

# LABORATÓRIO DE LINGUAGEM DE MONTAGEM

## MACROS e GERENCIAMENTO DE MEMÓRIA

Capítulos 13 e 14 do livro-texto, págs. 257 a 308

### ❖ MACRO:

- é um bloco de texto que recebe um **nome especial**
- consiste de instruções, diretivas, comentários ou referências à outras macros
- a macro é chamada no momento da montagem e é expandida
- o montador copia o bloco texto na posição de cada chamada de macro
- as expansões podem ser vistas no arquivo .LST
- uso:
  - ✓ criar **novas instruções**
  - ✓ operacionalizar tarefas freqüentes e repetitivas

### Vantagens x desvantagens

	<b>Macros</b>	<b>Subrotinas</b>
Tempo para montar	maior	menor
Quantidade de código de máquina (.EXE)	maior	menor
Tempo de execução	menor	maior
Pequenas tarefas	x	
Grandes tarefas		x

### ❖ Rótulos locais (*local labels*):

diretiva     **LOCAL** lista\_de\_labels

informa ao montador que os labels internos são locais

### ❖ Biblioteca de macros (*macro library*):

diretiva     **INCLUDE** caminho\nome\_do\_arquivo\_texto

informa ao montador onde se encontra uma coleção de macros pré-definidas

### ❖ EXEMPLO

Crie uma macro para realizar a cópia de *strings*

```
COPY          MACRO    FONTE,DESTINO,QUANTIDADE
LOCAL        REPETE
SALVA_REGS   CX,SI,DI    ;existente na biblioteca
LEA          SI,FONTE
LEA          DI,DESTINO
CLD
MOV          CX,QUANTIDADE

REPETE:
MOVSB
LOOP        REPETE
RESTAURA_REGS  DI,SI,CX    ;existente na biblioteca
ENDM
```

### Exemplo de chamada:

```
...
;definição da macro COPY
;definição de outras macros
INCLUDE  A:\MY_LIB.TXT
; depois da subrotina que lê o nome de um arquivo
COPY    NOME_1,NOME_2,BX
...
;determine o que resulta da expansão desta macro.
```

No arquivo **MY\_LIB.TXT**, parte do texto contem:

```
SALVA_REGS   MACRO    R1,R2,R3
              PUSH R1
              PUSH R2
              PUSH R3
              ENDM

;
RESTAURA_REGS  MACRO    S1,S2,S3
              POP     S1
              POP     S2
              POP     S3
              ENDM
```

### ❖ Programas .COM:

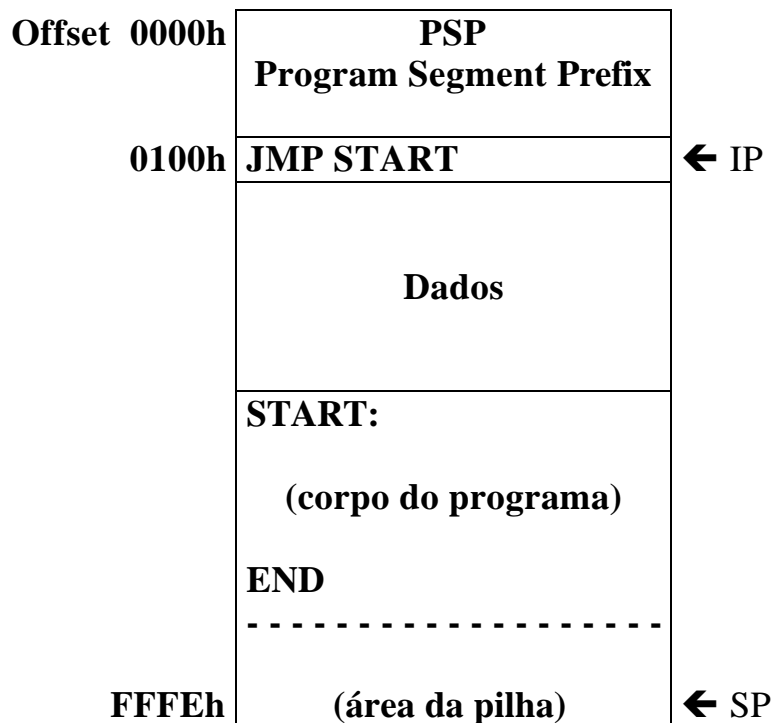
- estrutura simples
- pilha, dados e código ficam localizados todos no mesmo segmento
- ocupam menos espaço do que os equivalentes .EXE
- possuem versatilidade (?) e tamanho limitados -> 1 segmento = 64 kbytes

