

```

OD82:0100 B402      MOV     AH, 02
OD82:0102 B241      MOV     DL, 41
OD82:0104 CD21      INT     21
OD82:0106 CD20      INT     20
OD82:0108 69        DB      69

```

# Capítulo 11

## RECURSOS ESPECÍFICOS

Este capítulo aborda as funções da biblioteca externa que acompanha o programa emu8086. São apresentados endereçamento e acesso à memória nos formatos direto, indireto com registrador, indexado, base indexada, base indexada com deslocamento e matrizes.

### 11.1 - Funcionalidades da biblioteca emu8086.inc

Anteriormente viu-se uma rápida descrição das funções existentes na biblioteca **emu8086.inc** encontrada na ferramenta **emu8086**. Cabe neste ponto um estudo mais detalhado de cada função em particular.

#### 11.1.1 - Funções de macro

A biblioteca **emu8086.inc** apresenta seis macros que oferecem os seguintes recursos operacionais:

##### **PUTC caractere**

Função que apresenta como saída o caractere informado como parâmetro na posição atual do cursor. O caractere pode ser um valor em notação binária, decimal, octal ou hexadecimal correspondente a um valor da tabela ASCII. Pode também ser informado o caractere desejado entre aspas ou apóstrofos.

```

;*****+
;*  Programa: BIBLIOT2.ASM   *
;*****+
INCLUDE 'emu8086.inc'

.MODEL small
.STACK 512d

.CODE

PUTC    65d
PUTC    154o
PUTC    06Fh
PUTC    00100000b
PUTC    "M"
PUTC    165o
PUTC    110d
PUTC    064h
PUTC    'o'
INT     020h
END

```

O programa anterior exibe a mensagem **Alo Mundo**, constituída das diversas formas de definição de caracteres ASCII para a sua composição. Salve o programa com o nome **BIBLIOT2.asm**.

### GOTOXY coluna, linha

Função que coloca o cursor em uma determinada coordenada de tela. É necessário levar em consideração que a tela em modo texto padrão tem 80 colunas (numeradas de 0 até 79) e 24 linhas (numeradas de 0 até 23). No momento de definir a posição, é necessário levar em consideração o valor da posição em si. Para posicionar o cursor na coluna um, linha um, devem ser utilizadas as coordenadas zero e zero.

```
;*****  
;*      Programa: BIBLIOT3.ASM      *  
;*****  
  
INCLUDE 'emu8086.inc'  
  
.MODEL small  
.STACK 512d  
  
.CODE  
  
GOTOXY    0d, 0d  
PUTC      'A'  
GOTOXY    1d, 1d  
PUTC      'l'  
GOTOXY    2d, 2d  
PUTC      'o'  
GOTOXY    3d, 3d  
PUTC      '  
GOTOXY    4d, 4d  
PUTC      'M'  
GOTOXY    5d, 5d  
PUTC      'u'  
GOTOXY    6d, 6d  
PUTC      'n'  
GOTOXY    7d, 7d  
PUTC      'd'  
GOTOXY    8d, 8d  
PUTC      'o'  
INT       20h  
END
```

O programa anterior apresenta a mensagem **Alo Mundo** num formato diagonal. Salve-o com o nome **BIBLIOT3.asm**.

### PRINT mensagem/PRINTN mensagem

Funções que exibem uma mensagem na tela do monitor de vídeo. A rotina **PRINT** apresenta uma mensagem e mantém o cursor na mesma linha. A rotina **PRINTN** apresenta uma mensagem e movimenta o cursor para a próxima linha.

```
;*****  
;*      Programa: BIBLIOT4.ASM      *  
;*****  
  
INCLUDE 'emu8086.inc'  
  
.MODEL small  
.STACK 512d
```

```

.CODE

PRINTN    'Alo Mundo 1'
PRINTN    'Alo Mundo 2'
PRINT     'Alo Mundo 3'
PRINT    'Alo Mundo 4'
INT      20h

```

END

O programa anterior mostra quatro mensagens com **Alo Mundo**. A primeira e a segunda linhas aparecem uma abaixo da outra num formato diagonal. A mensagem da terceira linha de programa é apresentada abaixo da segunda mensagem, mas a quarta mensagem surge imediatamente ao lado da terceira mensagem. Salve o programa com o nome **BIBLIOT4.asm**.

### CURSOROFF/CURSORON

Funções que ocultam ou apresentam o cursor. A rotina **CURSOROFF** desabilita a apresentação do cursor. A rotina **CURSORON** habilita a apresentação do cursor.

```

;*****
;*  Programa: BIBLIOT5.ASM   *
;*****

INCLUDE 'emu8086.inc'

.MODEL small
.STACK 512d

.CODE

CURSOROFF
PRINTN  'Sem cursor - tecle algo'

MOV     AH, 00h
INT     16h

CURSORON
PRINTN  'Com Cursor - tecle algo'

MOV     AH, 00h
INT     16h

INT     20h

END

```

O programa anterior apresenta duas mensagens. A primeira não mostra o cursor na tela. A segunda mensagem apresenta o cursor. Salve o programa com o nome **BIBLIOT5.asm**.

## 11.1.2 - Funções de procedimento

A biblioteca **emu8086.inc** apresenta sete procedimentos que oferecem os seguintes recursos operacionais:

### GET\_STRING/PRINT\_STRING

O procedimento **GET\_STRING** lê via teclado uma sequência de caracteres terminada em nulo. A sequência de caracteres é escrita no *buffer* utilizando o par de registradores **DS:DI** e armazena o tamanho do *buffer* no registrador geral **DX**. Esse procedimento é interrompido quando usada a tecla <Enter>. Antes da diretiva **END** no programa principal é necessário utilizar a declaração **DEFINE\_GET\_STRING** para que o funcionamento da rotina **GET\_STRING** ocorra sem problemas.

O procedimento **PRINT\_STRING** apresenta uma sequência de caracteres terminada com caractere nulo na posição atual do cursor. A mensagem a ser impressa é obtida a partir do *buffer* identificado pelo par de registradores **DS:SI**. Antes da diretiva **END** no programa principal é necessário utilizar a declaração **DEFINE\_PRINT\_STRING** para que a rotina **PRINT\_STRING** funcione sem problemas.

```
;*****  
;*      Programa: BIBLIOT6.ASM      *  
;*****  
  
INCLUDE 'emu8086.inc'  
  
org 100h  
  
.DATA  
tamanho EQU 30d + 1d  
buffer DB tamanho DUP ('x')  
msg1 DB 'Entre seu nome: ', 0  
msg2 DB 'Ola, ', 0  
  
.CODE  
LEA SI, msg1  
CALL PRINT_STRING  
  
LEA DI, buffer  
MOV DX, tamanho  
CALL GET_STRING  
PUTC 13d  
PUTC 10d  
  
LEA SI, msg2  
CALL PRINT_STRING  
  
MOV SI, DI  
CALL PRINT_STRING  
  
INT 20h  
  
DEFINE_PRINT_STRING  
DEFINE_GET_STRING  
END
```

O programa anterior apresenta na área de definição de dados (**.DATA**) duas novidades. A primeira é o uso da diretiva **EQU** que permite a definição de constantes, que neste caso está sendo representada pelo rótulo denominado **tamanho**. A segunda novidade é com relação ao valor da constante que está definido como **30d + 1d**. Observe o uso do operador de adição entre os dois valores para determinar o valor **31d**. Certamente pode ser definido diretamente o valor **31d**, mas há possibilidade de estabelecer valores com operadores aritméticos, como foi feito. Salve o programa com o nome **BIBLIOT6.asm**.

Ainda na área de definição de dados (.DATA) há a definição da variável **buffer** do tipo **DB** com o valor estabelecido para a constante **tamanho**. Está sendo estabelecido na memória um espaço para a variável denominada **buffer** com tamanho **31d** contendo uma série de caracteres **x**.

Na área de código (.CODE) o programa apresenta em tela a mensagem **Entre seu nome:**, manipulada pelo procedimento **PRINT\_STRING**, pelas linhas de código **LEA SI, msg1** e **CALL PRINT\_STRING**.

Em seguida o programa solicita o nome pela linha de código **CALL GET\_STRING**. Nessa etapa acompanhe a definição do deslocamento da área de **buffer** através da linha de código **LEA DI, buffer** e do armazenamento do tamanho da área de **buffer** através da linha de código **MOV DX, tamanho**. Está sendo transferido o endereço do conteúdo informado pelo teclado para o par de registradores **DS:DI**. Depois o programa retorna o carro e muda de linha com a instrução **PUTC 13d** e **PUTC 10d**.

Após a execução das etapas anteriores o programa exibe a mensagem armazenada na variável **msg2** e resgata dos registradores **DS:DI** o endereço em que se encontra a mensagem entrada pelo procedimento **GET\_STRING**.

Antes da diretiva **END** que finaliza o código do programa estão definidos os rótulos **DEFINE\_PRINT\_STRING** e **DEFINE\_GET\_STRING**, que devem ser utilizados antes da diretiva **END**, pois representam as rotinas de macros internas que estabelecem para o procedimento em uso os rótulos locais usados pelos procedimentos **GET\_STRING** e **PRINT\_STRING**.

### PTHIS

O procedimento **PTHIS** exibe a mensagem terminada com caractere nulo na posição atual do cursor, semelhante ao procedimento **PRINT\_STRING**, mas com a diferença de receber o endereço da sequência de caracteres que formam a mensagem a partir da pilha. Para sequências de caracteres terminadas em zero, é necessário que a mensagem ocorra logo após a chamada do procedimento. Antes da diretiva **END** no programa principal é necessário utilizar a declaração **DEFINE\_PTHIS** para que o funcionamento da rotina **PTHIS** ocorra sem problemas.

```
;*****  
;*      Programa: BIBLIOT7.ASM      *  
;*****  
  
INCLUDE 'emu8086.inc'  
  
org 100h  
  
.DATA  
    tamanho EQU 30d + 1d  
    buffer DB tamanho DUP ('x')  
  
.CODE  
  
    CALL pthis  
    DB 'Entre seu nome: ', 0  
  
    LEA    DI, buffer  
    MOV    DX, tamanho  
    CALL   GET_STRING  
    PUTC  13d  
    PUTC  10d  
  
    CALL pthis  
    DB 'Ola, ', 0  
  
    MOV    SI, DI  
    CALL  PRINT_STRING  
  
    INT    20h
```

```

DEFINE_PRINT_STRING
DEFINE_GET_STRING
DEFINE_PTHIS
END

```

As chamadas são efetuadas no código anterior a partir da instrução **CALL pthis**. É definida a mensagem a ser apresentada imediatamente após a chamada do procedimento **pthis**. Salve o programa com o nome **BIBLIOT7.asm**.

### CLEAR\_SCREEN

Procedimento que limpa a tela e posiciona o cursor no seu topo esquerdo. Antes da diretiva **END**, no programa principal é necessário utilizar a declaração **DEFINE\_CLEAR\_SCREEN**.

Para esse tipo de rotina de procedimento não será apresentado um exemplo de programa, pois esse efeito será mostrado no próximo exemplo.

### SCAN\_NUM/PRINT\_NUM/PRINT\_NUM\_UNS

O procedimento **SCAN\_NUM** obtém um valor numérico com mais de um dígito fornecido pelo teclado, podendo ser um valor positivo ou negativo. Esse procedimento armazena o resultado da entrada no registrador geral **CX**. Antes da diretiva **END** no programa principal é necessário utilizar a declaração **DEFINE\_SCAN\_NUM** para que a rotina **SCAN\_NUM** funcione sem problemas.

