MOV     AH,09
0CE1:0102 BA0002        MOV     DX,0200
0CE1:0105 CD21          INT     21
0CE1:0107 CD20          INT     20
0CE1:0109 756D          JNZ     0178
0CE1:010B 65            DB      65
0CE1:010C 20822025      AND     [BP+SI+2520],AL
0CE1:0110 312D          XOR     [DI],BP
```

No entanto é possível também visualizar as informações referentes ao armazenamento dos códigos hexadecimais dos caracteres da mensagem. Utilize o comando *D* (*dump* com o endereço do deslocamento desejado. Neste caso, acione o comando **D 0200 0280** e serão apresentados as seguintes informações:

```
-D 0200 0300
0CE1:0200  41 4C 4F 20 4D 55 4E 44-4F 21 24 72 71 75 69 76   ALO MUNDO!$rquiv
0CE1:0210  6F 20 69 6E 76 A0 6C 69-64 6F 0D 0A 10 41 63 65   o inv.lido...Ace
0CE1:0220  73 73 6F 20 6E 65 67 61-64 6F 20 0D 0A 2C 43 6F   sso negado ..,Co
0CE1:0230  6E 74 65 A3 64 6F 20 64-6F 20 64 65 73 74 69 6E   nte.do do destin
0CE1:0240  6F 20 70 65 72 64 69 64-6F 20 61 6E 74 65 73 20   o perdido antes
0CE1:0250  64 61 20 63 A2 70 69 61-0D 0A 34 4E 6F 6D 65 20   da c.pia..4Nome
0CE1:0260  64 65 20 61 72 71 75 69-76 6F 20 69 6E 76 A0 6C   de arquivo inv.l
0CE1:0270  69 64 6F 20 6F 75 20 61-72 71 75 69 76 6F 20 6E   ido ou arquivo n
0CE1:0280  61 6F 20 65 6E 63 6F 6E-74 72 61 64 6F 0D 0A 1A   ao encontrado...
-
```

V. Programação com EMU8086

A ferramenta **Emu8086** é um simulador gráfico do processador 8086 que também compila os programas escritos em linguagem de programação de computadores *Assembly*.

Ao ser carregado, o programa apresenta um código exemplo escrito em linguagem de programação *Assembly*. O programa traz automaticamente o cabeçalho de identificação de um programa do tipo .COM. Além desse detalhe ele apresenta a mensagem **COM File is loaded at CS:0100h**, avisando que o programa será montado na memória a partir do endereço de deslocamento **0100h** do registrador de segmento **CS**. A Figura 3 apresenta a tela inicial do **Emu8086** com o cabeçalho de um programa do tipo .COM.

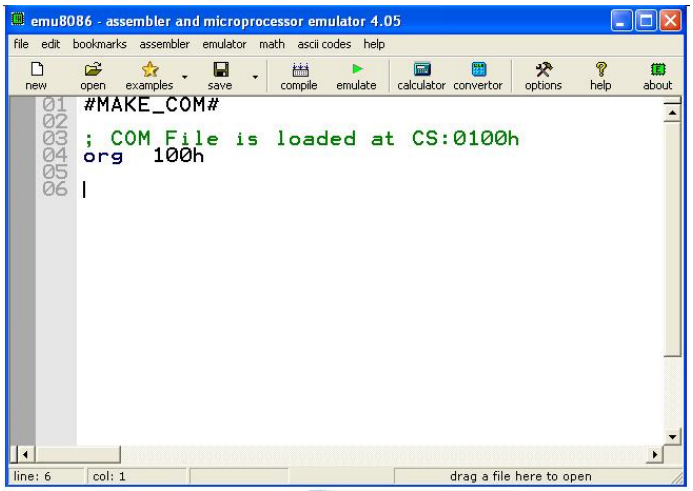

Figura 3. Tela inicial do Emu8086.

Em seguida será desenvolvido um programa que fará a apresentação da mensagem **ALO MUNDO!**, na tela do monitor de vídeo. Nesta etapa o programa será codificado dentro da ferramenta **Emu8086**, e por esta razão terá algumas pequnas difereças em relação ap programa **DEBUG**. Assim sendo, a partir da linha **06**, na Figura 3, escreva o código seguinte:

```
MOV AH, 09H
LEA DX, mensagem
INT 21h
INT 20h
mensagem DB 041h,04Ch,04Fh,020h,04Dh,055h,04Eh,044h,04Fh,021h,024h
```

A Figura 4 apresenta o código anterior definido na tela do editor do programa **Emu8086**. Observe que a sequência de códigos hexadecimais está definida para o rótulo **mensagem** pela diretiva **DB**. O rótulo **mensagem** se assemelha à definição de uma variável em uma linguagem de alto nível. Outro detalhe é o uso da instrução **LEA DX, mensagem** que tem a função de mover os dados definidos no rótulo **mensagem** para o registrador **DX** para que possam ser apresentados na tela.

A partir deste momento, o programa pode ser compilado dentro da ferramenta **Emu8086** ou apenas executado passo a passo. Se for compilado, será criado 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, conforme mostrado na Figura 5.

Emule o programa e execute a ação da tecla **<F8>** e observe a apresentação da mensagem **ALO MUNDO!**, como indicada na Figura 6

Continue clicando na tecla de função **<F8>** até a mensagem de término seja apresentada, como na Figura 7. Para finalizar o processo por completo, acione o botão **OK**.

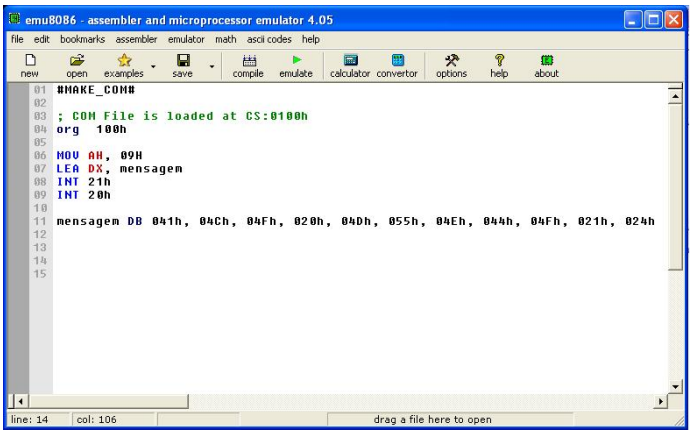

Figura 4. Programa ALO MUNDO!.

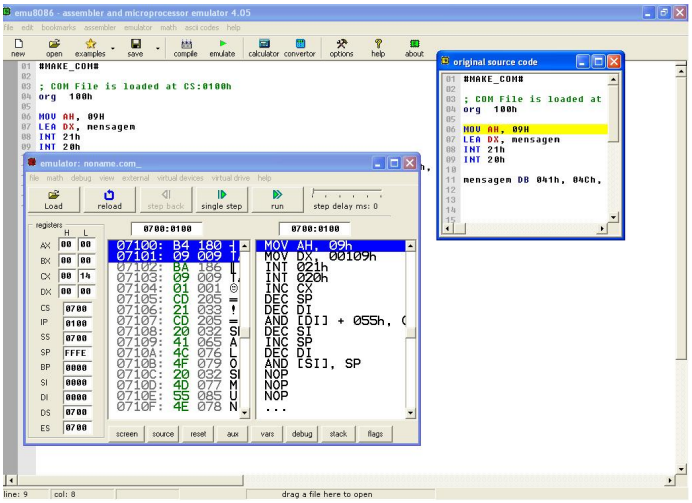

Figura 5. Programa ALO MUNDO! em execução.

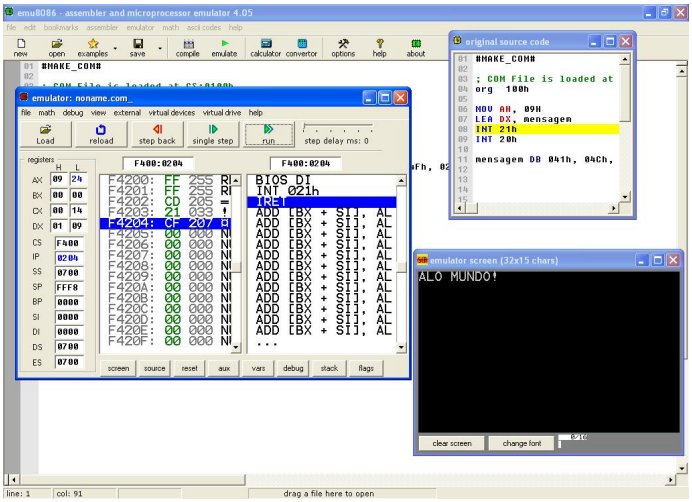

Figura 6. Apresentação da saída do programa.

Figura 7. Término da execução do programa.

VI. Cálculos Matemáticos com EMU8086

A. *Adição*

As operações de adição podem ocorrer com a utilização da instrução **ADD** que tem a mesma função do operador aritmético "+". O funcionamento lógico desta instrução será **ADD DESTINO, ORIGEM**.