### ❖ Estrutura de escrita de um programa .COM:

- diretiva **ORG** posição
  - ✓ define a localização do *location counter*, e a partir dele, inicia-se o programa
- só existe um único segmento de código (.CODE)
- dados -> ficam definidos logo no início e o programa deve saltar esta região
- pilha -> o SP aponta para o último word do segmento (FFFEh)  
a pilha cresce em direção ao início

### ❖ Estrutura na memória:

- todo arquivo executável, quando carregado na memória, é precedido pelo PSP



### ❖ Exemplo: criando uma versão .COM

## Versão .EXE

```
TITLE TESTE_1: BOM DIA!  
.MODEL SMALL  
.STACK 100h  
.DATA  
MENSAGEM DB 'BOM DIA!$'  
.CODE  
MAIN PROC  
MOV AX,@DATA  
MOV DS,AX  
MOV AH,9  
LEA DX,MENSAGEM  
INT 21h  
MOV AH,4Ch  
INT 21h  
MAIN ENDP  
END MAIN
```

## Versão .COM

```
TITLE TESTE_2: BOM DIA!  
.MODEL SMALL  
.CODE  
START:  
JMP MAIN  
MENSAGEM DB 'BOM DIA!$'  
MAIN PROC  
MOV AH,9  
LEA DX,MENSAGEM  
INT 21h  
MOV AH,4Ch  
INT 21h  
MAIN ENDP  
END START
```

### Procedimento para montagem:

```
C:\> TASM TESTE_2.ASM
```

```
C:\> TLINK TESTE_2.OBJ
```

```
C:\> EXE2BIN TESTE_2.EXE TESTE_2.COM
```

Pode-se agora executar ambos os arquivos .EXE e .COM

Compare os tamanhos de ambos e verifique que o .COM é bem menor.

Todo arquivo .EXE contem adicionalmente um bloco de cabeçalho de **512 bytes** (*header*), que orienta o DOS durante o seu carregamento na memória e informa detalhes de tamanho dos diversos segmentos e posição das variáveis e rótulos.

### ❖ DEFINIÇÃO COMPLETA DE SEGMENTOS

❖ **Definição simplificada:** uso da diretivas **.MODEL**, **.STACK**, **.DATA** e **.CODE**

❖ **A diretiva SEGMENT**

```
name SEGMENT align combine 'class'  
; diretivas, dados e instruções  
name ENDS
```

Tipos de <i>align</i>	Descrição
PARA	O segmento inicia no próximo parágrafo* disponível
BYTE	O segmento inicia no próximo byte disponível
WORD	O segmento inicia no próximo word disponível
PAGE	O segmento inicia na próxima página** disponível

Tipos de <i>combine</i>	Descrição
PUBLIC	Segmentos com o mesmo nome são concatenados para formar um único bloco contínuo de memória
COMMON	Segmentos com mesmo nome são sobepostos na memória
STACK	Mesmo efeito de PUBLIC, exceto que o offset de todos os dados nele contidos estão referidos ao SS; SP inicia apontando para o final do segmento
AT <i>paragraph</i>	Indica que o segmento deve iniciar-se num parágrafo especificado

Tipo de ' <i>class</i> ' #	Descrição
'CODE'	Especifica que o segmento é de código
'DATA'	Especifica que o segmento é de dados

(\*) Parágrafo: dígito menos significativo do endereço físico em hexa valendo 0.

(\*) Página: 2 dígitos menos significativo do endereço físico em hexa valendo 00.