Os procedimentos **PRINT\_NUM** e **PRINT\_NUM\_UNS** apresentam, respectivamente, os valores numéricos negativos e positivos existentes no registrador geral **AX**. Antes da diretiva **END** no programa principal é necessário utilizar as declarações **DEFINE\_PRINT\_NUM** e **DEFINE\_PRINT\_NUM\_UNS** para que as rotinas de procedimento **PRINT\_NUM** e **PRINT\_NUM\_UNS** funcionem sem problemas.

```

;*****
;*      Programa: BIBLIOT8.ASM      *
;*****

INCLUDE 'emu8086.inc'

org 100h

.DATA
tamanho EQU 30d + 1d
idade DW 0
buffer DB tamanho DUP ('x')
msg1 DB 'Entre seu nome ....: ', 0
msg2 DB 'Entre sua idade ...: ', 0
msg3 DB 'Ola, ', 0
msg4 DB ' voce tem ', 0
msg5 DB ' anos.', 0

.CODE
LEA SI, msg1
CALL PRINT_STRING
LEA DI, buffer
MOV DX, tamanho
CALL GET_STRING
PUTC 13d
PUTC 10d

LEA SI, msg2
CALL PRINT_STRING
CALL SCAN_NUM
MOV idade, CX
PUTC 13d
PUTC 10d

```

```

CALL    CLEAR_SCREEN
LEA     SI, msg3
CALL    PRINT_STRING
MOV     SI, DI
CALL    PRINT_STRING
LEA     SI, msg4
CALL    PRINT_STRING
MOV     AX, idade
CALL    PRINT_NUM
LEA     SI, msg5
CALL    PRINT_STRING

INT    20h

```

```

DEFINE_PRINT_STRING
DEFINE_GET_STRING
DEFINE_SCAN_NUM
DEFINE_CLEAR_SCREEN
DEFINE_PRINT_NUM
DEFINE_PRINT_NUM_UNS
END

```

Observe no programa anterior as linhas de código **CALL SCAN\_NUM** e na sequência **MOV idade, CX**. Assim que o valor é capturado e armazenado na memória (registraror geral **CX**), ele é transferido para a variável **idade**. Para fazer a apresentação do valor da variável **idade**, o programa executa as linhas de código **MOV AX, idade** e **CALL PRINT\_NUM**. Salve o programa com o nome **BIBLIOT8.asm**.

## 11.2 - Endereçamento e acesso à memória

Anteriormente foi feito um rápido comentário a respeito das formas de acesso e uso de endereçamento de memória. Neste capítulo, devido à necessidade de estudar tabelas em memória (matrizes), é preciso aprofundar este tema.

Segundo o exposto por Hyde (2003), existem dezessete formas de acesso à memória do microprocessador 8086/8088, sendo uma de acesso de deslocamento direto, quatro de acesso com o modo de endereçamento indireto com registradores, quatro de acesso com o modo de endereçamento indexado, quatro de acesso de endereçamento de base indexada e quatro de acesso de endereçamento de base indexada mais deslocamento. Observe a Tabela 11.1.

Tabela 11.1 - Definição do Endereçamento de Memória

| Direto       | Indireto com Registrador     | Indexado   | Base Indexada                                | Base Indexada mais Deslocamento  |
|--------------|------------------------------|--|--|--|
| ds:[posição] | [BX]<br>[BP]<br>[SI]<br>[DI] | desloc[BX]<br>desloc[BP]<br>desloc[SI]<br>desloc[DI] | [BX][SI]<br>[BX][DI]<br>[BP][SI]<br>[BP][DI] | desloc[BX][SI]<br>desloc[BX + DI]<br>[BP + SI + desloc]<br>[BP][DI][x] |

A indicação do rótulo **desloc** na tabela refere-se à definição de um valor de deslocamento a ser informado. A tabela anterior apresenta as formas válidas de definição de deslocamentos de memória para a manipulação de dados que podem ser utilizadas em um programa escrito em linguagem de programação **Assembly** para um microprocessador 8086/8088.

Para os casos de acesso a endereçamento de memória não direto, Hyde (2003) sugere um diagrama para facilitar a memorização das estruturas de endereçamento: indireto com registrador, indexado, base indexada e base indexada mais deslocamento, como é indicado na Figura 11.1.

|              |              |              |
|--------------|--------------|--------------|
| Deslocamento | [BX]<br>[BP] | [SI]<br>[DI] |
|--------------|--------------|--------------|

Figura 11.1 - Endereçamento de memória (Adaptado de Hyde, 2003).

Para saber o endereçamento de memória correto, basta escolher pelo menos um item de cada coluna para ter o modo de endereçamento de memória válido. São válidos os seguintes endereçamentos de memória:

- ◆ Escolhe-se o deslocamento da coluna da esquerda, não se escolhe o registrador [BX] da coluna do meio, escolhe-se o registrador [DI] da coluna da direita e obtém-se o endereçamento **deslocamento[DI]**.
- ◆ Escolhe-se o deslocamento da coluna da esquerda, o registrador [BX] da coluna do meio, escolhe-se o registrador [DI] da coluna da direita e obtém-se o endereçamento **deslocamento[BX][DI]**.
- ◆ Não se escolhe o deslocamento da coluna da esquerda, não se escolhe o registrador [BX] da coluna do meio, escolhe-se o registrador [SI] e obtém-se o endereçamento **[SI]**.
- ◆ Não se escolhe o deslocamento da coluna da esquerda, escolhe-se o registrador [BX] da coluna do meio, o registrador [DI] da coluna da direita e obtém-se o endereçamento **[BX][DI]**.

De acordo com o exposto, qualquer combinação de endereçamento diferente da forma permitida pelo diagrama indicado na Figura 11.1 é considerado inválido. Hyde (2003) apresenta como endereçamento inválido de memória a definição **deslocamento[DX][SI]**, pois o registrador [DX] não está definido na coluna do meio do diagrama.

É pertinente lembrar que as operações de acesso à memória são efetuadas normalmente com a instrução **MOV** e dependendo do programa **assemblador** em uso, é necessário utilizar para algumas operações a instrução **LEA**.

A partir da definição dos princípios de formas de acesso aos endereçamentos de memória, cabe ressaltar os pontos de conceituação apontados por Hyde (2003) em seu trabalho, identificados a seguir.

## 11.2.1 - Acesso direto

O modo de acesso direto ao endereçamento de memória é baseado no uso de uma constante de 8 bits, ou seja, uma constante de um byte, que representa o valor de acesso a um determinado endereço de memória do segmento de dados em uso.

Por exemplo, a instrução **MOV AL, DS:[1234h]** transfere para o registrador menos significativo **AL** uma cópia do valor do byte residente no endereço de deslocamento **1234h** do segmento **DS**. O registrador menos significativo **AL** passa a ter o valor armazenado no endereço de memória **1234h**. É possível ter acesso ao conteúdo de memória armazenado no endereço de deslocamento **1234h**, como mostra a Figura 11.2.

Em contrapartida, se fosse usada a instrução **MOV DS:[1234h], AL**, seria armazenado o valor do registrador menos significativo **AL** no endereço de memória **1234h**, como indica a Figura 11.3.

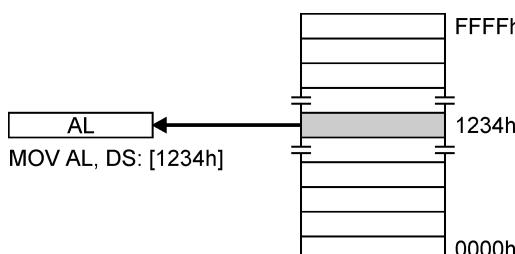


Figura 11.2 - Endereçamento por deslocamento (Adaptado de Hyde, 2003).

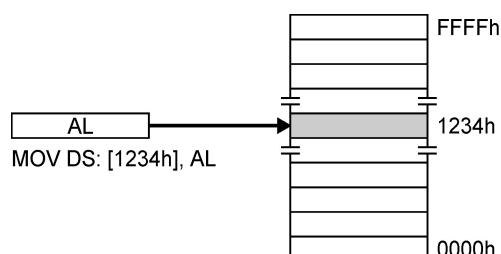


Figura 11.3 - Endereçamento por deslocamento (Adaptado de Hyde, 2003).

É importante levar em consideração que essa forma de acesso é muito utilizada com valores simples. Para valores em posições indexadas (matrizes) é necessário utilizar outras formas de acesso à memória.

Para exemplificar uma ação de acesso direto à memória de forma simples, considere o seguinte programa:

```

;*****
;*      Programa: MEMO1.ASM      *
;*****

.MODEL small
.STACK 512d

.CODE

MOV AL, 5d
MOV DS:[010Ch], AL
SUB AX, AX
MOV AL, DS:[010Ch]

HLT

END

```

Execute no programa **emu8086** o comando de menu **file/new/com template**, ação as teclas de atalho **<Ctrl> + <A>** do editor de texto e escreva o programa anterior, gravando-o por meio dos comandos de menu **File/Save** com o nome **MEMO1**, de forma que fique semelhante à imagem da Figura 11.4.

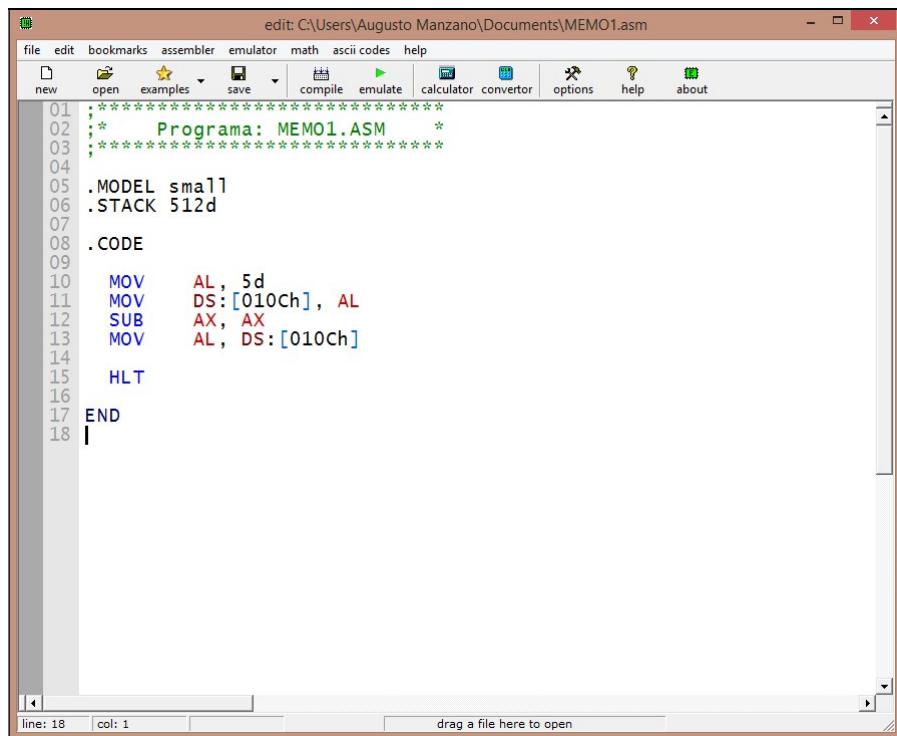


Figura 11.4 - Programa MEMO1 na ferramenta emu8086.

Antes de qualquer comentário a respeito da ação principal do programa, observe a instrução **HLT** (*halt*) de encerramento do programa. Ela para a execução do programa, interrompendo o processador.

Em seguida peça a execução do programa por meio da tecla de função **<F5>** e observe atentamente os detalhes na janela **emulator: MEMO1.exe**, conforme a Figura 11.5.

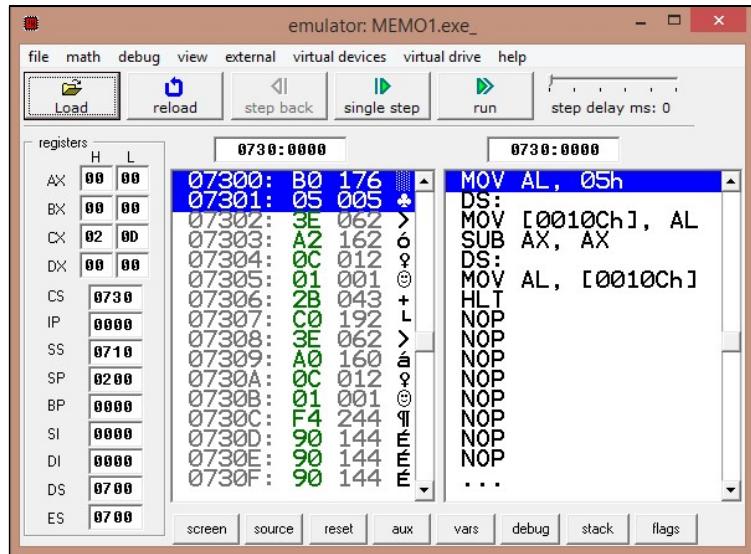


Figura 11.5 - Janela Emulator: MEMO1.exe\_ na ferramenta emu8086.

