

### 3. Introdução à linguagem montadora do 8086

#### 3.1 A sintaxe assembly do 8086

A linguagem montadora não é sensível à letra maiúscula ou minúscula

Para facilitar a compreensão do texto do programa, **sugere-se:**

- uso de letra maiúscula para código
- uso de letra minúscula para comentários

#### Declarações (*statements*):

- **instruções**, que são convertidas em código de máquina
- **diretivas**, que instruem o montador a realizar alguma tarefa específica:
  - alocar espaço de memória para variáveis
  - criar uma sub-rotina (***procedure*** ou procedimento)

Formato de uma declaração (linha de programa):

**[Nome] [Cod. oper.] [Operando(s)] [;Comentário]**

Exemplo:

INICIO:           MOV   CX,5h           ;inicializar contador

A separação entre os campos deve ser do tipo **<espaço>** ou **<tab>**.

- **O campo Nome:**

Pode ser um **rótulo** de instrução, um **nome de sub-rotina**, um **nome de variável**, contendo de 1 a 31 caracteres, iniciando por uma letra e contendo somente letras, números e os caracteres **? . @ \_ : \$ % .**

Obs: o Montador traduz os **nomes** por endereços de memória.

Exemplos:	<b>nomes válidos</b>	<b>nomes inválidos</b>
	LOOP1:	DOIS BITS
	.TEST	2abc
	@caracter	A42.25
	SOMA_TOTAL4	#33
	\$100	

- **Campo de código de operação:**

Contem o código de operação simbólico (**mnemônico**)

No caso de diretivas, contem o código de **pseudo-instrução**

Exemplos:	instruções	diretivas
	MOV	.MODEL
	ADD	.STACK
	INC	nome PROC
	JMP	

- **Campo de operandos:**

Instruções podem conter 0, 1 ou 2 operandos no 8086.

Exemplos:

NOP		;sem operandos: instrui para fazer nada
INC AX		;um operando: soma 1 ao conteúdo de AX
ADD	A,2d	;dois operandos: soma 2 ao conteúdo da palavra ;de memória A

No caso de **instruções** de dois operandos:

- o primeiro, **operando destino**: registrador ou posição de memória onde o resultado é armazenado; o conteúdo inicial é modificado;
- o segundo, **operando fonte**: não modificado pela instrução;
- os operandos são separados por uma vírgula.

No caso de **diretivas**, o campo de operandos contem mais informações acerca da diretiva.

- **Campo de comentário:**

- um ponto-e-vírgula ( ; ) marca o início deste campo;
- o Montador ignora tudo após o este marcador;
- comentários são opcionais.

***Uma boa prática de programação é comentar tudo e incluir a informação acerca da idéia por trás da codificação (o algoritmo).***

Exemplos:

```
MOV CX,0          ;movimenta 0 para CX (óbvio!)
MOV CX,0          ;CX conta no. de caracteres, inicialmente vale 0
;
;                (linhas em branco: separação)
;
;inicialização dos registradores      (linha inteira de comentário)
```

## 3.2 Formato de dados, variáveis e constantes

### • Números:

Exemplos:

- binário:            1110101b    ou    1110101B
- decimal:           64223    ou    64223d    ou    64223D  
1110101 é considerado decimal (ausência do B)  
-2184D (número negativo)
- hexa:                64223h    ou    64223H  
0FFFFh    começa com um decimal e termina com **h**  
1B4Dh

Exemplos de números **ilegais**:

- 1,234            caracter estranho (vírgula)
- FFFFh          não começa por número de 0 a 9  
difícil distinguir do nome de uma variável
- 1B4D            não termina com h ou H

### • Caracteres ASCII:

Caracteres **isolados** ou **strings de caracteres** devem estar escritos dentro de aspas simples ( ' ) ou duplas ( " ).