```
ADD AX, 5d
```

No exemplo apresentado acima, o registrador **AX** está sendo adicionado com o valor decimal 5. Se no registrador geral **AX** existir algum valor anterior, o valor 5 será somado ao existente.

Tome como base um programa que deva executar a soma de dois valores numéricos que ocupem no máximo 1 *word*. O primeiro valor deve estar associado a uma variável denominada **a**, o segundo valor a uma variável denominada **b** e defini-se uma terceira variável denominada **x** para armazenar o valor da soma. Observe o código a seguir:

```
#MAKE_COM#

ORG 100h

.MODEL small
.STACK 512d

.DATA
 a DW 6d
 b DW 2d
 x DW 0, '$'

 .CODE
    MOV AX, @DATA
    MOV DS, AX

    MOV AX, a
    ADD AX, b
    MOV x, AX

    ADD x,030h
    MOV DX, OFFSET x

    MOV AH, 09h
    INT 021h

    MOV AH, 04h
    INT 021H
```

A diretiva **.MODEL** indica o tipo de modelo de memória que deve ser usado pelo programa. A diretiva **.MODEL**

normalmente deve ser utilizada antes de qualquer definição de segmento de memória, ou seja, deve ser a primeira linha do programa. O parâmetro **small** é a forma mais adequada para a maioria das aplicações, devido à sua rapidez de carga e facilidade depuração. Esse modelo estabelece que todo o código do programa estará em um segmento de memória e os dados estarão no mesmo segmento.

A diretiva **.STACK** tem por finalidade estabelecer a reserva de espaço na pilha do programa. O tamanho da pilha a ser definido depende de alguns fatores com relação à chamada de sub-rotinas, registradores salvos, interrupções e passagens de parâmetros. Normalmente, utiliza-se um tamanho em torno de 512 *bytes* (valor decimal). Desta forma 512 pode ser usado como padrão.

A diretiva **.DATA** define o segmento de dados e é criada três variáveis **a**, **b** e **x**, todas do tipo **DW** com seus respectivos valores decimais 6, 2 e 0. O caractere "**$**"existente após a definição da variável **x** determina o fim da definição de variáveis.

No segmento **.CODE** é definido a sequência de instruções do programa. Logo em seguida é escrito o programa que executa a soma de duas variáveis e apresentada o resultado na tela.

```
MOV AX, @DATA
MOV DS, AX
```

No trecho do código apresentado acima, o endereço memória onde estão armazenadas as variáveis é colocado no registrador de segmento **DS**.