Perceba a apresentação em separado da área **disassembler** (lado direito), Figura 11.6, com a definição do código do programa escrito em **Assembly** e da área **memory** (área central), Figura 11.7, com a definição do código em linguagem de máquina (**opcodes**).

```
MOV AL, 05h
DS:
MOV [0010Ch], AL
SUB AX, AX
DS:
MOV AL, [0010Ch]
HLT
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
...
...
```

Figura 11.6 - Código em Assembly.

|        |    |     |
|--------|----|-----|
| 07300: | B0 | 176 |
| 07301: | 05 | 005 |
| 07302: | 3E | 062 |
| 07303: | A2 | 162 |
| 07304: | 0C | 012 |
| 07305: | 01 | 001 |
| 07306: | 2B | 043 |
| 07307: | C0 | 192 |
| 07308: | 3E | 062 |
| 07309: | A0 | 160 |
| 0730A: | 0C | 012 |
| 0730B: | 01 | 001 |
| 0730C: | F4 | 244 |
| 0730D: | 90 | 144 |
| 0730E: | 90 | 144 |
| 0730F: | 90 | 144 |

Figura 11.7 - Código em Opcode.

Na Figura 11.6 o código apresentado é ligeiramente diferente do código-fonte definido no editor do programa **emu8086**. As instruções **MOV DS:[010Ch], AL** e **MOV AL, DS:[010Ch]** são indicadas na área de código, Figura 11.6, respectivamente, como **MOV [0010Ch], AL** e **MOV AL, [0010Ch]**. O registrador **DS** fará referência ao endereço de segmento que estiver em uso pelo microprocessador. Basta acionar pela primeira vez a tecla de função <F8> para proceder à execução passo a passo do programa. A Figura 11.8 mostra as alterações na janela **emulator: MEMO1.exe\_**.

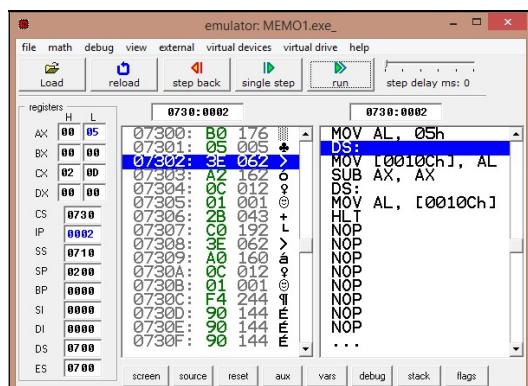


Figura 11.8 - Janela Emulator: MEMO1.exe\_ (primeira execução de <F8>).

Ocorre a definição do valor **05h** para o registrador menos significativo **AL**, como indica a Figura 11.8. Em seguida acione a tecla de função **<F8>** pela segunda vez para continuar a ação do programa. A Figura 11.9 exibe a mudança do valor do registrador **IP** de **0002h** para **0003h**.

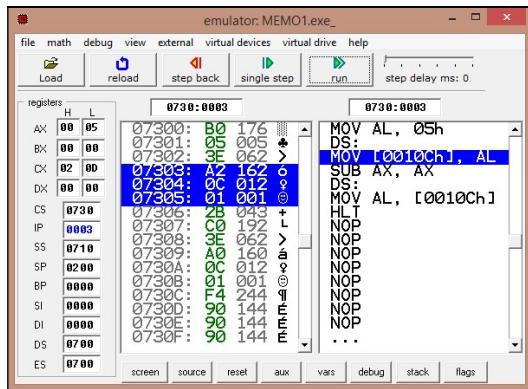


Figura 11.9 - Janela MEMO1.exe\_ (segunda execução de <F8>).

No momento, acione na tela **emulator: MEMO1.exe** a opção de menu **view/memory** para que seja apresentada a tela **Random Access Memory**. No campo existente do lado esquerdo superior, ao lado do botão **update**, entre o endereço de memória **0700:010C** no formato **segmento:deslocamento** (é importante lembrar que o endereço de segmento apresentado no computador do leitor pode ser diferente, esteja atento). Como se deduziu o segmento **0700h**? Simples, o programa iniciou sua execução no endereço de segmento **0730h** e como se está utilizando o conceito de acesso direto, este é feito num endereço de memória do segmento de dados em uso. Isto posto, percebe-se que o programa está operando no endereço de segmento **0730h**, desta forma o microprocessador direciona o armazenamento do dado para o início do segmento em operação que para este caso é o endereço de segmento **0700h**. A Figura 11.10 mostra a janela **Random Access Memory** posicionada no endereço **0700:010C**.

Observe na Figura 11.10 que a área de memória de **0700:010C** até **0700:017C** possui armazenado o conteúdo de código **54h** que já estava na memória.

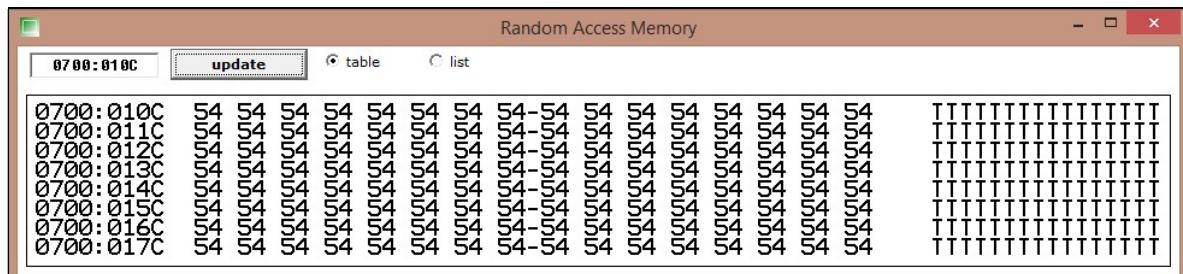


Figura 11.10 - Janela Random Access Memory no endereço 0730:010C.

Em seguida acione a tecla de função **<F8>** pela terceira vez e observe a mudança do registrador **IP** para **0006h** e o posicionamento sobre a instrução **SUB AX, AX** para zerar o registrador **AX**. Note também o armazenamento do valor **05h** na primeira posição da janela **Random Access Memory** como na Figura 11.11.

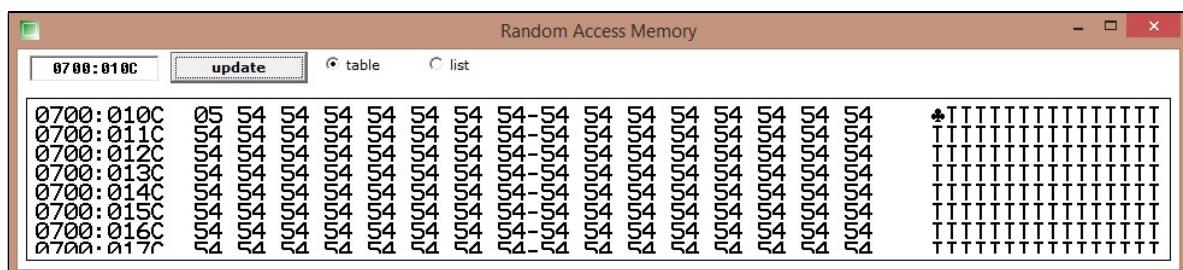


Figura 11.11 - Janela Random Access Memory no endereço 0730:010C com valor 05h.

A partir desse instante o registrador geral **AX** será zerado pela instrução **SUB AX, AX**, como indicado na Figura 11.12, e a próxima instrução a ser executada é a preparação para a transferência de volta do valor **05h** que se encontra armazenado no endereço de memória **0700:010C** para o registrador menos significativo **AL**. Lembre-se de que esse armazenamento foi efetivado pela linha de instrução **MOV DS:[010Ch], AL**. Em seguida, acione a tecla de função <F8> pela quarta vez e veja o resultado da operação na Figura 11.13.

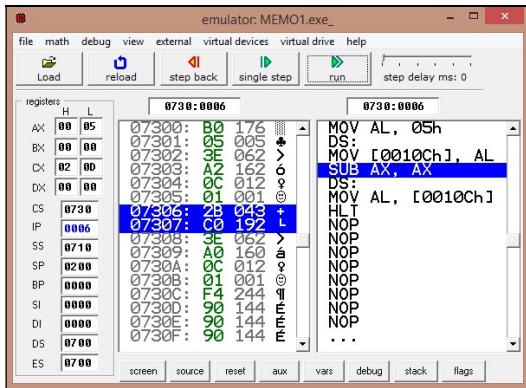


Figura 11.12 - Janela Emulator: MEMO1.exe\_ (terceira execução de <F8>).

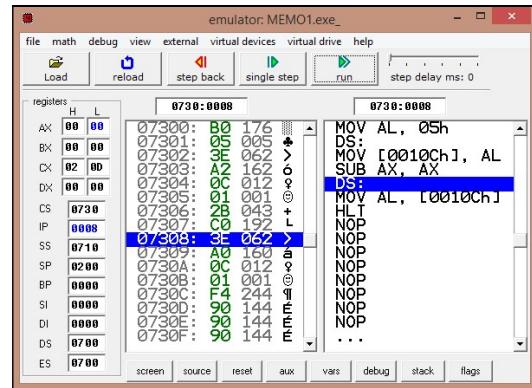


Figura 11.13 - Janela Emulator: MEMO1.exe\_ (quarta execução de <F8>).

Acione a tecla de função <F8> pela quinta vez. A execução do programa estará sobre a instrução **MOV AL, [0010Ch]** que efetua a transferência do valor **05h** armazenado no endereço **0700:010C** de volta para o registrador **AL**, como indicado na Figura 11.14. Em seguida acione a tecla de função <F8> pela sexta vez e observe a mudança do valor do registrador **AL** de **00h** para **05h** e o posicionamento sobre a instrução **HLT**, como mostra a Figura 11.15.

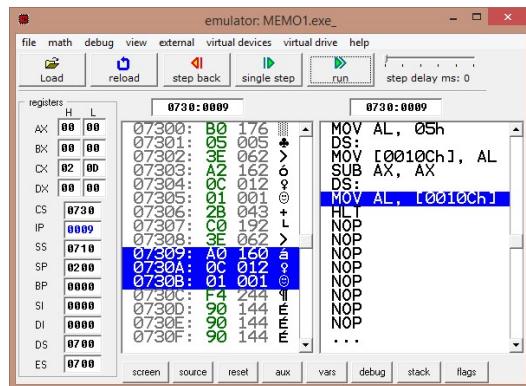


Figura 11.14 - Janela Emulator: MEMO1.exe\_ (quinta execução de <F8>).

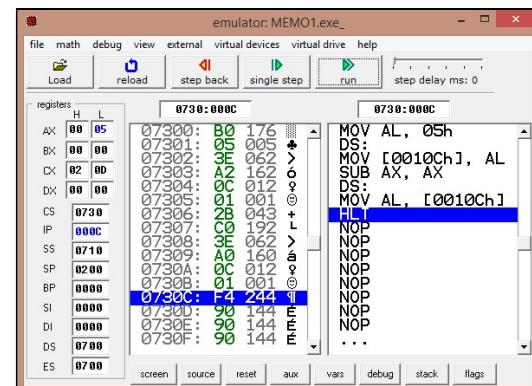


Figura 11.15 - Janela Emulator: MEMO1.exe\_ (sexta execução de <F8>).

Pela sétima e última vez acione a tecla de função <F8>, com o ponteiro do mouse selecione o botão **OK** da caixa de mensagem **message** e na janela **emulator: MEMO1.exe\_** selecione o comando de menu **file/close the emulator**.

## 11.2.2 - Acesso indireto com registradores

O modo de acesso indireto com registrador permite obter o endereço de uma determinada posição de memória com a utilização de um dos registradores **[BX]**, **[BP]**, **[DI]** e **[SI]**. Como padrão, os registradores **[BX]**, **[DI]** e **[SI]** usam como registrador de segmento o **DS** e o **[BP]** utiliza como registrador de segmento o **SS**.

Por exemplo, a instrução **MOV AL, [BX]** transfere para o registrador menos significativo **AL** uma cópia do byte residente no endereço de deslocamento identificado pelo registrador **[BX]** do segmento **DS**. Desta forma, o registrador menos significativo **AL** passa a ter o valor armazenado no endereço de memória identificado por **[BX]**, como mostra a Figura 11.16.

Se usada a instrução **MOV AL, valor[BX]** o registrador menos significativo **AL** será armazenado com os valores existentes da área de dados identificada pela variável **valor**, apontada pelo registrador de segmento **DS** até a posição limite

(a ser definida no programa) a ser obtida com o uso do registrador [BX]. Para utilizar esse recurso, é preciso definir um laço que desloque o endereço a partir de uma posição inicial. Vá até um valor que determine a posição final de memória. A informação [BX] da Figura 11.16 também pode ser entendida como [DI] e [SI].

