Aula 07

Já executamos alguns comandos de adição, subtração, divisão, multiplicação, movimento e exibição de dados utilizando o DEBUG (modo DOS). Com ele, tivemos contato com as INSTRUÇÕES, seus OPCODES e os respectivos MNEMÔNICOS (instruções).

No final da última aula, vimos a seqüência de edição, compilação, ligação e execução de programas em Assembly.

Neste processo de desenvolvimento de programas em linguagem Assembly, uma das ferramentas mais utilizadas e importantes é o DEBUG, pois além de nos possibilitar a construção de programas, ele permite a verificação (depois de compilado e ligado) de um programa em execução, permitindo o controle da execução, como por exemplo, execução passo a passo.

Não se importando ainda com a estrutura e os comandos da linguagem Assembly, vamos digitar o programa abaixo que carregará dois registradores, AX e BX, com os valores 000B e 00A1 e fará a soma desses dois valores, e armazenará o resultado em AX. Logo em seguida, carregará o valor 0005 em BX e fará a subtração com o resultado obtido na operação anterior, armazenando o resultado em AX. E por fim, o programa encerrará sua execução, devolvendo o controle ao Sistema Operacional.

Programa 7.1

```
TITLE PRM71:ADICAO E SUBTRACAO
;Programa para ilustração do programa DEBUG
.MODEL SMALL
.STACK 100H
.CODE
MAIN PROC
  MOV AX,000BH  ;carrega o primeiro valor em AX
  MOV BX,00A1H  ;carrega o segundo valor em BX
  ADD AX,BX     ;soma os dois valores e armazena
;                o resultado em AX
  MOV BX,0005H  ;carrega o terceiro valor em BX
  SUB AX,BX     ;subtrai o terceiro valor do resultado
;                da operação anterior
  MOV AH,4CH    ;código para devolver o controle p/ DOS
  INT 21H       ;interrupção que executa a função em AH
MAIN ENDP
  END MAIN
```

VAMOS COMPILAR, LIGAR E EXECUTAR O PROGRAMA.

COMPILAÇÃO:

```
C:\TASM>TASM PRM71.ASM <enter>
Turbo Assembler  Version 4.1  Copyright (c) 1988, 1996
Borland International

Assembling file:   PRM71.ASM
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  413k
```

LIGAÇÃO:

```
C:\TASM>TLINK PRM71.OBJ <enter>
Turbo Link  Version 7.1.30.1. Copyright (c) 1987, 1996
Borland International
```

EXECUÇÃO:

```
C:\TASM>PRM71 <enter>

C:\TASM>_
```

Como podemos perceber, nada visível aconteceu.
Mas as operações foram executadas, adequadamente?
Como podemos verificar isto?

Uma das formas é utilizando o programa DEBUG

Digite na linha de comando logo após a execução:

```
C:\TASM>DEBUG PRM71.EXE
-
```

Agora o programa DEBUG está em execução. Mas executando o quê?
O programa PRM71.EXE !!

Utilizando os comandos do programa DEBUG que já vimos anteriormente,vamos verificar a situação dos registradores:

Tecle R <enter>

```
-R
AX=0000  BX=0000  CX=0011  DX=0000  SP=0100  BP=0000  SI=0000  DI=0000
DS=152B  ES=152B  SS=153D  CS=153B  IP=0000   NV UP EI PL NZ NA PO NC
153B:0000 B80B00        MOV     AX,000B
-
```

Como podemos notar, os registradores de dados estão "zerados" e a próxima instrução a ser executada é a instrução MOV AX,000B
(a primeira instrução do nosso programa PRM71.EXE !!)

Estamos, portanto, "dentro" da execução do programa.

Vamos executar a instrução corrente com o comando T (trace).

```
-T

AX=000B  BX=0000  CX=0011  DX=0000  SP=0100  BP=0000  SI=0000  DI=0000
DS=152B  ES=152B  SS=153D  CS=153B  IP=0003   NV UP EI PL NZ NA PO NC
153B:0003 BBA100        MOV     BX,00A1
-
```

Note que após a execução da instrução corrente (T), o registrador AX agora armazena o valor 000B como queríamos. Veja que a instrução corrente agora é MOV BX,00A1 (segunda instrução no programa PRM71.EXE!!)

E assim, executando instrução a instrução, ou executando um determinado trecho de programa (até posição final, com o comando G) podemos verificar se tudo o que programamos está sendo executado adequadamente.