Exemplos:

" A " ou ' A '  
'ola, como vai'  
"EXEMPLO"

- **Variáveis:**

**Variável** é um **nome simbólico** para um dado atualizável pelo programa.

- cada variável possui um **tipo** e recebe um **endereço** de memória;
- usa-se **pseudo-instruções** para definir o tipo da variável;
- o Montador atribui o endereço de memória.

Pseudo-instrução	Entende-se por
<b>DB</b>	define byte (8 bits)
<b>DW</b>	define word (16 bits, 2 bytes consecutivos)
<b>DD</b>	define doubleword (2 palavras, 4 bytes consecutivos)
<b>DQ</b>	define quadword (4 palavras, 8 bytes consecutivos)
<b>DT</b>	define ten bytes (10 bytes consecutivos)

- **Definição de variáveis de tipo byte:**

**Nome      DB      valor\_ inicial**

Exemplos:

Alfa	DB	0	;equivale a 00h
A	DB	10h	
B	DB	0150h	; <b>ilegal</b> , por que?
BIT	DB	?	; não inicializada

### - Definição de variáveis de tipo word:

Nome	DW	valor_inicial
------	----	---------------

Exemplos:

WORD1	DW	0h	;equivale a 0000h
CONTA	DW	0150h	; <b>OK!</b> , por que?
C	DW	?	;não inicializada
WORD1	DW	1234h	;byte baixo 34h, endereço WORD1 ;byte alto 12h endereço WORD1+1

- **Array**: sequência de bytes ou words **consecutivos** na memória

- armazenar dados relacionados
- armazenar caracteres ASCII organizados (ex: texto)

Exemplos:

BYTE_ARRAY	DB	10h,20h,30h
WORD_ARRAY	DW	1000h,123h,0h,0FFFFh

Um **array** pode conter um **string** de caracteres, sendo definido como:

LETRAS	DB	'abC'	;é equivalente aos caracteres ASCII
LETRAS	DB	61h,62h,43h	;depende se maiúscula ou minúscula

- **Combinação de caracteres e números numa mesma definição:**

```
MENSAGEM    DB    'Alo!', 0Ah,0Dh,'$'
```

O caracter '\$' marca o fim de um *string* de caracteres e não é exibido.

• **Constantes:**

**Constante** é um **nome simbólico** para um dado de valor constante, que seja muito utilizado num programa.

Para atribuir um nome a uma constante, utiliza-se a pseudo-instrução **EQU** (*equates* -> igual a) e a sintaxe:

```
Nome          EQU    valor_da_constante
```

Exemplos:

```
LF            EQU    0Ah    ;caracter Line Feed como LF
CR            EQU    0Dh    ;caracter Carriage return como CR
LINHA1        EQU    'Digite seu nome completo'

MENSAGEM      DB     LINHA1,LF,CR
```

Obs:

- Constantes não geram código de máquina.



### 3.3 A estrutura do programa - algumas instruções iniciais

#### MOV destino,fonte

Usada para transferir dados entre:

- registrador e registrador
- registrador e uma posição de memória
- mover um número diretamente para um registrador ou posição de memória

Combinações legais de operandos:

Operando fonte	Operando destino		
	Registrador de dados	Registrador de segmento	Posição de memória
Reg. de dados	sim	sim	sim
Reg. de segmento	sim	<b>não</b>	sim
Posição de memória	sim	sim	<b>não</b>
Constante	sim	<b>não</b>	sim

Exemplos de instruções válidas:

MOV AX,WORD1 ;movimenta o conteúdo da posição de memória WORD1  
;para o registrador AX