Para utilizar o registrador de pilha [BP], deve ser considerado como registrador de segmento o **SS**. A Figura 11.17 apresenta a estrutura de funcionamento desse tipo de operação.

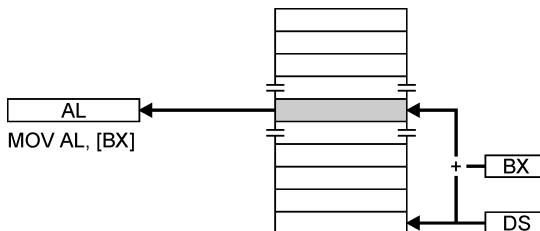


Figura 11.16 - Endereçamento indireto com registro (Adaptado de Hyde, 2003).

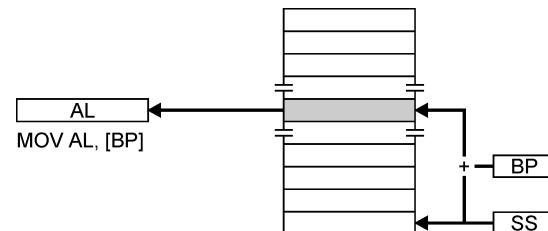


Figura 11.17 - Endereçamento indireto com registro (Adaptado de Hyde, 2003).

Para exemplificar a ação de acesso indireto com registradores de memória de forma simples, considere o seguinte programa:

```
;*****  
;*      Programa: MEMO2.ASM      *  
;*****  
  
.org 100h  
  
.DATA  
    valor DB 05d, 04d, 03d, 02d, 01d  
  
.CODE  
    MOV CX, 05h  
    LEA BX, valor  
laco:  
    MOV AL, [BX]  
    ADD AL, 30h  
    MOV AH, 0Eh  
    INT 10h  
    INC BX  
    LOOP laco  
    HLT  
  
END
```

Execute no programa **emu8086** o comando de menu **file/new/com template**, acione as teclas de atalho **<Ctrl> + <A>** do editor de texto e escreva o programa anterior, gravando-o por meio dos comandos de menu **file/save** com o nome **MEMO2**, de forma que fique semelhante à imagem da Figura 11.18.

Observe o trecho de código definido entre o rótulo **laco:** e a instrução **LOOP laço** e a utilização das instruções **MOV AL, [BX]** e **INC BX**. A instrução **MOV AL, [BX]** movimenta um dos conteúdos identificados na área de dados com a variável **valor**. À medida que se obtém o conteúdo de memória e sua respectiva apresentação, o programa acrescenta 1 ao valor existente no registrador **[BX]**. Desta forma é possível obter todos os valores definidos na variável **valor**.

Em seguida peça a execução do programa pela tecla de função **<F5>** e acione também uma vez a tecla de função **<F8>**. Observe os detalhes apresentados na janela **MEMO2**, como indica a Figura 11.19.

Observe atentamente as informações apresentadas nas áreas **disassemble** e **memory**. Atente para o posicionamento sobre a instrução do programa **MOV CX, 00005h** e o posicionamento no endereço de memória **0700:0107**.

A tabela seguinte apresenta um paralelo entre o código escrito no editor e o código assumido pelo simulador. Ocorrem algumas pequenas mudanças, entre elas a substituição da instrução **LEA BX, valor** pela instrução **MOV BX, 00102h**, indicando o endereço de deslocamento que marca o início da área de dados denominada **valor**.

```

01 ;***** Programa: MEMO2.ASM *****
02 ;
03 ;
04
05 org 100h
06
07 .DATA
08     valor DB 05d, 04d, 03d, 02d, 01d
09
10 .CODE
11     MOV CX, 05h
12     LEA BX, valor
13     laco:
14         MOV AL, [BX]
15         ADD AL, 30h
16         MOV AH, 0Eh
17         INT 10h
18         INC BX
19     LOOP laco
20     HLT
21
22 END
23

```

Figura 11.18 - Programa MEMO2 na ferramenta emu8086.

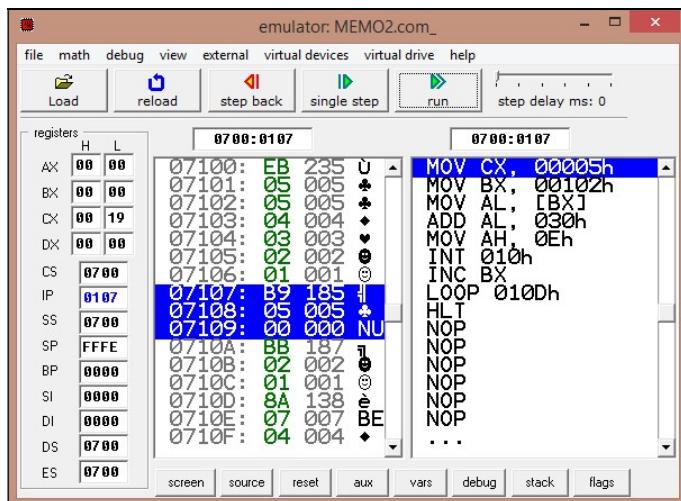


Figura 11.19 - Janela MEMO2 na ferramenta emu8086 (<F8> - primeira vez).

A linha de código **MOV AL, [BX]** transfere o valor apontado pelo registrador **[BX]** para o registrador menos significativo **AL**. Em seguida esse valor é apresentado na tela do monitor de vídeo. O registrador **[BX]** possibilita obter o primeiro valor da área de dados (**valor**), que está armazenado na posição **0** (zero). Por este motivo, antes da instrução **LOOP** se encontra a instrução **INC BX**, que acrescenta **1** (um) ao valor atual do registrador **[BX]**. A Tabela 20.2 demonstra a ocorrência interna dos valores do registrador **[BX]**.

Em vez de acionar a tecla de função **<F8>**, utilize em conjunto as teclas de atalho **<Shift> + <F8>** para executar o programa a partir da instrução **MOV CX, 00005h** que enviará o valor **0005h** para o registrador **CX**, como mostra a Figura 11.20. O valor armazenado no registrador **CX** será utilizado para controlar a ação da instrução **LOOP**.

Tabela 11.2 – Ocorrência de valores do registrador [BX]

| Código no Editor |  | Código no Simulador |
|------------------|--|---------------------|
| MOV CX, 05h      |  | MOV CX, 00005h      |
| LEA BX, valor    |  | MOV BX, 00102h      |
| laco:            |  |                     |
| MOV AL, [BX]     |  | MOV AL, [BX]        |
| ADD AL, 30h      |  | ADD AL, 030h        |
| MOV AH, 0Eh      |  | MOV AH, 0Eh         |
| INT 10h          |  | INT 010h            |
| INC BX           |  | INC BX              |
| LOOP laco        |  | LOOP 010Dh          |
| HLT              |  | HLT                 |

Em seguida, execute pela terceira vez as teclas de atalho <Shift> + <F8> para que a instrução **MOV BX, 00102h** seja processada. A Figura 11.21 mostra a etapa atual de execução em que o registrador **BX** assume o valor **0102h**.

Agora acione as teclas de atalho <Shift> + <F8> pela quarta vez para se posicionar no próximo passo. A Figura 11.22 mostra o resultado da ação. Note o posicionamento na instrução **ADD AL, 030h** e a alteração do valor do registrador **AL** com o valor de **[BX]** que neste momento é **05h**.

Pela quinta vez acione as teclas de atalho <Shift> + <F8> e observe o envio do valor **30h** para o registrador **AL**, como mostra a Figura 11.23. Lembre-se de que o procedimento de somar **30h** ao conteúdo do registrador **AL** que está com valor **05h** é tornar esse valor **35h** que é o código ASCII do valor **5d** para ser, na sequência, apresentado na tela do monitor de vídeo.

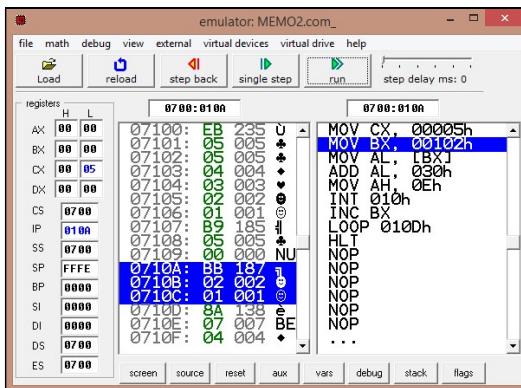


Figura 11.20 - Janela MEMO2 na ferramenta emu8086 (<Shift> + <F8> - segunda vez).

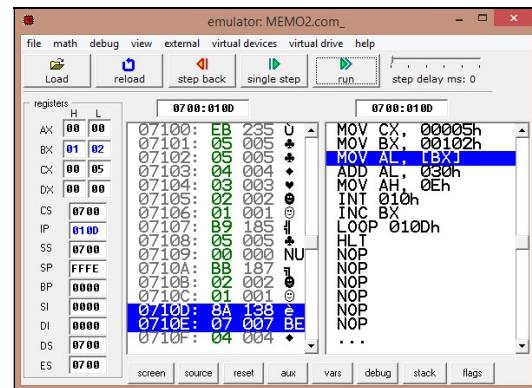


Figura 11.21 - Janela MEMO2 na ferramenta emu8086 (<Shift> + <F8> - terceira vez).

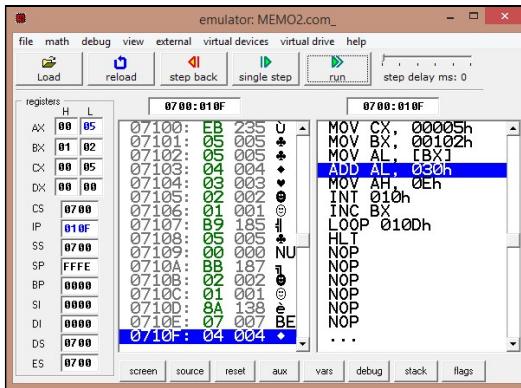


Figura 11.22 - Janela MEMO2 na ferramenta emu8086 (<Shift> + <F8> - quarta vez).

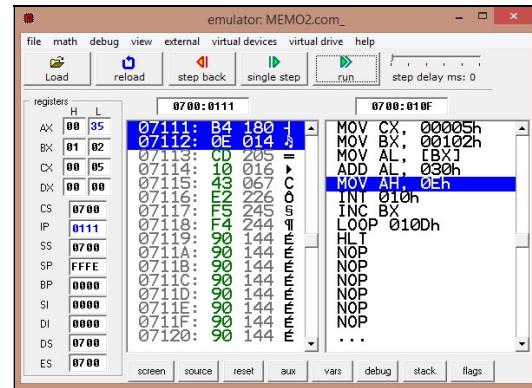


Figura 11.23 - Janela MEMO2 na ferramenta emu8086 (<Shift> + <F8> - quinta vez).

Assim que o valor **5d** estiver armazenado na memória, ele precisa ser apresentado. Assim sendo, execute as teclas de atalho <Shift> + <F8> para proceder à sexta execução passo a passo. Nessa etapa o valor **0Eh** será armazenado no

registraror **AH**. Esse valor corresponde ao código de escrita de um caractere quando do uso da instrução **INT 10h**. A Figura 11.24 mostra essa etapa da execução.

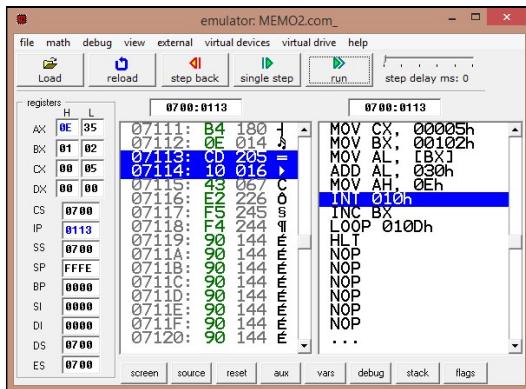


Figura 11.24 - Janela MEMO2 na ferramenta emu8086 (<Shift> + <F8> - sexta vez).

Perceba a inserção do valor **0Eh** no registrador menos significativo **AH** por intermédio da linha de instrução **MOV AH, 0Eh**. Na sequência execute pela sétima vez as teclas de atalho **<Shift> + <F8>**.

Será executada a instrução **INT 10h** que por meio do código **0Eh** armazenado no registrador **AH** fará a apresentação do conteúdo do registrador **AL** na tela do monitor de vídeo na sua forma decimal. A Figura 11.25 apresenta o resultado da ação.