A título de ilustração, abaixo segue a execução do programa utilizando os comandos T e G.

```
-T
AX=000B  BX=00A1  CX=0011  DX=0000  SP=0100  BP=0000  SI=0000  DI=0000
DS=152B  ES=152B  SS=153D  CS=153B  IP=0006   NV UP EI PL NZ NA PO NC
153B:0006 03C3          ADD     AX,BX
-T
AX=00AC  BX=00A1  CX=0011  DX=0000  SP=0100  BP=0000  SI=0000  DI=0000
DS=152B  ES=152B  SS=153D  CS=153B  IP=0008   NV UP EI PL NZ NA PE NC
153B:0008 BB0500        MOV     BX,0005
```

```
-T
AX=00AC  BX=0005  CX=0011  DX=0000  SP=0100  BP=0000  SI=0000  DI=0000
DS=152B  ES=152B  SS=153D  CS=153B  IP=000B   NV UP EI PL NZ NA PE NC
153B:000B 2BC3          SUB     AX,BX
-T
AX=00A7  BX=0005  CX=0011  DX=0000  SP=0100  BP=0000  SI=0000  DI=0000
DS=152B  ES=152B  SS=153D  CS=153B  IP=000D   NV UP EI PL NZ NA PO NC
153B:000D B44C          MOV     AH,4C
-G

Program terminated normally
-
```

Mesmo agora tendo o controle, podendo verificar o que está ocorrendo no processador a cada instrução, essa forma de trabalho não é muito utilizada (pela visibilidade limitada).

Existe um outro programa DEBUG, chamado TURBO DEBUGGER da Borland. A partir de agora, vamos utilizá-lo.

Entretanto, para fazê-lo, no processo de COMPILAÇÃO-LIGAÇÃO, temos que informar ao compilador e ao ligador (linker) que carregue, juntamente com o código, informações para que o programa TURBO DEBUGGER possa controlar a execução do programa.

Para tanto, no processo de compilação, utilize a diretiva /zi .
E no processo de ligação, utilize a diretiva /v

Dessa forma, a compilação e ligação do programa PRM71.ASM ficaria da seguinte forma:

```
C:\TASM>TASM /zi PRM71.ASM  <enter>
Turbo Assembler  Version 4.1  Copyright (c) 1988, 1996
Borland International

Assembling file:   PRM71.ASM
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  411k

C:\TASM>TLINK /v PRM71.OBJ  <enter>
Turbo Link  Version 7.1.30.1. Copyright (c) 1987, 1996
Borland International

C:\TASM>
```

Pronto! Agora nosso programa PRM71.EXE possui informações que possibilitarão ao TD (Turbo Debugger) gerenciar a execução do código.

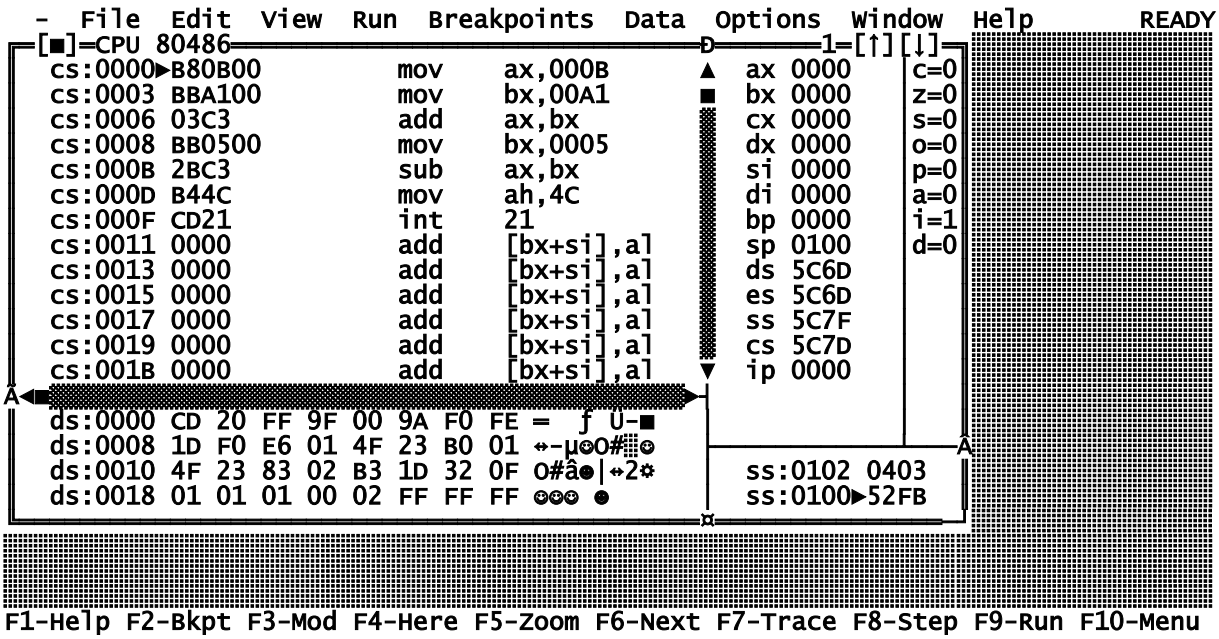
Para executar o programa com o Turbo Debugger, digite na linha de comando do DOS:

TD <nome do programa.exe> <enter>

No nosso caso:

TD PRM71.EXE <enter>

Se tudo der certo, você terá a seguinte tela:

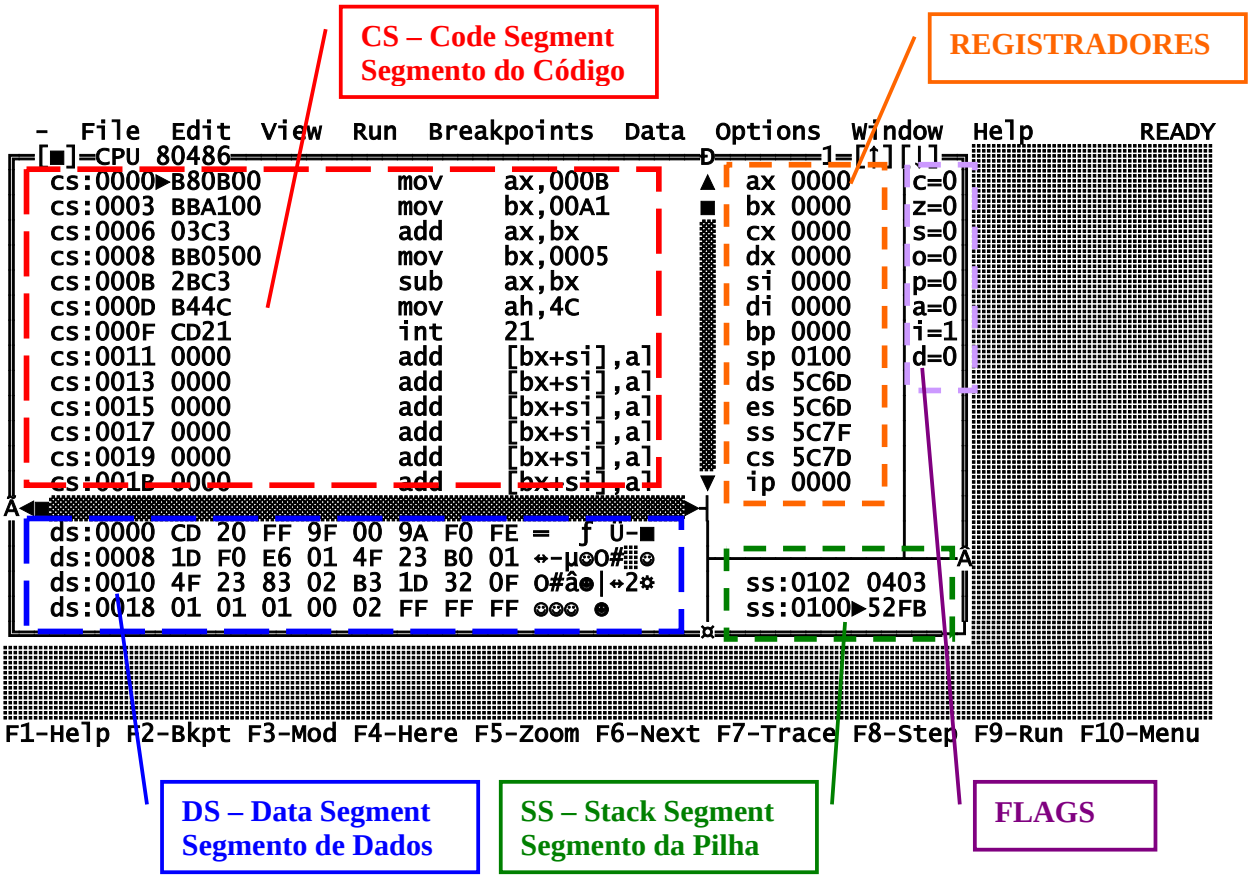


Esta é a tela do TURBO DEBUGGER. Ela é bem semelhante ao TURBO PASCAL!!