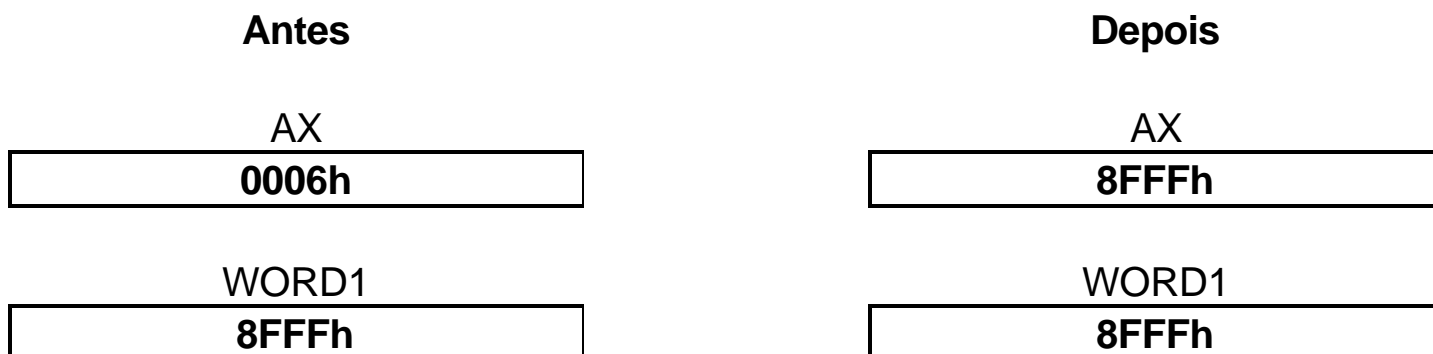
MOV AH,'A' ;transfere o caracter ASCII 'A' para AH

MOV AH,41h ;idem anterior: 41h corresponde ao caracter A

MOV AH,BL ;move o conteúdo do byte baixo de BX  
;o byte alto de AX

MOV AX,CS ;transfere uma cópia do conteúdo de CS para AX

Graficamente: suponha a instrução **MOV AX,WORD1**



**Obs:** para a instrução **MOV** não é permitido operar de posição de memória para posição de memória diretamente, por motivos técnicos do 8086.

Por exemplo:

```

MOV WORD1,WORD2 ;instrução inválida
                  ;esta restrição é contornada como segue
                  ;
                  ;
                  ;
MOV AX,WORD2      ;primeiro o conteúdo de WORD2 vai para AX
MOV WORD1,AX      ;depois, o conteúdo de AX é movido para a
                  ;posição de memória WORD1
  
```

## XCHG destino,fonte

Usada para trocar dados (nos dois sentidos) entre:

- registrador e registrador
- registrador e uma posição de memória
- não é permitido trocas diretas entre posições de memória

Combinações legais de operandos:

Operando fonte	Operando destino	
	Registrador de dados	Posição de memória
Reg. de dados	sim	sim
Reg. de segmento	<b>não</b>	<b>não</b>
Posição de memória	sim	<b>não</b>

Exemplos de instruções válidas:

XCHG AX,WORD1 ;troca o conteúdo da posição de memória WORD1  
;com o do registrador AX

XCHG AH,BL ;troca o conteúdo do byte baixo de BX com o  
;do byte alto de AX

Graficamente: suponha a instrução **XCHG AH,BL**

**Antes**

AH	AL
<b>14h</b>	<b>FFh</b>

BH	BL
<b>C2h</b>	<b>E0h</b>

**Depois**

AH	AL
<b>E0h</b>	<b>FFh</b>

BH	BL
<b>C2h</b>	<b>14h</b>

**ADD destino,fonte**  
**SUB destino,fonte**

Usadas para adicionar (ou subtrair) dados entre:

- registrador e registrador
- registrador e uma posição de memória
- adicionar (ou subtrair) um número diretamente a (de) um registrador ou posição de memória

Combinações legais de operandos:

Operando fonte	Operando destino	
	Registrador de dados	Posição de memória
Reg. de dados	sim	sim
Posição de memória	sim	<b>não</b>
Constante	sim	sim

Exemplos de instruções válidas:

ADD AX,BX ;soma o conteúdo de BX com AX, resultado em AX

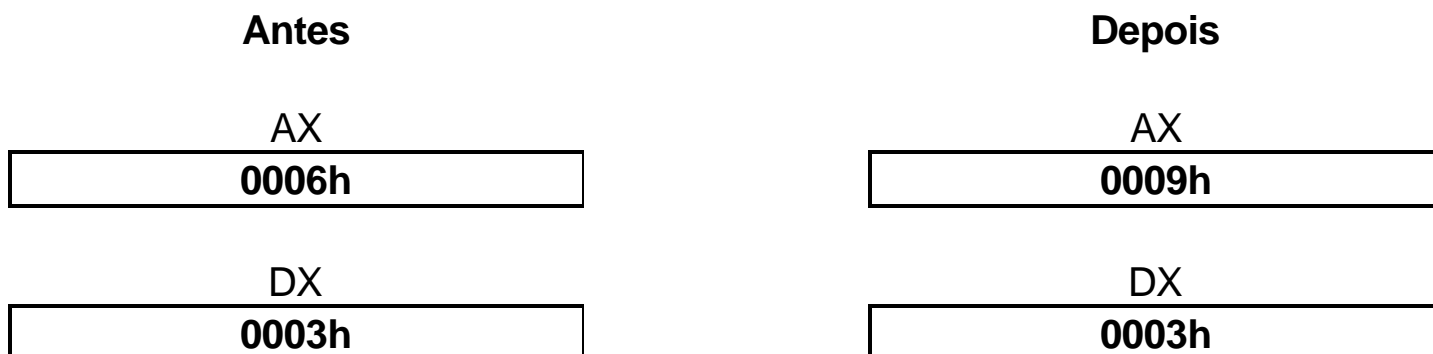
ADD BX,AX ;soma o conteúdo de AX com BX, resultado em BX

ADD AX,WORD1 ;soma o conteúdo da posição de memória WORD1  
;ao do registrador AX, resultado em AX

SUB WORD2,AX ;subtrai o conteúdo do registrador AX do conteúdo da  
;posição de memória WORD2, resultado em WORD2

SUB BL,5 ;subtrai a quantidade 5 decimal do conteúdo de BL

Graficamente: suponha a instrução **ADD AX,DX**



**Obs 1:**

ADD BYTE1,BYTE2

;instrução inválida  
;esta restrição é contornada como segue

;  
;  
;

MOV AL,BYTE2

ADD BYTE1,AL

;primeiro o conteúdo de BYTE2 vai para AL  
;depois, o conteúdo de AL é somado ao da  
;posição de memória BYTE1, resultado final  
;em BYTE1

**Obs 2:** o resultado de **SUB**, se for negativo, estará armazenado no registrador destino em **complemento de 2**.

**INC destino**  
**DEC destino**

Usadas para adicionar 1 (incrementar) ou subtrair 1 (decrementar) ao/do conteúdo de:

- um registrador
- uma posição de memória

Exemplos:

INC CX ;incrementa o conteúdo de CX

INC WORD1 ;incrementa o conteúdo da posição de memória WORD1

DEC BYTE2 ;decrementa o conteúdo da posição de memória BYTE2

DEC CL ;decrementa o conteúdo de CL (byte baixo de CX)

Graficamente: suponha a instrução **INC BYTE1**

**Antes**

BYTE1

**06h**

**Depois**

BYTE1

**07h**

## NEG destino

Usada para **substituir** o conteúdo *destino* pelo seu **complemento de 2**, operando sobre:

- um registrador
- uma posição de memória

Exemplos:

NEG BX ;gera o complemento de 2 do conteúdo de BX

NEG WORD1 ;idem, no conteúdo da posição de memória WORD1

Graficamente: suponha a instrução **NEG BX**

**Antes**

BX

**0002h**

**Depois**

BX

**FFFEh**

## Tradução de expressões matemáticas em Linguagem de Alto Nível para Linguagem Montadora

Exemplo1:	<b>B = A</b>	(equivalente a B recebe A)
tradução:	MOV AX,A	;transfere o conteúdo da posição de memória ;A para AX e
	MOV B,AX	;transfere AX para a posição de memória B
Exemplo 2:	<b>A = 5 - A</b>	
tradução	NEG A	;gera o complemento de 2 da posição de ;memória A e
	ADD A,5	;realiza (-A) + 5, que equivale a 5 - A
Exemplo 3:	<b>A = B - 2A</b>	
tradução	MOV AX,B	;AX contem a variável B
	SUB AX,A	;AX contem B - A
	SUB AX,A	;AX contem B - 2A
	MOV A,AX	;movimenta o resultado para A



## A estrutura de um programa em Linguagem Montadora

### • Modelos de memória

O tamanho que os segmentos de código e de dados devem ter é especificado pelo **modelo de memória** por meio da diretiva **.MODEL**.