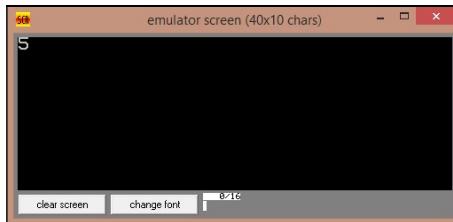


Figura 11.25 - Apresentação do resultado do programa.

Após a apresentação do valor 5 feche a janela **emulator screen** e observe que será executada a instrução **INC BX**. Acione o conjunto de teclas de atalho **<Shift> + <F8>** pela sétima vez e observe a alteração do valor do registrador **BX** para **0103h**, como na Figura 11.26. O valor **0002h** de **BX** passa a ser **0003h**.

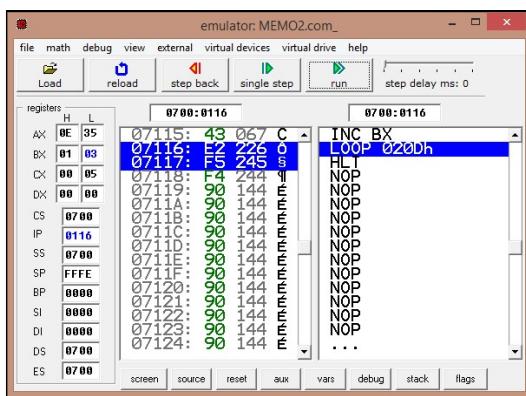


Figura 11.26 - Janela MEMO2 na ferramenta emu8086 (<Shift> + <F8> - sétima vez).

Ao ser executada a instrução **INC BX**, definiu-se o novo valor de endereço de memória a ser usado pela instrução **MOV AL, [BX]** assim que o laço ocorrer. Isso é repetido até que o registrador geral **CX** tenha o valor **0**. Vá repetindo as ações do programa para ver esse efeito ocorrer.

### 11.2.3 - Acesso indexado

O modo de acesso indexado (também denominado endereçamento de base ou endereçamento indexado) permite obter-se o endereço de uma posição de memória pelo deslocamento sequencial de um índice definido a partir de um dos registradores [BX], [BP], [DI] e [SI].

É interessante lembrar que os registradores [BX], [DI] e [SI] usam como registrador de segmento o **DS** e o [BP] utiliza como registrador de segmento o **SS**.

Por exemplo, a instrução **MOV AL, valor[BX]** transfere para o registrador menos significativo **AL** uma cópia do valor do byte residente no endereço de deslocamento identificado pelo registrador **[BX]** do segmento **DS** denominado **valor**. Desta forma, o registrador menos significativo **AL** passa a ter o valor armazenado no endereço de memória identificado por **[BX]**.

É possível ter acesso ao conteúdo de memória armazenado a partir do endereço de deslocamento indireto **[BX]**, como mostra a Figura 11.27.

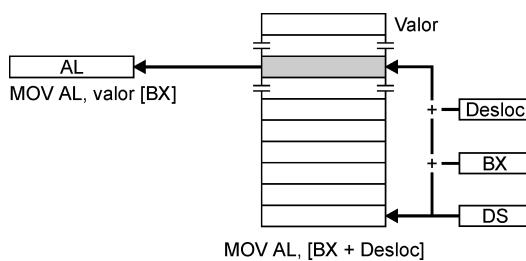


Figura 11.27 - Endereçamento indireto com registro (Adaptado de HYDE, 2003).

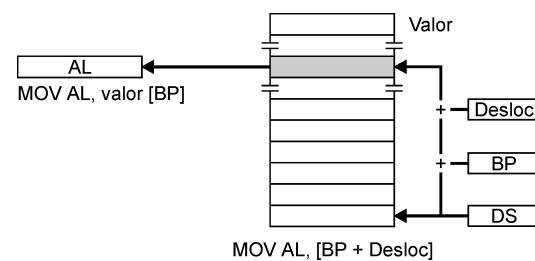


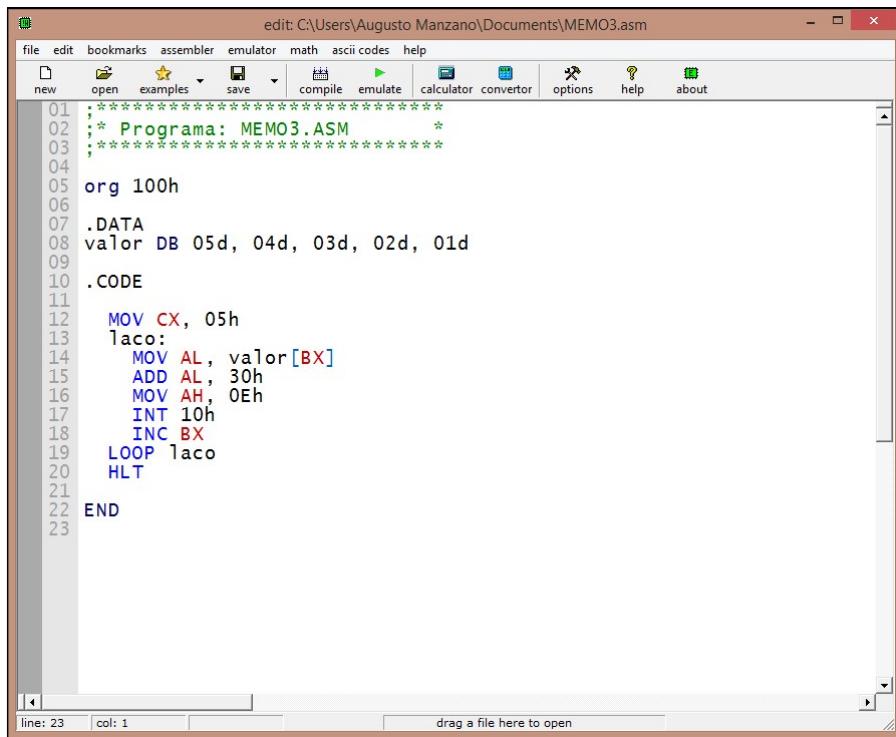
Figura 11.28 - Endereçamento indireto com registro (Adaptado de HYDE, 2003).

A Figura 11.27 indica que por meio da instrução **MOV AL, valor[BX]** (ou instrução **MOV AL, [BX + DESLOC]**, em que **DESLOC** pode ser um valor de 8 ou 16 bits) o registrador menos significativo **AL** é armazenado com os valores existentes da área de dados identificada pela variável **valor**, apontada pelo registrador de segmento **DS** até a posição limite (a ser definido no programa) a ser obtida com o uso do registrador **[BX]**. Para fazer isso, é necessário definir um laço de repetição. A informação **[BX]** da Figura 11.27 pode ser entendida como **[DI]** ou **[SI]**.

```
;*****  
;* Programa: MEMO3.ASM *  
;*****  
  
.org 100h  
  
.DATA  
valor DB 05d, 04d, 03d, 02d, 01d  
  
.CODE  
  
MOV CX, 05h  
laco:  
    MOV AL, valor[BX]  
    ADD AL, 30h  
    MOV AH, 0Eh  
    INT 10h  
    INC BX  
LOOP laco  
HLT  
  
END
```

Para usar o registrador de pilha **[BP]**, deve ser considerado como registrador de segmento o **SS**. A Figura 11.28 mostra a estrutura de funcionamento desse tipo de operação. Para exemplificar o acesso indexado de memória, considere o seguinte programa:

Execute no programa **emu8086** o comando de menu **file/new/com template**, ação as teclas de atalho **<Ctrl> + <A>** do editor de texto e escreva o programa anterior, gravando-o por meio dos comandos de menu **file/save** com o nome **MEMO3**, de forma que fique semelhante à imagem da Figura 11.29.



```

edit: C:\Users\Augusto Manzano\Documents\MEMO3.asm
file edit bookmarks assembler emulator math ascii codes help
new open examples save compile emulate calculator converter options help about
01 ; **** Programa: MEMO3.ASM ****
02 ;
03 ;
04
05 org 100h
06
07 .DATA
08 valor DB 05d, 04d, 03d, 02d, 01d
09
10 .CODE
11
12 MOV CX, 05h
13 laço:
14     MOV AL, valor[BX]
15     ADD AL, 30h
16     MOV AH, 0Eh
17     INT 10h
18     INC BX
19     LOOP laço
20     HLT
21
22 END
23

```

Figura 11.29 - Programa MEMO3 na ferramenta emu8086.

Observe então o trecho de código definido entre o rótulo **laço**: e a instrução **LOOP laço**. Note atentamente a utilização das instruções **MOV AL, valor[BX]** e **INC BX**. A instrução **MOV AL, valor[BX]** movimenta um dos conteúdos identificados na área de dados pela variável **valor**. À medida que se obtém memória e sua respectiva apresentação, o programa acrescenta 1 ao valor existente no registrador **[BX]**. Desta forma, é possível obter todos os valores definidos na variável **valor**. O acesso indireto à memória por registrador é útil quando se necessita trabalhar com matrizes.

Em seguida peça a execução do programa pela tecla de função **<F5>** e ação também uma vez a tecla de função **<F8>**. Observe os detalhes apresentados na janela **MEMO3.com**, como indica a Figura 11.30.

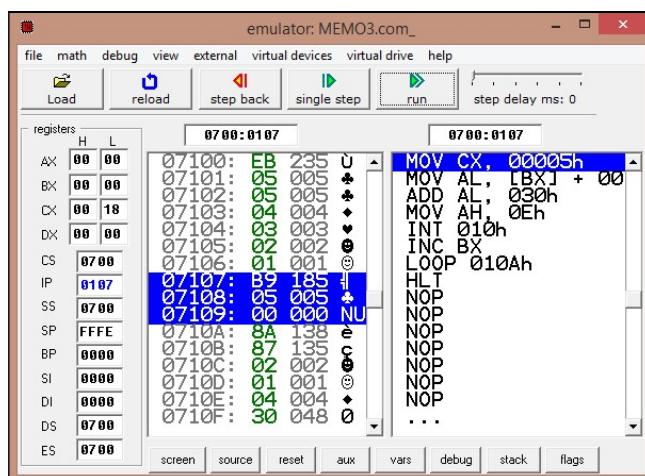


Figura 11.30 - Janela MEMO3.com na ferramenta emu8086.

Observe atentamente as informações apresentadas nas áreas **disassembler** e **memory**. A primeira instrução do programa **MOV CX, 00005h** está posicionada no endereço de memória **0700:0107**. Vale a pena fazer um paralelo entre o código escrito no editor e o código assumido pelo simulador. Atente para a Tabela 11.3.

Tabela 11.3 - Código no editor e no simulador

| Código no Editor  | Código no Simulador   |
|---|---|
| <pre>MOV CX, 05h laco:     MOV AL, valor[BX]     ADD AL, 030h     MOV AH, 0Eh     INT 010h     INC BX     LOOP laco     HLT</pre> | <pre>MOV CX, 00005h MOV AL, [BX] + 00102h ADD AL, 030h ADD AH, 0Eh INT 010h INC BX LOOP 010Ah HLT</pre> |

Considere a linha de código **MOV AL, valor[BX]** (código no editor) com a linha de código **MOV AL, [BX] + 00102h** (código no simulador).

O primeiro valor (**05d**) definido na área de dados representada pela variável **valor** está posicionado no endereço de memória **0102h** do segmento **0700h**, como pode ser observado na área **memory** da janela **emulator: MEMO2.com** na Figura 11.30.

O uso do registrador **[BX]** na linha de código **MOV AL, valor[BX]** (que internamente é interpretada como **MOV AL, [BX] + 00102h**) possibilita obter o primeiro valor da área de dados (**valor**) que está armazenado na posição **0** (zero). Por este motivo é que antes da instrução **LOOP** se encontra a instrução **INC BX**, que acrescenta **1** (um) ao valor atual do registrador **[BX]**. A Tabela 11.4 demonstra a ocorrência interna dos valores do registrador **[BX]**.

Tabela 11.4 – Ocorrência interna dos valores do registrador **[BX]**

| Registrador <b>[BX]</b> | Instrução <b>MOV AL, [BX] + 00102h</b> | Conteúdo de Memória |
|-------------------------|--|---------------------|
| 0                       | $0 + 00102h = 00102h$                  | 5                   |
| 1                       | $1 + 00102h = 00103h$                  | 4                   |
| 2                       | $2 + 00102h = 00104h$                  | 3                   |
| 3                       | $3 + 00102h = 00105h$                  | 2                   |
| 4                       | $4 + 00102h = 00106h$                  | 1                   |

Utilize em conjunto as teclas **<Shift> + <F8>** para executar o programa. Observe a cada movimentação as alterações dos valores nos registradores e a apresentação dos valores na tela do monitor de vídeo.