Temos na primeira linha uma série de opções de comandos.

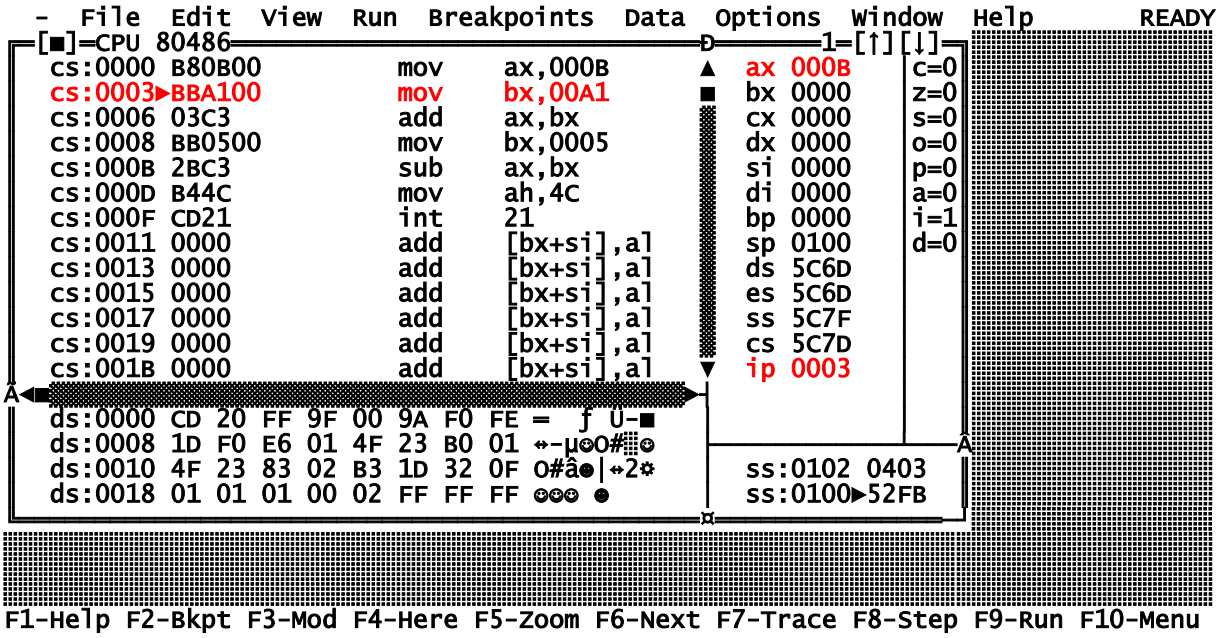
Na última linha as teclas de atalho para os comandos mais freqüentes, tais como: F8 – Step (executa o próximo comando – um passo do programa), F9 – Run (executa o programa até encontrar um final).

Além disso, pode-se ver que existem **áreas de exibição**:



Observe que no segmento do código, o ponteiro (que no segmento de registradores é identificado por *IP – Instruction Pointer*) aponta para a instrução **MOV AX,000B**, ou seja, a primeira instrução... a instrução que será executada.

Vamos executá-la...
Pressione **F8** e observe o que aconteceu...



Teste também os comandos (opção RUN no menu):

Run – Run (Executa todo o código)

Run – Go to cursor (Reset o código (Run – Program Reset). Selecione a 4 linha de instrução com o ponteiro do mouse. Execute novamente o código, com a opção Rn – Go to cursor. Observe que agora ele parou onde você tinha selecionado.

Run – Animated – Reset o código. Execute o programa com Run – Animated. Coloque um tempo igual a 20 e tecle Ok.

Breakpoints – Toggle (estabeleça um ponto de parada no endereço 000B. Depois Reset o programa (Run – Program Reset). E logo em seguida execute-o com Run – Run. Veja que agora o programa é executado até o ponto de parada estabeledido.

Depois, com mais calma, como exercício, explore as outras opções do TD.