Sintaxe: **.MODEL modelo\_de\_memória**

Modelo	Descrição
SMALL	Código em 1 segmento; Dados em 1 segmento
MEDIUM	Código em mais de 1 segmento; Dados em 1 segmento
COMPACT	Código em 1 segmento; Dados em mais de 1 segmento
LARGE	Código em mais de 1 segmento; Dados em mais de 1 segmento; Nenhum array maior que 64 Kbytes
HUGE	Código em mais de 1 segmento; Dados em mais de 1 segmento; Arrays maiores que 64 Kbytes

### Obs:

- Ao menos que haja muitas linhas de programa (muito código) ou muitos dados, o modelo apropriado é o **SMALL**.
- A diretiva **.MODEL** deve vir antes de qualquer definição de segmento.

- **Segmento de dados**

- Contem a definição e declaração das **variáveis**.
- Pode-se também fazer a atribuição de símbolos para **constantes**.

Sintaxe:                   **.DATA**

Exemplo:

```
.DATA
WORD1      DW      A8h
BYTE1      DB      5
MENSAGEM   DB      'Isto e uma mensagem'
LF         EQU     0Ah
```

- **Segmento de pilha (*stack segment*)**

- Reserva um bloco de posições de memória consecutivas para armazenar a pilha
- Deve ter espaço suficiente para suportar a pilha no seu máximo tamanho

Sintaxe:                   **.STACK tamanho**

Exemplo:

```
.STACK 100h                   ;reserva 100h bytes para a área de pilha, um
                              ;tamanho razoável para a maioria das aplicações
```

- **Segmento de código**

- Contem propriamente as **instruções** do programa
- Dentro do segmento de código, as instruções são organizadas em ***procedures*** ou procedimentos.

Sintaxe:                   **.CODE**

Exemplo:

```
.CODE
nome      PROC
;
;corpo da procedure -> instruções
;
nome ENDP
;
;outras procedures seguem abaixo, se existirem
```

onde:

- **nome** -> identificação da *procedure*
- **PROC e ENDP** -> **pseudo-instruções** usadas para delimitar a *procedure*
- para um programa simples, não há necessidade de se definir a *procedure*.

## Exemplo de uma estrutura de programa assembly completa:

```
TITLE nome_do_programa
.MODEL    SMALL
.STACK 100h
.DATA
;
;definição dos dados: variáveis e constantes
;
.CODE
EXEMPLO PROC
;
;seqüência de instruções
;
EXEMPLO ENDP
;
;segue outras procedures
;
END EXEMPLO
```

### Obs:

- na primeira linha tem-se a diretiva **TITLE** seguida do nome do programa;
- na última linha tem-se a diretiva **END**, seguida do nome da *procedure* principal;
- se não houver definição de *procedure*, usa-se apenas **END**.

### 3.4 Instruções de entrada e saída

**IN e OUT** -> instruções *Assembly* para acessar portas de E/S para periféricos

Não são utilizadas na maioria das aplicações:

- (1) os endereços das portas de E/S variam conforme o modelo do PC
- (2) **é mais fácil utilizar o BIOS ou o DOS para funções de E/S**

Para acessar as rotinas de E/S do BIOS ou DOS utiliza-se a instrução:

**INT número\_de\_interrupção**

Obs: o programa em curso é interrompido, passando o controle para o DOS, que realiza a operação de E/S e retorna o controle para o programa.