O programa **MEMO3** pode também ser escrito de uma maneira um pouco diferente da forma apresentada. A Tabela 11.5 exibe as versões **MEMO4** e **MEMO5** com outras formas de referência de uso do acesso indireto com registrador.

Tabela 11.5 – Comparativo de acesso indireto com registrador

| MEMO4.asm  | MEMO5.asm  |
|--|--|
| <pre>.CODE  MOV CX, 05h laco:     MOV AL, [BX + valor]     ADD AL, 30h     MOV AH, 0Eh     INT 10h     INC BX     LOOP laco     HLT  END</pre> | <pre>.CODE  MOV CX, 05h laco:     MOV AL, [BX] + valor     ADD AL, 30h     MOV AH, 0Eh     INT 10h     INC BX     LOOP laco     HLT  END</pre> |

Internamente os programas deste tópico serão interpretados e executados da mesma forma.

## 11.2.4 - Acesso de base indexada

O acesso de endereçamento de base indexada é uma variação do endereçamento indireto com registradores. Ele utiliza os registradores de base **BX** e **BP** vinculados aos registradores de índice **SI** e **DI**. As Figuras 11.31 e 11.32 apresentam as formas permitidas de uso.

O acesso a uma determinada posição de memória por meio de endereçamento de base indexada é conseguido com a soma de um registrador de base **BX** ou **BP** com um registrador de índice **SI** ou **DI**. Por exemplo, se o registrador **BX** estiver com o valor **0102h** e o registrador **SI** estiver com o valor **03h**, a instrução **MOV AL, [BX + SI]** carrega o registrador menos significativo **AL** com o valor **0105h**.

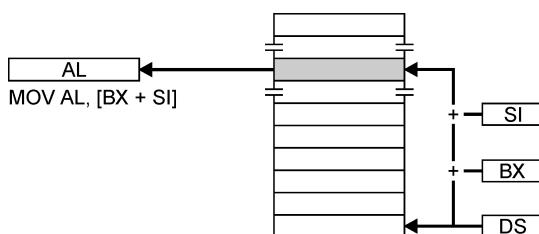


Figura 11.31 - Endereçamento de base indexada (Adaptado de Hyde, 2003).

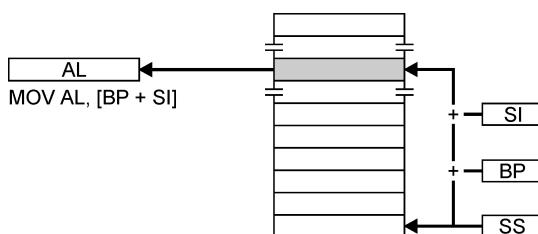


Figura 11.32 - Endereçamento de base indexada (Adaptado de Hyde, 2003).

Para exemplificar o acesso de base indexada de memória considere o seguinte programa:

```
;*****
;*      Programa: MEMO6.ASM      *
;*****
```

```
org 100h
.DATA
  valor DB 05d, 04d, 03d, 02d, 01d
.CODE

  MOV CX, 05h
  LEA BX, valor
  laco:
    MOV AL, [BX + SI]
    ADD AL, 30h
    MOV AH, 0Eh
    INT 10h
    INC SI
  LOOP laco
  HLT
```

**END**

Execute no programa **emu8086** o comando de menu **file/new/com template**, aione as teclas de atalho **<Ctrl> + <A>** do editor de texto e escreva o programa anterior, gravando-o por meio dos comandos de menu **file/save** com o nome **MEMO6**, de forma que fique semelhante à imagem da Figura 11.33.

Peça a execução do programa pela tecla de função **<F5>** e aione também uma vez a tecla de função **<F8>**. A partir desse ponto vá aionando o conjunto de teclas **<Shift> + <F8>** e observe a cada ciclo do laço as mudanças ocorridas nos registradores **BX** e **SI**.

```

edit: C:\Users\Augusto Manzano\Documents\MEMO6.asm
file edit bookmarks assembler emulator math ascii codes help
new open example save compile emulate calculator converter options help about
01 ;***** Programa: MEMO6.ASM *****
02 ;
03 ;
04 ;
05 org 100h
06
07 .DATA
08     valor DB 05d, 04d, 03d, 02d, 01d
09
10 .CODE
11
12     MOV CX, 05h
13     LEA BX, valor
14     laco:
15         MOV AL, [BX + SI]
16         ADD AL, 30h
17         MOV AH, 0Eh
18         INT 10h
19         INC SI
20     LOOP laco
21     HLT
22
23 END
24

```

Figura 11.33 - Programa MEMO6 na ferramenta emu8086.

O uso dos registradores **[BX + SI]** na linha de código **MOV AL, [BX + SI]** possibilita a obtenção dos valores armazenados da área de dados (**valor**). O primeiro valor está armazenado na posição (representado pelo registrador **SI**) de memória **0** (zero), por este motivo é que antes da instrução **LOOP** se encontra a instrução **INC SI**, que acrescenta **1** (um) ao valor atual do registrador **[SI]**. A Tabela 11.6 demonstra a ocorrência interna dos valores dos registradores **[BX + SI]**.

Tabela 11.6 - Ocorrência interna dos valores dos registradores **[BX + SI]**

| Registrador <b>[SI]</b> | Instrução <b>MOV AL, [BX + SI]</b> | Conteúdo de Memória |
|-------------------------|------------------------------------|---------------------|
| 0                       | $00102h + 0 = 00102h$              | 5                   |
| 1                       | $00102h + 1 = 00103h$              | 4                   |
| 2                       | $00102h + 2 = 00104h$              | 3                   |
| 3                       | $00102h + 3 = 00105h$              | 2                   |
| 4                       | $00102h + 4 = 00106h$              | 1                   |

## 11.2.5 - Acesso de base indexada mais deslocamento

O acesso de endereçamento de base indexada mais deslocamento é uma variação do endereçamento inde-xado com endereçamento de base indexada. O acesso de base indexada mais deslocamento utiliza os registradores de base **BX** e **BP** vinculados aos registradores de índice **SI** e **DI** e ao valor de deslocamento (constante de 8 ou 16 bits). As Figuras 11.34 e 11.35 apresentam as formas permitidas de uso.

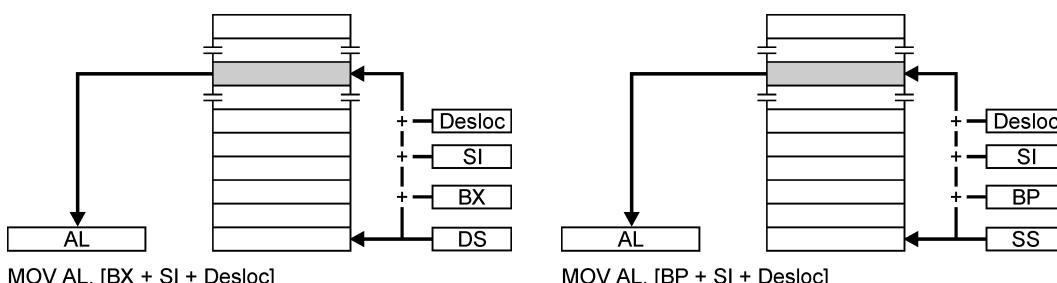


Figura 11.34 - Base indexada com deslocamento (Adaptado de Hyde, 2003).

Figura 11.35 - Base indexada com deslocamento (Adaptado de Hyde, 2003).

O acesso a uma determinada posição de memória pelo endereçamento de base indexada com deslocamento é conseguido com a soma de um registrador de base **BX** ou **BP** com um registrador de índice **SI** ou **DI** mais a constante de **DESLOC**. Por exemplo, se o registrador **BX** estiver com o valor **0102h** e o registrador **SI** estiver com o valor **03h**, e a constante for um valor como **02h**, a instrução **MOV AL, [BX + SI + 02h]** carrega o registrador menos significativo **AL** com o valor **0107h**.

Para exemplificar a ação de acesso de base indexada com deslocamento de memória, considere o seguinte programa:

```
;*****
;*      Programa: MEMO7.ASM      *
;*****

.org 100h

.DATA
    valor DB 05d, 04d, 03d, 02d, 01d

.CODE
    MOV CX, 05h
    LEA BX, valor - 02h
laco:
    MOV AL, [BX + SI + 02h]
    ADD AL, 30h
    MOV AH, 0Eh
    INT 10h
    INC SI
    LOOP laco
    HLT

END
```

Execute no programa **emu8086** o comando de menu **file/new/com template**, aione as teclas de atalho **<Ctrl> + <A>** do editor de texto e escreva o programa anterior, gravando-o por meio dos comandos de menu **file/save** com o nome **MEMO7**, de forma que fique semelhante à imagem da Figura 11.36.

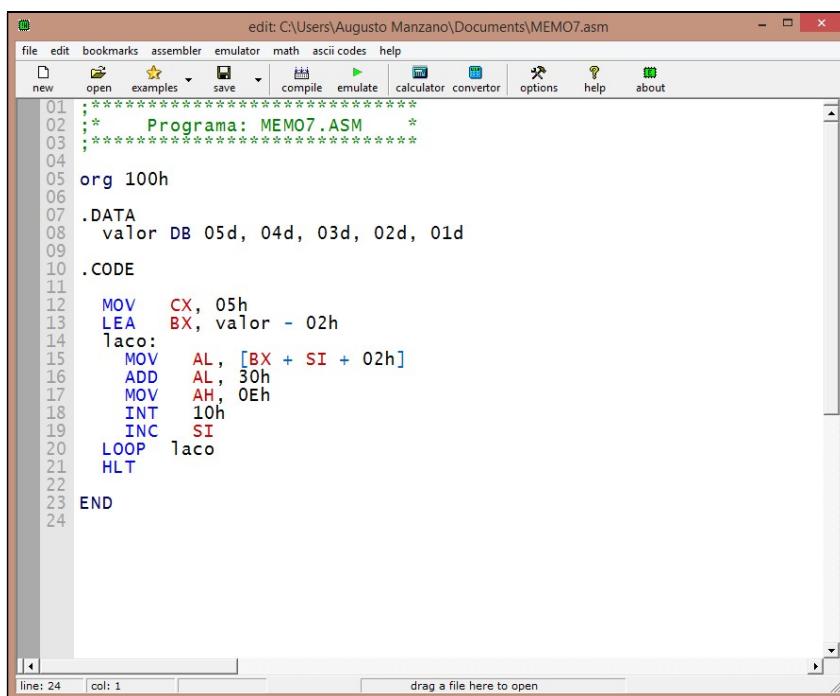


Figura 11.36 - Programa MEMO7 na ferramenta emu8086.

Execute o programa com a tecla de função <F5> e acione também uma vez a tecla de função <F8>. A partir desse ponto vá acionando o conjunto de teclas <Shift> + <F8> e observe a cada ciclo do laço as mudanças ocorridas nos registradores **BX** e **SI**. Atente para o fato da definição do valor constante de deslocamento **02h**.

Para exemplificar o uso da constante, foi alterado na linha de código **LEA BX, valor – 02h** o endereço de início da área de dados **valor** em duas posições a menos na memória. Torna-se possível exemplificar de forma didática o funcionamento do acesso ao endereçamento de base indexada mais deslocamento.

O uso de **[BX + SI + 02h]** na linha de código **MOV AL, [BX + SI + 02h]** possibilita obter os valores armazenados da área de dados (**valor**). A Tabela 11.7 demonstra a ocorrência interna dos valores dos registradores **[BX + SI + 02h]**.

**Tabela 11.7 - Ocorrência interna dos valores dos registradores [BX + SI + 02h]**

| Registrador [SI] | Instrução <b>MOV AL, [BX + SI + 02h]</b> | Conteúdo de Memória |
|------------------|--|---------------------|
| 0                | <b>00100h + 0 + 02h = 00102h</b>         | 5                   |
| 1                | <b>00100h + 1 + 02h = 00103h</b>         | 4                   |
| 2                | <b>00100h + 2 + 02h = 00104h</b>         | 3                   |
| 3                | <b>00100h + 3 + 02h = 00105h</b>         | 2                   |
| 4                | <b>00100h + 4 + 02h = 00106h</b>         | 1                   |

## 11.3 - Prefixos de anulação

---

O padrão de funcionamento de deslocamentos dentro de um determinado segmento é sempre feito dentro do próprio segmento, ou seja, o deslocamento ocorre no segmento em uso. No entanto, é possível utilizar deslocamentos em segmentos diferentes daquele em uso. É necessário utilizar um conceito denominado *override segment* (segmento anulado ou prefixo de anulação).