# Dependendo da seqüência de declarações dos segmentos, o tipo *class* permite definir a ordem dos segmentos na memória.

## ❖ EXEMPLO

```
TITLE TESTE_3: Programa de conversão para maiúscula - MÓDULO1
EXTRN    CONVERTE:NEAR          ;informa que necessita do módulo
                                           ;CONVERTE, alocado no mesmo
                                           ;segmento

S_SEG    SEGMENT    STACK
         DB    100 DUP (0)      ;pilha com 50 words
S_SEG    ENDS
D_SEG    SEGMENT    BYTE    PUBLIC    'DATA'
MSG      DB    'Entre com uma letra minúscula: $'
D_SEG    ENDS
C_SEG    SEGMENT    BYTE    PUBLIC    'CODE'
         ASSUME    CS:C_SEG,DS:D_SEG,SS:S_SEG

MAIN     PROC
         MOV      AX,D_SEG
         MOV      DS,AX        ;inicializa DS
;
IF1      ;condicional que ...
INCLUDE  A:\MY_LIB.TXT
ENDIF
; ... acessa a biblioteca mas apênda apenas as macros chamadas
         MOV      AH,9
         LEA     DX,MSG
         INT     21h          ;exibe a primeira mensagem
         MOV     AH,01h
         INT     21h          ;lê um caracter do teclado
         MOV     BL,AL        ;foi preciso salvar AL em BL !!!!!
         NOVA_LINHA          ;macro para mudança de linha
         CALL    CONVERTE     ;chama a rotina de conversão
         RETORNO_DOS         ;macro de retorno ao DOS

MAIN     ENDP
C_SEG    ENDS
         END      MAIN
```

❖ Obs: As diretivas EXTRN e PUBLIC implementam esquemas de variáveis globais.

## EXEMPLO - continuação

```
TITLE TESTE_4: Programa de conversão para maiúscula – MÓDULO2
PUBLIC  CONVERTE                ;informa que pode ser usada em
                                ;módulo diferente
D_SEG   SEGMENT BYTEPUBLIC  'DATA'
MSG     DB  'Convertendo para maiuscula:  '
LETRA   DB  ?, '$'
D_SEG   ENDS
C_SEG   SEGMENT BYTEPUBLIC  'CODE'
        ASSUME  CS:C_SEG,DS:D_SEG
CONVERTE PROC  NEAR
;converte um caracter existente em AL para maiuscula
;entrada:  AL = caracter
;saida: nenhuma, apenas a exibição na tela do monitor
        PUSH   DX                ;salva DX pois é usado pela função 09h
        AND    BL,0DFh           ;máscara que converte para maiúscula
        MOV    LETRA,BL         ;coloca na posição para exibição
        MOV    AH,09h           ;função de exibição de string
        LEA   DX,MSG            ;mensagem de modificação
        INT   21h              ;exibe no monitor
        POP   DX
        RET                    ;retorno à rotina principal
CONVERTE ENDP
C_SEG   ENDS
        END                    ;forma padrão de terminar neste caso
```

### ❖ Como fazer?

C:\TASM TESTE\_3.ASM (monta-se cada módulo em separado)

C:\TASM TESTE\_4.ASM

C:\TLINK TESTE\_3 + TESTE\_4 (linca-se o conjunto)

- veja páginas 289 a 291 sobre como criar uma *library* de módulos **.OBJ**

C:\TESTE\_3.EXE (executa-se o módulo principal)

- ❖ **EXERCÍCIO 1:** RODE OS ARQUIVOS **TESTE\_1.ASM** E **TESTE\_2.ASM** :
  - OBSERVE QUE AMBOS POSSUEM A MESMA FUNCIONALIDADE.
  - OBSERVE A QUANTIDADE DE BYTES DOS ARQUIVOS EXECUTÁVEIS.
  
- ❖ **EXERCÍCIO 2:** FAZER UM PROGRAMA **MEDE.COM** QUE DÊ O TAMANHO EM BYTES DE UM ARQUIVO QUALQUER.  
**DICA:** MODIFIQUE O PROGRAMA DO EXERCÍCIO 1 DA AULA SOBRE ARQUIVOS (LAB\_LM2) QUE DETERMINA O TAMANHO DE UM ARQUIVO ATRAVÉS DA FUNÇÃO 42h DA INT 21h.
  
- ❖ **EXERCÍCIO 3:** RODE OS ARQUIVOS **TESTE\_3.ASM** E **TESTE\_4.ASM** :
  - OBSERVE O ARQUIVOS **\*.LST** GERADOS APÓS A MONTAGEM. VEJA COMO AS MACROS FORAM EXPANDIDAS.
  - SUPRIMA A INCLUSÃO DA BIBLIOTECA **A:\MY\_LIB.TXT**, RODE NOVAMENTE E VEJA O QUE SE MODIFICA NOS **\*.LST** .
  - OBSERVE O ARQUIVO **TESTE\_3.MAP** , VERIFIQUE COMO OS SEGMENTOS FORAM CRIADOS E SUA OCUPAÇÃO DE ESPAÇO.
  