Exemplo:

**INT 21h** ;acessa um grande número de funções de E/S do DOS

#### **Algumas funções DOS de E/S:**

Função **1h**: Entrada de um caracter simples pelo teclado

Acesso: AH = 1h

Resultado: AL = código ASCII do caracter digitado no teclado

Função **2h**: Exibição de caracter simples no monitor de vídeo

Acesso: AH = 2h

DL = código ASCII do caracter a exibir

Resultado: exibição na tela do monitor

**Exemplos:**

a) Trecho padrão de programa para providenciar a entrada de um caracter ASCII pelo teclado:

```

MOV    AH,1h    ;prepara para entrar caracter pelo teclado
                ;o processador espera até que o usuário
                ;digite o caracter desejado
INT    21h      ;após a digitação, caracter ASCII em AL
                ;se um caracter não-ASCII for digitado, AL = 0h

```

**Obs:** o caracter teclado também aparece no monitor, por causa do DOS.

b) Trecho padrão de programa para providenciar a saída de um caracter ASCII para o monitor de vídeo:

```

MOV    AH,2h    ;prepara para exibir caracter no monitor
MOV    DL,'?'    ;o caracter é '?'
INT    21h      ;exibe (monitor apresenta '?')
                ;após a exibição, o cursor da tela avança para a
                ;próxima posição da linha (se já for atingido o fim
                ;da linha, vai para o início da próxima linha)

```

**Obs:** também se pode exibir caracteres ASCII de controle:

<b>Código ASCII</b>	<b>Símbolo</b>	<b>Função</b>
07h	BEL	<i>Bell</i> (som de bip)
08h	BS	<i>Back Space</i> (espaço para trás)
09h	HT	<i>Tab</i> (tabulação)
0Ah	LF	<i>Line Feed</i> (mover para uma nova linha)
0Dh	CR	<i>Carriage Return</i> (ir para o início da linha)

### 3.5 Criando e rodando um programa

#### Especificação do programa ECO DO TECLADO NA TELA:

- ler um caracter do teclado
- exibir o caracter lido na próxima linha da tela do monitor
- retornar ao DOS

Escrevendo as partes:

a) O programa estimula o usuário a interagir apresentando um '?':

```
MOV AH,2      ;funcao DOS para exibir caracter
MOV DL,'?'    ;caracter '?'
INT 21H       ;exibir
```

b) Lendo o caracter teclado pelo usuário e salvando-o em num registrador:

```
MOV AH,1      ;funcao DOS para leitura de caracter
INT 21H       ;caracter e' lido em AL
MOV BL,AL     ;salvando-o em BL
```

c) Movendo o cursor da tela para o início da próxima linha:

```
MOV AH,2      ;funcao DOS para exibir caracter
MOV DL,0DH    ;caracter ASCII <CR> - return
INT 21H       ;executando
MOV DL,0AH    ;caracter ASCII <LF> - line feed
INT 21H       ;executando
```

d) Recuperando o caracter lido e exibindo-o:

```
MOV DL,BL     ;recuperando o caracter salvo
INT 21H       ;exibir
```



**O programa ECO completo:**

```
TITLE PGM4_1: PROGRAMA DE ECO DO TECLADO NA TELA
.MODEL SMALL
.STACK 100H
.CODE
MAIN PROC
;
;apresentacao do prompt '?'
    MOV AH,2      ;funcao DOS para exibir caracter
    MOV DL,'?'   ;caracter '?'
    INT 21H      ;exibir
;
;entrada do caracter pelo teclado
    MOV AH,1      ;funcao DOS para leitura de caracter
    INT 21H      ;caracter e' lido em AL
    MOV BL,AL     ;salvando-o em BL
;
;movendo de linha
    MOV AH,2      ;funcao DOS para exibir caracter
    MOV DL,0DH    ;caracter ASCII <CR> - return
    INT 21H      ;executando
    MOV DL,0AH    ;caracter ASCII <LF> - line feed
    INT 21H      ;executando
;
;exibindo na tela o caracter lido: efeito de ECO
    MOV DL,BL     ;recuperando o caracter salvo
    INT 21H      ;exibir
;
;retorno ao DOS
    MOV AH,4CH    ;funcao DOS para saida
    INT 21H      ;saindo
MAIN ENDP
END MAIN
```