Por exemplo, imagine que se queira acessar o endereço de deslocamento **1234h** do segmento de pilha **ES**. Neste caso, deveria ser utilizada a instrução **MOV AX, ES:[1234h]**. De forma semelhante, para acessar esse deslocamento no segmento de código deveria ser utilizada a instrução **MOV AX, CS:[1234h]**. A indicação **ES:** é um prefixo de anulação. Segundo Hyde (2003), o prefixo **:DS** não é de anulação (*override segment*).

O prefixo de anulação pode ser usado com as formas de acesso de memória indireto com registrador e indexado, sendo válidas as seguintes definições:

- ◆ Indireto com registrador:    **MOV AL, CS:[BX]**  
**MOV AL, DS:[BP]**  
**MOV AL, SS:[SI]**  
**MOV AL, ES:[DI]**
  
- ◆ Indexado:                    **MOV AL, SS:DESLOC[BX]**  
**MOV AL, ES:DESLOC[BP]**  
**MOV AL, CS:DESLOC[SI]**  
**MOV AL, SS:DESLOC[DI]**

O tema deste tópico é um pouco mais complexo, por isso foge aos objetivos iniciais desta obra. Ele é citado apenas a título de ilustração, portanto não será exemplificado.

No entanto, para os mais curiosos, no arquivo de biblioteca **emu8086.inc** que acompanha a ferramenta **emu8086** existem algumas rotinas que utilizam prefixos, além do programa exemplo **ToBin.asm**. Esse material deve ser utilizado como fonte de consulta adicional a esta obra.

## 11.4 - Operações com Matrizes

---

As matrizes estão relacionadas ao conjunto de dados armazenado em memória num formato de lista (matriz de uma dimensão) ou tabela (matriz com mais de uma dimensão, normalmente denominada matriz de duas dimensões).

Em vários exemplos aplicados neste capítulo foi utilizada a matriz de uma dimensão de forma implícita. O uso de matriz de forma explícita é conseguido com a sintaxe:

```
vetor DW 5 DUP (0)
```

Neste caso é definida uma variável denominada **vetor** que tem a capacidade de manipular até cinco valores do tipo *word* inicializados com o valor **0** (zero). Se for necessário trabalhar com uma matriz com mais de uma dimensão (duas dimensões), deve ser utilizada a sintaxe:

```
matriz DW 5 DUP (3 DUP (0))
```

Neste caso é definida uma variável **matriz**, a qual é inicializada com 15 *words* com valor zero. Essa forma seria algo similar a uma tabela de cinco linhas e três colunas.

Para exemplificar e ilustrar a entrada e apresentação de cinco valores inteiros positivos em uma matriz, considere o seguinte código de programa:

```
#MAKE_COM#  
  
; Programas do tipo .COM são iniciados no endereço CS:0100h  
.org 100h  
  
;*****  
;*      Programa: MEMO8.ASM      *  
;*****  
  
.INCLUDE 'emu8086.inc'  
  
.DATA  
tamanho EQU 05h  
vetor DB tamanho DUP ('X')  
msg1a DB 'Entre o ', 0h  
msg1b DB 'o. valor numerico: ', 0h  
msg2a DB '0 ', 0h  
msg2b DB 'o. valor informado equivale a: ', 0h  
valor DB ?  
  
.CODE  
PRINTN 'Programa Matriz'  
PRINTN 'Entre valores na faixa de 0 ate 255'  
PUTC 13d  
PUTC 10d  
  
MOV CX, tamanho  
MOV valor, 01h  
entrada:  
    PUSH CX  
    LEA SI, msg1a  
    CALL PRINT_STRING  
    MOV AL, valor  
    CALL PRINT_NUM  
    INC valor  
    LEA SI, msg1b  
    CALL PRINT_STRING  
    CALL SCAN_NUM  
    MOV vetor[BX], CL  
    INC BX  
    POP CX  
    PUTC 13d  
    PUTC 10d  
LOOP entrada
```

```

PUTC 13d
PUTC 10d

MOV CX, tamanho
MOV valor, 01h
saida:
LEA SI, msg2a
CALL PRINT_STRING
MOV AL, valor
CALL PRINT_NUM
INC valor
LEA SI, msg2b
CALL PRINT_STRING
MOV AL, vetor[BX - tamanho]
CALL PRINT_NUM
INC BX
PUTC 13d
PUTC 10d
LOOP saida

HLT
DEFINE_PRINT_NUM
DEFINE_PRINT_NUM_UNS
DEFINE_PRINT_STRING
DEFINE_SCAN_NUM
END

```

Execute no programa **emu8086** o comando de menu **file/new/com template**, acione as teclas de atalho <Ctrl> + <A> do editor de texto e escreva o programa anterior, gravando-o por meio dos comandos de menu **file/save** com o nome **MEMO8**, de forma que fique semelhante à imagem das Figuras 11.37, 11.38 e 11.39.

```

edit: C:\Users\Augusto Manzano\Documents\MEMO8.asm
file edit bookmarks assembler emulator math ascii codes help
new open examples save compile emulate calculator converter options help about
01 ;***** Programa: MEMO8.ASM *****
02 ;*****
03 ;
04
05 INCLUDE 'emu8086.inc'
06
07 .DATA
08     tamanho EQU 05h
09     vetor DB tamanho DUP ('X')
10    msg1a DB 'Entre o ', 0h
11    msg1b DB 'o. valor numerico: ', 0h
12    msg2a DB 'O ', 0h
13    msg2b DB 'o. valor informado equivale a: ', 0h
14    valor DB ?
15
16 .CODE
17    PRINTN 'Programa Matriz'
18    PRINTN 'Entre valores na faixa de 0 ate 255'
19    PUTC 13d
20    PUTC 10d
21
22    MOV CX, tamanho
23    MOV valor, 01h
24    entrada:
25        PUSH CX
26        LEA SI, msg1a
27        CALL PRINT_STRING
28        MOV AL, valor
29        CALL PRINT_NUM
30        INC valor
31        LEA SI, msg1b

```

Figura 11.37 - Programa MEMO8 com a área de dados.

The screenshot shows the MEMO8 assembly editor interface with the file C:\Users\Augusto Manzano\Documents\MEMO8.asm open. The code window displays the following assembly code:

```
16 .CODE
17 PRINTN 'Programa Matriz'
18 PRINTN 'Entre valores na faixa de 0 ate 255'
19 PUTC 13d
20 PUTC 10d
21
22 MOV CX, tamanho
23 MOV valor, 01h
24 entrada:
25 PUSH CX
26 LEA SI, msg1a
27 CALL PRINT_STRING
28 MOV AL, valor
29 CALL PRINT_NUM
30 INC valor
31 LEA SI, msg1b
32 CALL PRINT_STRING
33 CALL SCAN_NUM
34 MOV vetor[BX], CL
35 INC BX
36 POP CX
37 PUTC 13d
38 PUTC 10d
39 LOOP entrada
40
41 PUTC 13d
42 PUTC 10d
43
44 MOV CX, tamanho
45 MOV valor, 01h
46 saida:
```

Figura 11.38 - Programa MEMO8 com rotina de entrada de dados.

The screenshot shows the MEMO8 assembly editor interface with the file C:\Users\Augusto Manzano\Documents\MEMO8.asm open. The code window displays the same assembly code as Figure 11.38, but with the output data routine removed (lines 41 to 46).

```
16 .CODE
17 PRINTN 'Programa Matriz'
18 PRINTN 'Entre valores na faixa de 0 ate 255'
19 PUTC 13d
20 PUTC 10d
21
22 MOV CX, tamanho
23 MOV valor, 01h
24 entrada:
25 PUSH CX
26 LEA SI, msg1a
27 CALL PRINT_STRING
28 MOV AL, valor
29 CALL PRINT_NUM
30 INC valor
31 LEA SI, msg1b
32 CALL PRINT_STRING
33 CALL SCAN_NUM
34 MOV vetor[BX], CL
35 INC BX
36 POP CX
37 PUTC 13d
38 PUTC 10d
39 LOOP entrada
40
41 PUTC 13d
42 PUTC 10d
43
44 MOV CX, tamanho
45 MOV valor, 01h
46 saida:
```

Figura 11.39 - Programa MEMO8 com rotina de saída de dados.

A Figura 11.37 apresenta a área de dados do programa. Observe na linha 8 a definição da constante **tamanho** com o valor **05h** (valor máximo de elementos a serem inseridos na variável **vetor**). Desejando trabalhar com um número maior ou menor de valores, basta ajustar o valor **05h** para o desejado.

A linha 9 mostra uma matriz de uma dimensão denominada **vetor** e inicializada na memória com cinco posições contendo com o caractere **X**. Da linha 10 até a linha 13 são definidas as variáveis com as mensagens de orientação do programa.

A linha **14** apresenta a variável **valor** inicializada com um valor desconhecido, representado pelo caractere **?**.

Da linha **22** até a linha **39** encontra-se o trecho de programa responsável pela entrada dos valores e seu armazenamento na memória.

A linha **22** mostra o registrador geral **CX** com o valor armazenado na constante **tamanho**. Isso é necessário, uma vez que a instrução **LOOP** usa o registrador geral **CX** para controlar o número de vezes do laço de repetição em si.

Na linha **23** é exibida a variável **valor** com o valor **01h**, a qual será usada para armazenar o valor do indicador de número de ações do programa.

Da linha **24** até a linha **39** é executada uma série de ações. A primeira delas é armazenar na pilha (linha **25**) o valor do registrador geral **CX**. Isso é necessário devido ao fato de **SCAN\_NUM** (biblioteca **emu8086.inc**) utilizar o registrador geral **CX** para a ação da entrada de um valor numérico. Atente para as linhas **33** e **34**, em que está definida a manipulação de transferência do valor armazenado no registrador menos significativo **CL** para o **vetor[BX]**.

O trecho correspondente à saída de dados, Figura 11.33, situa-se entre as linhas **44** e **59**. Observe a linha da instrução da linha **62** correspondente a **MOV AL, vetor[BX - tamanho]**. Está sendo feito um balanceamento do deslocamento a partir de **vetor[BX - tamanho]**. Isso é necessário porque a rotina de entrada movimenta o deslocamento de **[BX]**.

Em relação à utilização de uma matriz de duas dimensões, pode-se utilizar como exemplo o programa **matrix.asm** encontrado na pasta **Samples** que acompanha o programa **emu8086**, o qual inverte os valores existentes entre as linhas e colunas de uma determinada matriz que possui como código:

```
; matrix transpose sample (reverse rows with columns).

name "matrix"

org 100h

jmp start ; go to code...

msg db "to the view matrix click vars button,", 0dh,0ah
      db " and set elements property to 3 for these items:", 0dh,0ah, 0ah
      db "    matrix    ", 0dh,0ah
      db "    row1     ", 0dh,0ah
      db "    row2     ", 0dh,0ah, 0dh,0ah
      db "or add print-out support to this program...", 0dh,0ah, '$'

matrix_size equ 3

; ----- matrix -----
matrix    db 1,2,3
row1     db 4,5,6
row2     db 7,8,9
;-----

i dw ?
j dw ?

start:
mov i, 0
next_i:

        ; j = i + 1
        mov cx, i
        inc cx
        mov j, cx
        next_j:

        mov si, i
        mov bx, j
```

```

    mov al, matrix_size
    mov cx, si
    mul cl
    mov si, ax
    mov dl, matrix[si][bx]

    mov si, i
    mov al, matrix_size
    mul bl
    mov bx, ax
    xchg matrix[bx][si], dl

    mov bx, j
    mov al, matrix_size
    mov cx, si
    mul cl
    mov si, ax
    mov matrix[si][bx], dl

    inc j
    cmp j, matrix_size
    jb next_j

inc i
cmp i, matrix_size/2
jbe next_i

; print message....
lea dx, msg
mov ah, 9
int 21h
; wait for any key press...
mov ah, 0
int 16h
ret

```

A Figura 11.40 exemplifica a ação do programa **matrix.asm**. que mostra a primeira linha da matriz torna-se a sua primeira coluna, a segunda linha torna-se a segunda coluna e a terceira linha torna-se a terceira coluna. A seguir é apresentado na íntegra o código do programa **matrix.asm**.