```
MOV AX, a
ADD AX, b
MOV x, AX
```

Neste trecho, o valor da variável **a** é armazenado no registrador geral **AX**, em seguida o valor da variável **b** é somado ao existe neste registrador e posteriormente a soma é transferida para a variável **x**.

Na sequência da execução do código soma-se o valor hexadecimal 30 ao valor de **x** para obter o equivalente ASCII e este possa ser apresentado no monitor de vídeo.

A Figura 8 apresenta o resultado da emulação do programa no **EMU8086**.

B. Subtração

As operações de subtração podem ocorrer com a utilização da instrução **SUB**. O funcionamento lógico desta instrução será **SUB DESTINO, ORIGEM**.

```
SUB AX, 5d
```

No exemplo apresentado acima, o registrador **AX** está sendo subtraído do valor decimal 5. Se no registrador geral **AX** existir algum valor anterior, o valor 5 será subtraído ao existente.

Tome como base um programa que deva executar a equação **x**← **a - b**, em que a variável **a** tem o valor decimal 6 e a

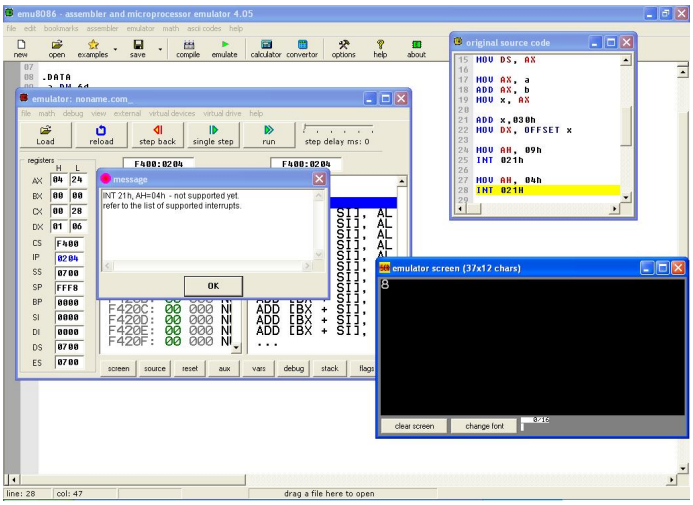

Figura 8. Programa de Adição.

variável **b** o valor decimal 4. Observe o código do programa a seguir:

```
#MAKE_COM#

ORG 100h

.MODEL small
.STACK 512d

.DATA
 a DW 6d
 b DW 4d
 x DW 0, '$'

 .CODE
    MOV AX, @DATA
    MOV DS, AX

    MOV AX, a
    SUB AX, b
    MOV x, AX

    ADD x,030h
    MOV DX, OFFSET x

    MOV AH, 09h
    INT 021h

    MOV AH, 04h
    INT 021H
```

A diferença deste programa em relação ao que executada a operação de adição esta ilustrada no trecho do código abaixo.