- ❖ **EXERCÍCIO 4:** MODIFIQUE O PROGRAMA OBTIDO NO EXERCÍCIO 2 DA AULA SOBRE TECLADO (LAB\_LM3, EDITOR DE TEXTO), PARA USAR A DIRETIVA **SEGMENT** E DEFINIR COMPLETAMENTE TODOS OS SEUS SEGMENTOS. IMPLEMENTE A ROTINA PRINCIPAL NUM MÓDULO E A SUBROTINA DE MOVIMENTAÇÃO DO CURSOR NUM SEGUNDO MÓDULO. USE MACROS PARA AS FUNÇÕES DAS TECLAS **↑**, **↓**, **→** E **←**, ALÉM DAS FUNÇÕES DE SCROLL-UP E DOWN.

Observe e comente o conteúdo do arquivo **.MAP**



## ❖ PARA O RELATÓRIO

a) Escreva macros para:

- mover o cursor para a linha L e coluna C  
formato - **MOVE\_CURSOR MACRO L,C .**
- limpar uma janela da tela com canto superior esquerdo (L1,C1), canto inferior direito (L2,C2) e atributo COR  
formato - **LIMPA\_TELA MACRO L1,C1,L2,C2,COR .**
- implementar a nova instrução **PWR N**, que eleva o conteúdo de AX à N-esima potência, deixando o resultado em DX:AX; se o número não couber só em AX, CF/OF devem estar setados.

## ❖ APÊNDICE: Biblioteca de macros.

\*\*\*\*\* Não funcionou: acho que falta formatação para este arquivo, pois resulta em \*\*\*\*\* erro de montagem: o TASM vê um END antes da hora

```
;MY_LIB - BIBLIOTECA PESSOAL DE MACROS
;para acrescentar novas macros, basta apendá-las no final
;procure seguir o padrão de documentação adotado, para facilitar a identificação
;
RETORNO_DOS      MACRO
    MOV  AH,4Ch
    INT  21h
    ENDM
;
NOVA_LINHA      MACRO
;macro para a mudança de linha no monitor
    MOV  AH,02h
    MOV  DL,0Ah
    INT  21h
    MOV  DL,0Dh
    INT  21h
    ENDM
;
EXIBE_STRING    MACRO      STRING
;macro para exibição de string no monitor
;STRING e' uma variável no .DATA
;exemplo de chamada:  EXIBE_STRING 'BOM DIA!'
;
    MOV  AH,09h
    LEA  DX,STRING
    INT  21h
    ENDM
;
EXIBE_MENSAGEM  MACRO      STRING
;macro para exibição de mensagens textuais
;STRING vem juntamente com a chamada
    LOCAL INICIO,MSG
;salva os registradores que serão usados: macro dentro de macro
    SALVA_REGS <AX,DX,DS>
    JMP  INICIO
    MSG    DB    STRING,'$'
    INICIO:
    MOV  AX,CS
    MOV  DS,AX
    MOV  AH,09h
    LEA  DX,MSG
    INT  21h
;restaura os registradores utilizados: macro dentro de macro
    RESTAURA_REGS      MACRO      <DS,DX,AX>
    ENDM
;
SALVA_REGS      MACRO      REGS
;macro para salvar a lista de registradores informada entre <...>
;exemplo de chamada:  SALVA_REGS <AX,CX,DS>
;
```

```

    IRP    D,<REGS>
    PUSH  D
    ENDM
    ENDM
;
RESTAURA_REGS    MACRO    REGS
;macro para restaurar a lista de registradores informada entre <...>
;exemplo de chamada:    SALVA_REGS <AX,CX,DS>
;
    IRP    D,<REGS>
    POP   D
    ENDM
    ENDM
;
LER_TECLADO      MACRO    BUFFER,QUANT_MAX
;macro para uniformizar a digitação feita no teclado
;parâmetros:    BUFFER = array onde armazenar string
;                QUANT_MAX = quant. máxima de caracteres do string
;se os parâmetros BUFFER e QUANT_MAX forem fornecidos:
;                a função 0Ah carrega o string em BUFFER
;                o segundo byte contem a quantidade real de caracteres
;se os parâmetros BUFFER e QUANT_MAX forem omitidos:
;                resulta na leitura de apenas um único caracter em AL
;
    IFNB <BUFFER>
    IFNB <QUANT_MAX>
        MOV  AH,0Ah
        MOV  BUFFER,QUANT_MAX
;o primeiro byte do BUFFER recebe quantidade máxima de bytes fornecida
        LEA  DX,BUFFER
        INT  21H
    ENDIF
    ELSE
        MOV  AH,01h
        INT  21h          ;AL = caracter
    ENDIF
    ENDM
;

```