Exercício 1.

Crie um programa em Assembly para somar dois valores (A001h e 00C1h). A este resultado, deve ser subtraído o valor de 00FFh. Em seguida, coloque o resultado nos registradores BX e CX. Termine o programa, devolvendo o controle para o DOS.

Depois disso, compile, ligue e execute o programa.

Em seguida, faça novamente a compilação e a ligação com as opções para execução do programa com o Turbo Debugger.

Execute o programa, passo-a-passo, e verifique se as operações são executadas.

ALGUNS COMANDOS BÁSICOS E SINTAXE DA LINGUAGEM ASSEMBLY

a) Linhas de Comando (statements):

```
nome        operação   operando(s) ;comentário
```

Exemplo:

```
START:      MOV        CX,5         ;inicializa o contador
```

Esses campos devem ser separados por espaço(s) ou tab
START é um rótulo.

b) 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 **? . @ _ : $ % .**

Exemplos:

nomes válidos	*nomes inválidos*
LOOP1:	DOIS BITS
.TEST	2abc
@caracter	A42.25
SOMA_TOTAL4	#33
$100	

Observação:
O Montador traduz os nomes por endereços de memória.

c) Dados do Programa

Números:

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

Variáveis:

Variáveis Byte:

```
Nome DB valor_inicial
```

Exemplo:

```
ALPHA  DB  4
TESTE  DB  ?
```

Variáveis Word

```
Nome DW valor_inicial
```

Exemplos:

```
WRD     DW   -2
TESTE   DW   ?
```

Constantes:

EQU (Equates)

```
Nome  EQU   constante
```

Exemplo:

```
LF    EQU   0AH
```

Isto associa a LF (constante) o valor em hexa 0A, que corresponde ao código ascii do caracter *Line Feed*.

d) Instruções Básicas:

Movimentação de dados:

MOV – usado para transferir dados entre registradores, entre registradores e memória e para mover valores concretos (números) para registradores e locais de memória.

Exemplos:

```
MOV AX,BX
MOV AX,'A'
MOV AX,WRD
```

XCHG – troca os valores entre registradores ou registradores e memória.

Exemplos:

```
XCHG AX,BX
XCHG AX,WRD
```

Adição, Subtração, Incremento e Decremento

```
ADD destino,fonte
SUB destino,fonte
```

Exemplos:

```
ADD AX,DX
ADD WRD,AX
SUB AX,BX
SUB WRD,AX
ADD AL,5
```

OPERAÇÃO ILEGAL: ADD BYTE1,BYTE2

Solução:

```
MOV AL,BYTE2
MOV BYTE1,AL
```

```
INC destino
DEC destino
```

Exemplos:

```
INC WORD1
DEC BYTE1
INC AX
INC AL
DEC AX
DEC BL
```

e) Tradução de Linguagem de Alto-Nível para Assembly

Exemplo 1:

Linguagem de Alto Nível:

```
B = A
```

A variável **B** recebe o conteúdo da variável **A**

Tradução:

```
MOV AX,A
MOV B,AX
```

Exemplo 2:

Linguagem de Alto Nível:

```
A = 5 - A
```

A variável **A** recebe o resultado de 5 menos o valor anterior de **A**

Tradução:

```
MOV AX,5
SUB AX,A
MOV A,AX
```

Exemplo 3:

Linguagem de Alto Nível:

```
A = B - 2 x A
```

A variável **A** recebe o resultado de **B** menos duas vezes o valor anterior de **A**

Tradução:

```
MOV AX,B
SUB AX,A
SUB AX,A
MOV A,AX
```

Exercício 2:

Quais nomes abaixo são ilegais para a linguagem Assembly no IBM-PC?

a) TWO_WORDS
b) ?1
c) Two words
d) .@?
e) $145
f) LET'S_GO
g) T = .

Exercício 3:

Quais dos números abaixo são ilegais? Se são legais, diga se serão tratados como binário, decimal ou hexa

a) 246
b) 246h
c) 1001
d) 1,101
e) 2A3h
f) FFFEh
g) 0Ah
h) Bh
i) 1110b

Exercício 4:

Suponha que os dados abaixo estão armazenados, iniciando no offset (deslocamento) 0000h:

```
A   DB  7
B   DW  1ABCh
C   DB  'HELLO'
```

a) Dê o endereço de offset associado às variáveis A,B,C
b) Dê o conteúdo do byte no offset 0002h
c) Dê o conteúdo do byte no offset 0004h
d) Dê o endereço de offset do caracter "O" in "HELLO."