```
MOV AX, a
SUB AX, b
MOV x, AX
```

Neste trecho, o valor da variável **a** é copiado para o registrador geral **AX** e em seguinte o valor da variável **b** é subtraído do existente no registrador **AX**. E por fim, o

resultado da operação é copiado para a variável **x**. A emulação, no **EMU8086**, do programa terá aparência semelhante ao da Figura 9.

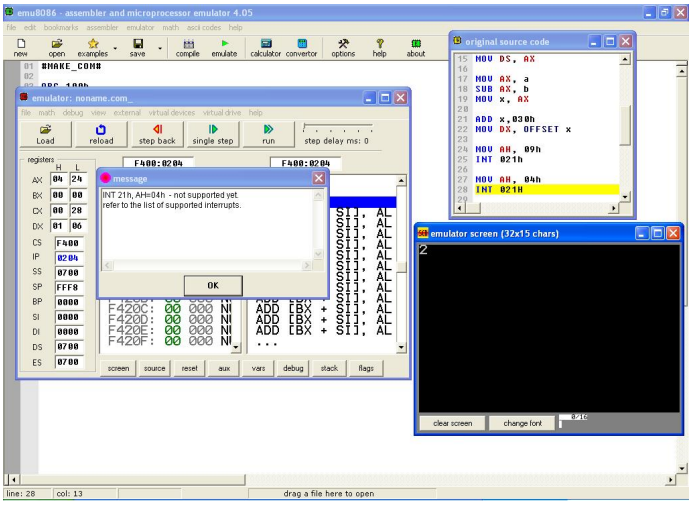

Figura 9. Programa de Subtração.

VII. Laços de Repetição

Outra estrutura de programação muito utilizada e importante são os laços de repetição (*looping*) ou malhas de repetição cuja característica operacional é a capacidade de executar um trecho de programa por um determinado número de vezes. Normalmente um laço de repetição é estabelecido com uso da instrução **LOOP**. O laço de repetição baseado na instrução **LOOP** é iterativo, ou seja, executa uma ação um determinado número de vezes (semelhante ao laço **FOR** em linguagens de alto nível.

É importante mencionar que as instruções de laço usam o valor que estiver armazenado no registrador **CX** para a operação do contador de passos. O valor é sempre subtraído do registrador geral **CX**.

Para exemplificar operações com laços de repetição, considere um programa que apresenta cinco vezes na tela do monitor de vídeo a mensagem **Alo Mundo!**, conforme indicado a seguir:

```
 #MAKE_COM#

ORG 100h

.MODEL small
.STACK 512d

.DATA
 msg DB 'Alo Mundo', 0Dh, 0Ah, 24h

.CODE
  LEA DX,msg
  MOV CX, 5d
  MOV AH, 09h
  laco:
    INT  021h
    LOOP laco
INT 20h
```

O uso dos valores 0Dh, 0Ah, 24h após a definição da mensagem são responsáveis pela definição dos códigos de controle para apresentação de um *string*.

A definição da instrução **MOV CX, 5d** estabelece para o registrador geral **CX** o valor decimal 5, que será automaticamente decrementado em 1 toda vez que a instrução **LOOP** for executada. A Figura 10 apresenta a tela do **EMU8086** após a execução do programa.

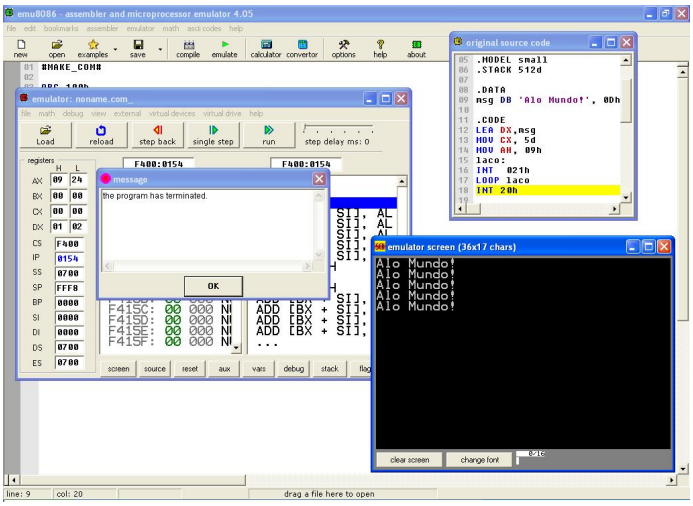

Figura 10. Programa Alo Mundo! com laço de repetição.

VIII. Relatório

Sabendo que o produto entre dois números pode ser obtido por sucessivas operações de adição. Elabore um programa que execute a operação **x←a x b**, sendo a variável **a**=3 e a variável **b**=2. Relate detalhadamente os recursos utilizados na elaboração do código.

Referências

[1] Manzano, José N.G, "Fundamentos em Programação Assembly: para computadores IBM-PC a partir dos microprocessadores intel 8086/8088", 3ª Ed., São Paulo: Érica, 2007.