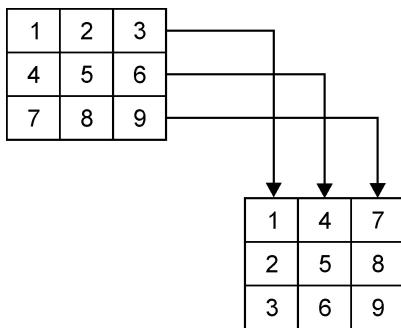


Figura 11.40 - Exemplo de funcionalidade do programa MATRIX.

Para executar e visualizar esse programa, é necessário utilizar alguns outros recursos da ferramenta **emu8086** ainda não apresentados. Primeiramente acione a tecla de função <**F5**> para iniciar o processo de execução do programa.

Depois, conforme orientação no próprio código do programa, acione o comando de menu **view/variables** na janela de depuração para que a caixa de diálogo **variables** seja apresentada como na Figura 11.41.

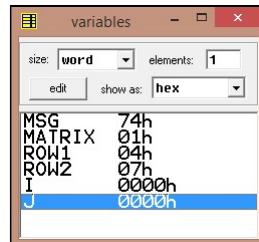


Figura 11.41 - Caixa de diálogo Variables.

A caixa de diálogo **variables** apresenta todas as variáveis definidas no programa. No entanto, não estão sendo apresentados todos os valores das variáveis **MATRIX**, **ROW1** e **ROW2**, apenas mostra o primeiro valor de cada uma delas. Para poder visualizar os três valores de cada variável, selecione com o ponteiro do mouse a variável **MATRIX** na lista apresentada. No campo **elements** informe sobre o valor 1 o valor 3. Repita a mesma ação para as variáveis **ROW1** e **ROW2**, de forma que se tenha um visual semelhante à Figura 11.42.

A Figura 11.42 mostra a mesma sequência de valores definida no código de programa. Coloque o programa em execução e observe ao final as informações da janela **variables**, como indica a Figura 11.43 contendo a inversão dos valores da matriz.

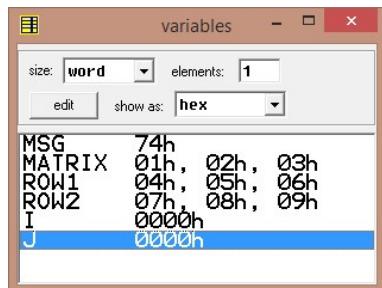


Figura 11.42 - Caixa de diálogo variables com a apresentação dos valores.

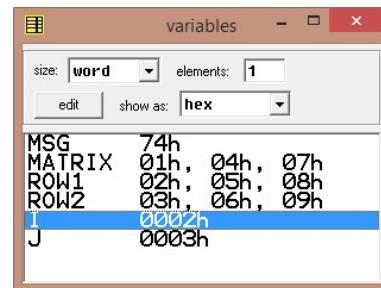


Figura 11.43 - Caixa de diálogo variables com a inversão da matriz.

Para finalizar a apresentação preliminar de introdução à linguagem de programação Assembly e no sentido de mostrar mais alguns recursos do programa **emu8086**, carregue para o ambiente de programação o programa **DIVIDE1**, acione a tecla de função **<F5>** e quando for apresentada a tela **emulator: DIVIDE1.exe\_**, selecione na parte inferior da tela o botão **aux**. Nas opções apresentadas selecione **listing** e será exibido dentro do programa **Bloco de notas** o código do programa escrito, sendo do lado direito indicado o programa em linguagem Assembly e do lado esquerdo é indicado o programa escrito em código de máquina como indica a Figura 11.41.

```

DIVIDE1.exe_list - Bloco de notas
Arquivo Editar Formatar Exibir Ajuda
EMU8086 GENERATED LISTING. MACHINE CODE <- SOURCE.

DIVIDE1.exe_ -- emu8086 assembler version: 4.08

[ 29/10/2018 -- 13:50:19 ]

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

[ 1]   :
[ 2]   :
[ 3]   :
[ 4]   :
[ 5]   :
[ 6]   :
[ 7]   :
[ 8]   :
[ 9] 0200: 09 00 ; ****
;*      Programa: DIVIDE1.ASM    *
;****

. MODEL small
. STACK 512d

. DATA
a DW 9d
b DB 2d
x DB 0, '$'

[ 10] 0202: 02
[ 11] 0203: 00 24
[ 12]   :

```

Figura 11.41 - Caixa de diálogo Variables.

No relatório apresentado junto ao programa **Bloco de notas** é indicado além do código em linguagem de máquina e linguagem de montagem a apresentação do cabeçalho de um programa no formato **.EXE**.

## 11.5 - Programas Exemplo

Além da biblioteca **emu8086.inc**, na pasta **samples** há uma série de programas exemplo que podem ser utilizados como mecanismos de aprendizado e aprofundamento de alguns detalhes operacionais da linguagem de programação de computadores Assembly 8086/8088.

Para ter acesso aos exemplos que acompanham a ferramenta, pelo menu principal do programa **emu8086** execute o comando de menu **file/examples** e será apresentada uma lista de programas exemplos como indica a Figura 11.42 ou por meio da pequena seta ao lado direito do botão **examples** como mostra a Figura 11.43.

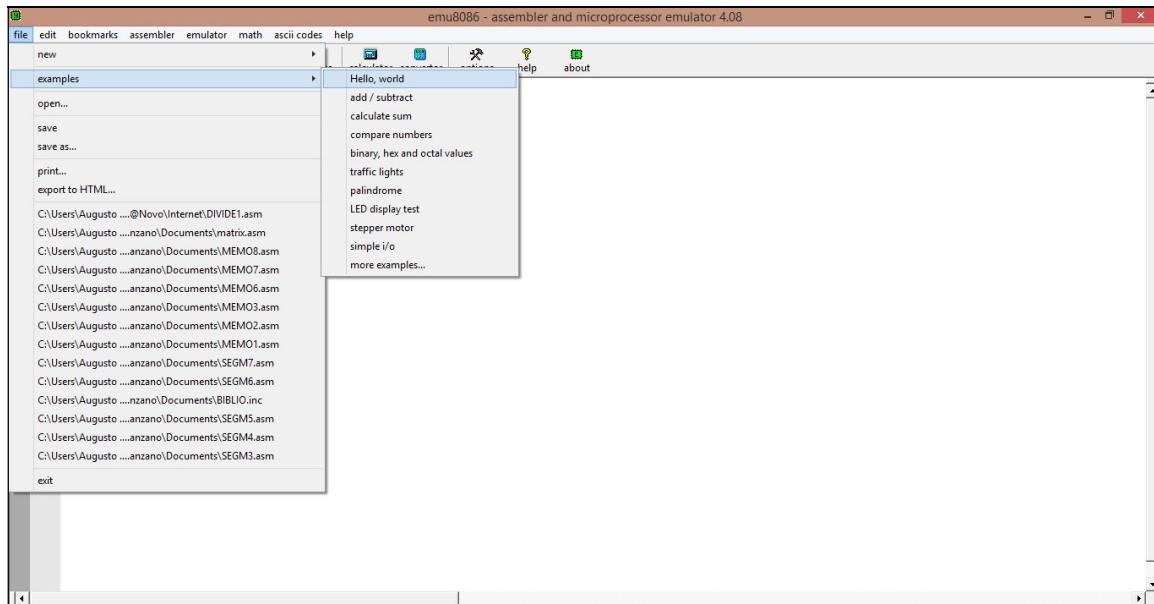


Figura 11.42 - Opções de Exemplos (File/Examples).

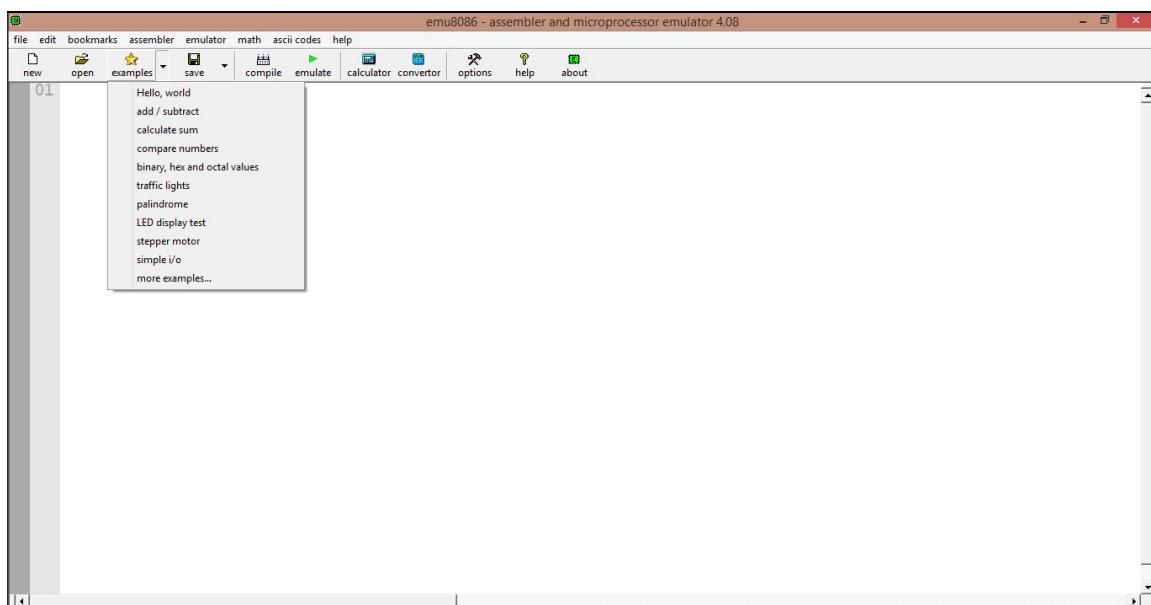


Figura 11.43 - Opções de Exemplos (Botão na barra de ferramentas).

As Figuras 11.42 e 11.43 apresentam um menu com alguns exemplos de programa. Atente para a opção **more samples...** (que indica a existência de mais programas a serem carregados e estudados). Se este botão for acionado, é apresentada a caixa de diálogo **Abrir**, Figura 11.44, que também pode ser aberta com o comando de menu **file/open**, com seleção da pasta **samples**.

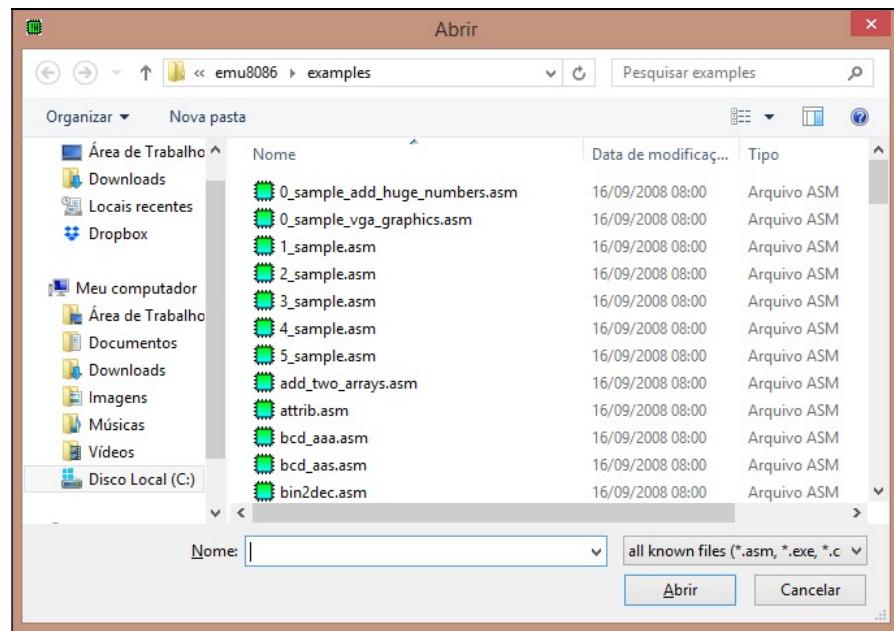


Figura 11.44 - Caixa de diálogo "Abrir" com mais exemplos.

Abra os programas, execute-os e estude os seus códigos. A ferramenta de simulação **emu8086** é voltada para o estudo e a aprendizagem. Por esta razão, além de seu foco didático em mostrar detalhes internos de um microprocessador padrão 8086, ela traz uma extensa coleção de programas para auxiliar por meio de exemplos a aprendizagem. Daqui para frente é só se divertir e aprofundar o conhecimento iniciado neste livro!

## Anotações