Exercício 5:

Usando apenas MOV, ADD, SUB, INC e DEC, traduza as expressões abaixo em linguagem de alto-nível para linguagem Assembly.

a) A = B – A
b) C = A + B
c) B = 3 X B + 7

ESTRUTURA DE UM PROGRAMA EM LINGUAGEM ASSEMBLY:

```
TITLE NOME:DESCRIÇÃO
.MODEL SMALL
.STACK tamanho em hexa
.DATA
    ;os dados vão aqui
.CODE
MAIN PROC
    ;as instruções vão aqui
MAIN ENDP
    ;outros procedimentos vão aqui
  END MAIN
```

Exemplo de Programa:

Nosso primeiro programa irá mostrar o character “*” na tela.

```
TITLE EX01:MOSTRA UM ASTERISCO
;Primeiro programa exemplo. Mostra um asterisco na tela
.MODEL SMALL
.STACK 100H
.CODE
MAIN PROC
; Exibição do caracter
  MOV AH,2       ;função para exibição de caracter
  MOV DL,'*'     ;caracter a ser exibido
  INT 21H        ;interrupção que executa a função armaz.
;                 em AH (exibir caracter na tela)
; Retorna para o DOS
  MOV AH,4CH     ;função para retorno ao DOS
  INT 21H        ;interrupção que executa a função em AH
MAIN ENDP
  END MAIN
```

Digite este documento em um Editor de texto puro (por exemplo o Bloco de Notas). Salve o documento com um nome significativo e com a extensão .ASM. Por exemplo: Ex01.ASM

Faça a compilação (TASM) e a ligação (TLINK). Em seguida, execute o arquivo e verifique se ele mostra o caracter “*” na tela.

Neste programa foi apresentado o comando INT 21H. Este comando pode ser utilizado para realizar uma série de funções do DOS. A função que será executada tem o seu código (número) armazenado em AH (indicando o que deve ser feito).

Dependendo da função a ser executada, deve-se utilizar outros registradores para guardar a informação de entrada e saída. No nosso exemplo, está sendo executada a função de escrita na saída padrão (código 2 em AH).

Esta função requer algo a ser exibido na saída padrão, que no caso deve, sempre para esta função, ser armazenado em DL (no caso o caracter '*').

Em Resumo:

02H - Função para exibição de um caracter.	
Entradas	AH = 2 DL = código ASCII do caracter a ser exibido
Saídas	AL = código ASCII do caracter exibido

Exercício 6:
Faça um programa para exibir um caracter "☺" e logo em seguida o programa deve soar um bip.
Dica: Código ASCII de ☺ = 01h e Bip = 07h

Vamos montar um segundo programa, que agora irá solicitar a digitação de um caracter. Depois de digitada uma determinada tecla, o caracter correspondente será exibido na tela. Em seguida, vamos exibir novamente este mesmo caracter, separado por um hífen do primeiro.

Programa para Entrada de Caracter:

```
TITLE EX02:RECEBE E MOSTRA UM CARACTER
;Segundo programa exemplo. Recebe um caracter e o exibe
.MODEL SMALL
.STACK 100H
.CODE
MAIN PROC
; Entrada de um Caracter
  MOV AH,1       ;função para entrada de caracter
  INT 21H        ;executa a função em AH
;
; Depois de pressionada uma tecla, seu código ascii é armaz.
; em AL. Se for digitada uma tecla não-caracter, AL recebe 0
; Salva o caracter digitado em BL
;
  MOV BL,AL      ;salva o caracter digitado em BL
;
; Exibe o Hifen
  MOV AH,2       ;função para exibição de caracter
  MOV DL,'-'     ;caracter a ser exibido
  INT 21H        ;interrupção que executa a função armaz.
;                 em AH (exibir caracter na tela)
; Recupera e Exibe o caracter digitado
  MOV DL,BL
  MOV AH,2       ;função para exibição de caracter
  INT 21H        ;interrupção que executa a função armaz.
;                 em AH (exibir caracter na tela)
; Retorna para o DOS
  MOV AH,4CH     ;função para retorno ao DOS
  INT 21H        ;interrupção que executa a função em AH
MAIN ENDP
  END MAIN
```

Como você já pode deduzir, acabamos de envidenciar mais uma função do DOS que pode ser utilizada: entrada de caracter.