## Como obter o programa **ECO.EXE** executável.

1. Edite o program ECO utilizando um editor de texto simples, com saída em texto ASCII. Sugestão: use o **EDIT** do DOS. O arquivo (texto ASCII) deve ter a extensão **.ASM**

```
C:\ > EDIT ECO.ASM <enter>
```

2. Rode o programa Montador **TASM** (Borland). Como resultado, aparece em seu diretório de trabalho um arquivo ECO.OBJ

```
C:\ > TASM ECO.ASM <enter>
```

3. Rode o programa Lincador **TLINK**. Como resultado, aparece em seu diretório de trabalho um arquivo ECO.EXE.

```
C:\ > TLINK ECO.OBJ <enter>
```

4. Rode o programa **ECO.EXE**, respondendo ao '?' com uma letra K, por exemplo.

```
C:\ > ECO.EXE <enter>
```

```
?K
```

<- letra K digitada pelo usuário

```
K
```

<- eco da letra K aparece na tela

```
C:\ >
```

<- note que o controle retorna ao DOS

Tente com outras letras ou procure modificar o programa para obter outros efeitos com caracteres digitados no teclado.

## Mais funções DOS de E/S:

Função **4Ch**: Termina o processo corrente e transfere controle para o DOS

Acesso: AH = 4Ch

Resultado: saída para o DOS

Função **9h**: Exibição de *string* de caracteres no monitor de vídeo

Acesso: AH = 9h  
DX = offset do endereço onde começa o *string*

Resultado: *string* exibido

**Obs:** o *string* de caracteres deve terminar com o caracter '\$', que marca o fim da sequência e não é exibido.

Para exibição de um ***string de caracteres*** há dois problemas:

a) **DS** inicialmente não está apontando para o segmento de dados do programa recém iniciado (DS ainda aponta para algum segmento de dados do DOS);

b) deve-se colocar em **DX** o *offset* do endereço do *string* que queremos exibir

## Como apontar DS para o segmento de dados do programa?

**@DATA** -> palavra reservada para obter o número do segmento de dados definido pela diretiva **.DATA**, que contem as variáveis e constantes.

Exemplo: para inicializar corretamente **DS** para o programa corrente

```
.DATA
```

```
...
```

```
.CODE
```

```
MOV     AX,@DATA      ;coloca o número do segmento de dados em AX
```

```
MOV     DS,AX         ;pois DS não pode receber @DATA diretamente
```

**Obs:** o programa Montador traduz o nome **@DATA** pelo número de segmento onde se encontram os dados definidos pela diretiva **.DATA**.

## Como colocar em DX o *offset* do endereço de um *string* a exibir?

### LEA destino,fonte

Significa **Load Effective Address** -> coloca uma cópia do **offset** do endereço da posição de memória fonte no registrador destino.

Exemplo:

```
.DATA
```

```
MENSAGEM  DB  'Adoro ISB!$'
```

```
...
```

```
.CODE
```

```
LEA  DX,MENSAGEM      ;DX carregado com o offset de MENSAGEM
```

**Obs:** após esta operação, DX conterá o *offset* da posição de memória onde inicia o *string* MENSAGEM

**Programa para imprimir um *string* de caracteres:**

```

TITLE  PROGRAMA PARA IMPRESSAO DE 'STRING'
.MODEL  SMALL
.STACK 100H
.DATA
MENSAGEM      DB   'ALO! Como voces estao indo!$'
.CODE
MAIN  PROC
;
;inicializando o registrador DS
;
      MOV AX,@DATA
      MOV DS,AX           ;segmento de dados inicializado
;
;obtendo o offset da posição de memória de MENSAGEM
;
      LEA DX,MENSAGEM   ;offset do endereço vai para DX
;
;exibindo a MENSAGEM
;
      MOV AH,9           ;funcao DOS para exibir 'string'
      INT 21H           ;exibindo
;
;retorno ao DOS
;
      MOV AH,4CH        ;funcao DOS para saida
      INT 21H          ;saindo
MAIN  ENDP
      END MAIN