01H - Função para Leitura (entrada) de uma tecla.	
Entradas	AH = 1
Saídas	AL = código ASCII da tecla pressionada = 0 se tecla não-caracter

Usamos nos dois programas anteriores uma terceira função: Retorno do controle ao Sistema Operacional.

Acho que vocês já sabem do que estou falando:

4CH - Função para retorno ao S.O. (DOS)	
Entradas	AH = 4CH AL = código de retorno
Saídas	Nenhuma

Exercício 7:

Faça um programa programa para receber uma tecla (caracter) e mostrá-lo na linha seguinte entre asterisco. Por exemplo: Se você teclou "A", ele exibirá:

```
Ex07 <enter>
A
*A*
```

Dica: o caracter "line feed" = 0Ah e
O caracter "carriage return" = 0Dh

Vamos agora, montar um programa para exibir uma seqüência de caracteres (conhecida como String).

Para fazê-lo, devemos armazenar esta String em um local de memória. Digamos em uma variável chamada MSG:

```
MSG     DB      'HELLO!$'
```

Note que o último caracter da mensagem é "$". Este caracter indica o final da string para o S.O. e ele não é exibido.

Bem, mas agora como podemos exibir esta mensagem?

Devemos utilizar uma função do DOS que faz esta tarefa: Exibição de uma String.

09H – Função para exibição de um String	
Entradas	AH = 9 DX = endereço de offset da String
Saídas	Nenhuma

Temos um problema: como vamos carregar em DX o endereço de offset da String?

Uma das formas de se fazer isso é utilizando o comando LEA.

LEA é acrônico de Load Effective Address, ou seja, carrega o endereço efetivo de algo (offset).

```
LEA destino, fonte
```

Assim, ele colocará o endereço da fonte no destino (por exemplo o registrador DX).

Voltando ao nosso exemplo, onde colocamos a nossa mensagem na variável MSG, teríamos o seguinte comando:

```
LEA DX,MSG
```

Mas ainda temos um problema...

Quando um programa é carregado na memória, o DOS cria e faz uso de um segmento de memória de 256 bytes que contem informações sobre o programa (este segmento é conhecido como PSP – program segment prefix).

Com isso, o DOS coloca o número deste segmento nos registradores DS e ES antes de executar o programa.

O resultado disso é que, no início do programa, DS não contém o endereço do segmento de dados.

Dessa forma, devemos colocar em DS o endereço correto do segmento de dados corrente. Para tanto, usamos:

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

@DATA é o nome do segmento de dados definido em .DATA
O Assembly traduz @DATA para o número do segmento.

Um outro detalhe é que não podemos mover este número diretamente para DS. Por isso temos que utilizar duas instruções.

Pronto! Agora é só construir o programa!!

Programa para exibir uma mensagem

```
TITLE MSG01:MOSTRA UMA MENSAGEM
;Terceiro programa exemplo. Mostra uma mensagem na tela
.MODEL SMALL
.STACK 100H
.DATA
MSG   DB  'HELLO!$'
.CODE
MAIN PROC
; Inicializa DS
  MOV AX,@DATA
  MOV DS,AX      ;inicializa DS
;
;exibe mensagem
  LEA DX,MSG     ;carrega o endereço da mensagem
  MOV AH,9       ;função para exibição de string
  INT 21H        ;executa a função em AH
;
; Retorna para o DOS
  MOV AH,4CH     ;função para retorno ao DOS
  INT 21H        ;interrupção que executa a função em AH
MAIN ENDP
  END MAIN
```

Exercício 8:

Escreva um programa para:
(a) mostrar um character “?”;
(b) ler dois dígitos decimais cuja soma seja < 10;
(c) mostre-os e o resultado de sua soma na linha seguinte, com uma mensagem apropriada, por exemplo:

```
?27
A SOMA DE 2 E 7 EH 9
```

Exercício 9:

Escreva um programa para:
(a) Apresentar uma mensagem: “entre com três iniciais:”
(b) Ler a primeira, a segunda e a terceira iniciais
(c) Apresentá-las, uma em cada linha, conforme exemplo abaixo:

```
Entre com tres iniciais: JFK
J
F
K
```

Exercício 10:

Escreva um programa para apresentar uma caixa de 10x10 asteriscos.
Dica: declare uma string com 10 ‘’ no segmento de dados*