```

**Exercício:** Programa de conversão de letra minúscula para maiúscula.

**Especificação do programa:**

- apresente ao usuário uma mensagem do tipo  
**Entre com uma letra minúscula:**
- ler um caracter do teclado (não é necessário testar se é letra)
- apresente uma segunda mensagem do tipo  
**Em maiuscula ela fica:**
- apresente em seguida a letra convertida
- retornar ao DOS

**Sugestão 1:** Caracteres ASCII *Carriage Return* e *Line Feed* como variáveis.

CR	EQU	0DH
LF	EQU	0AH

**Sugestão 2:** Como exibir um *string* de caracteres juntamente com uma variável?

MENSAGEM2	DB	CR,LF,'Em maiuscula ela fica: '
CHAR	DB	?,'\$'

Obs: quando for exibida a MENSAGEM2, INT 21h continua ativa pois não há o caracter '\$', permitindo que a variável CHAR seja exibida em seguida.





**Uma possível solução:**

```

TITLE PROGRAMA PARA CONVERSAO DE LETRA MINUSC./MAIUSC.
.MODEL SMALL
.STACK 100H
.DATA
CR      EQU      0DH
LF      EQU      0AH
MENSAGEM1      DB      'Entre com uma letra minuscula: $'
MENSAGEM2      DB      CR,LF,'Em maiuscula ela fica: '
CHAR          DB      '?,$'
.CODE
MAIN  PROC
;inicializando o registrador DS
      MOV AX,@DATA
      MOV DS,AX          ;DS inicializado
;exibindo a MENSAGEM1
      LEA DX,MENSAGEM1  ;offset do endereco de MENSAGEM1 em DX
      MOV AH,9          ;funcao DOS para exibir 'string'
      INT 21H          ;exibindo
;entrada do caracter e conversao para maiuscula
      MOV AH,1          ;funcao DOS para leitura de caracter
      INT 21H          ;entrada
      SUB AL,20H        ;conversao ASCII de minuscula/maiuscula
      MOV CHAR,AL       ;salvando caracter na variavel CHAR
;exibindo a MENSAGEM2 e o caracter convertido
      LEA DX,MENSAGEM2  ;offset do endereco de MENSAGEM1 em DX
      MOV AH,9          ;funcao DOS para exibir 'string'
      INT 21H          ;exibindo
;retorno ao DOS
      MOV AH,4CH        ;funcao DOS para saida
      INT 21H          ;saindo
MAIN  ENDP
      END MAIN

```

## Exercícios sugeridos:

1) Escreva um programa para (a) exibir um '?', (b) ler dois dígitos decimais cuja soma seja menor que 10 e (c) exibir sua soma na próxima linha.

Sugestão:

**?27**

<- em negrito, a entrada do usuário

A soma de 2 e 7 vale 9

2) Escreva um programa para (a) questionar o usuário sobre suas iniciais (três por exemplo), (b) ler as iniciais e (c) exibi-las de cima para baixo, em linhas separadas e na margem esquerda da tela.

3) Escreva um programa para ler um dígito hexadecimal de A a F (maiúsculo) e exibi-lo em decimal na próxima linha. Utilize mensagens convenientes.

Sugestão:

Entre um dígito hexa: **C**

<- em negrito, a resposta à mensagem

O seu valor decimal vale: 12

4) Escreva um programa que leia três iniciais, exiba-as de forma centrada dentro de uma moldura de asteriscos de 11 x 5 (horizontal x vertical) e por fim produza um bip no computador.

Sugestão: declare a moldura de asteriscos e as iniciais como um *string*.

Obs: adapte os programas 2 e 4 para ler 2, 4, 5 ou mais iniciais em função de seu nome.