Faculdade de Computação e Engenharia Elétrica

Microprocessadores e Microcontroladores

Programação Básica

Prof. Dr. Elton Alves

Utilização de Macros

- Os **procedimentos** utilizados na linguagem de programação Assembly 8086/8088 são similares às sub-rotinas usadas em algumas linguagens de alto nível.
- Os **macros** têm similaridades as **funções** em linguagem de alto nível.
- Uma rotina de macro pode ser definida com e sem parâmetros.
- Sintaxe:

nome MACRO [parâmetro1, parâmetro2, ...]

corpo da macro

ENDM

COM e EXE

- Diretivas utilizadas em Assembly
  - **org 100h –** valores numéricos e manipulação de strings.
  - **.MODEL small e .STACK 512d –** operações com valores numéricos.
- Os programas que usam as diretivas **.MODEL small** e **.STACK 512d**, quando compilados, possuem a extensão **.EXE** e os programas que usam a diretiva **org 100h**, quando compilados, possuem a extensão **.COM**.
- Os programas compilados com extensão .EXE são executáveis com estrutura avançada, pois é possível trabalhar com **cabeçalhos**, **realocação de recursos**, entre outras possibilidades.

COM e EXE

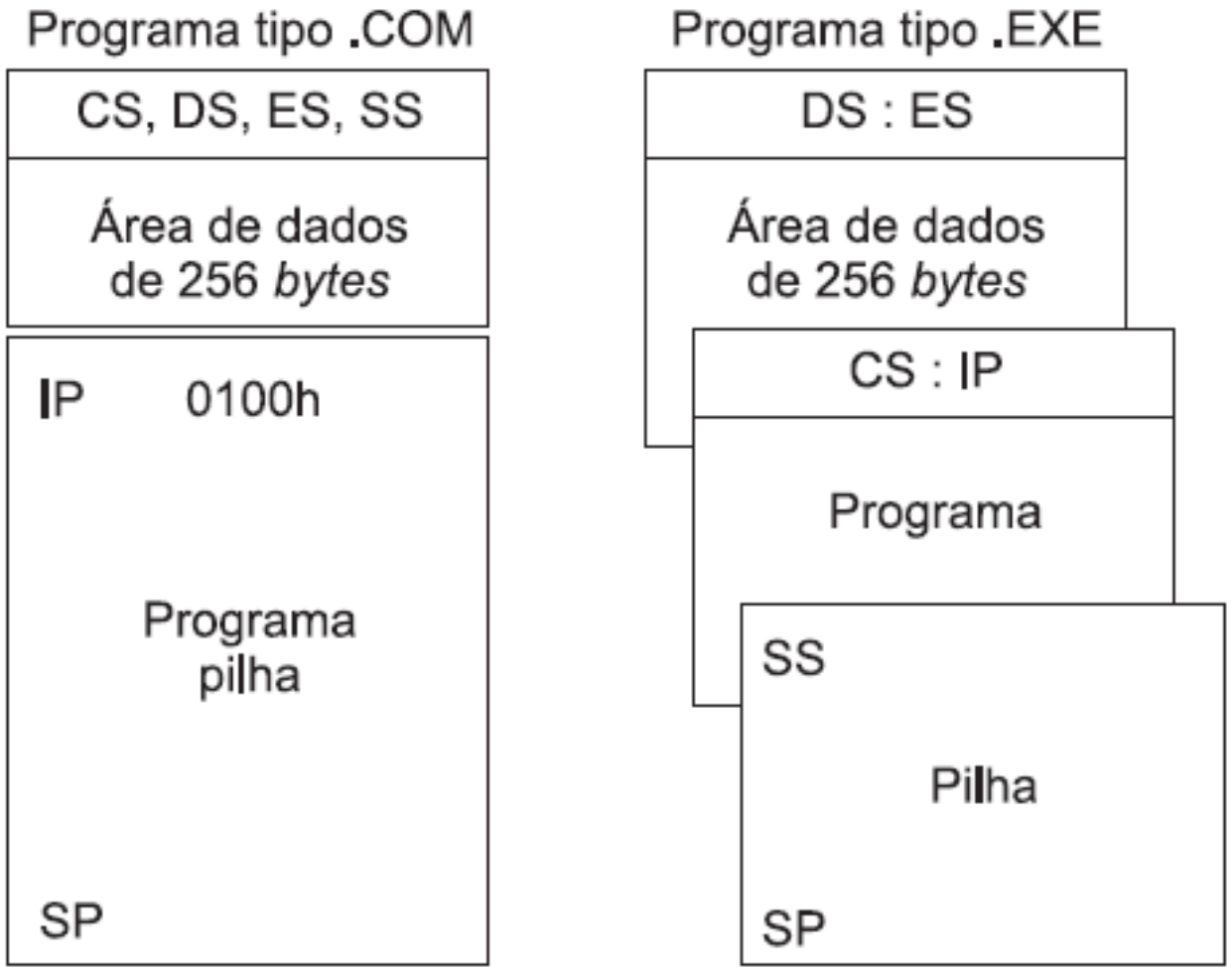


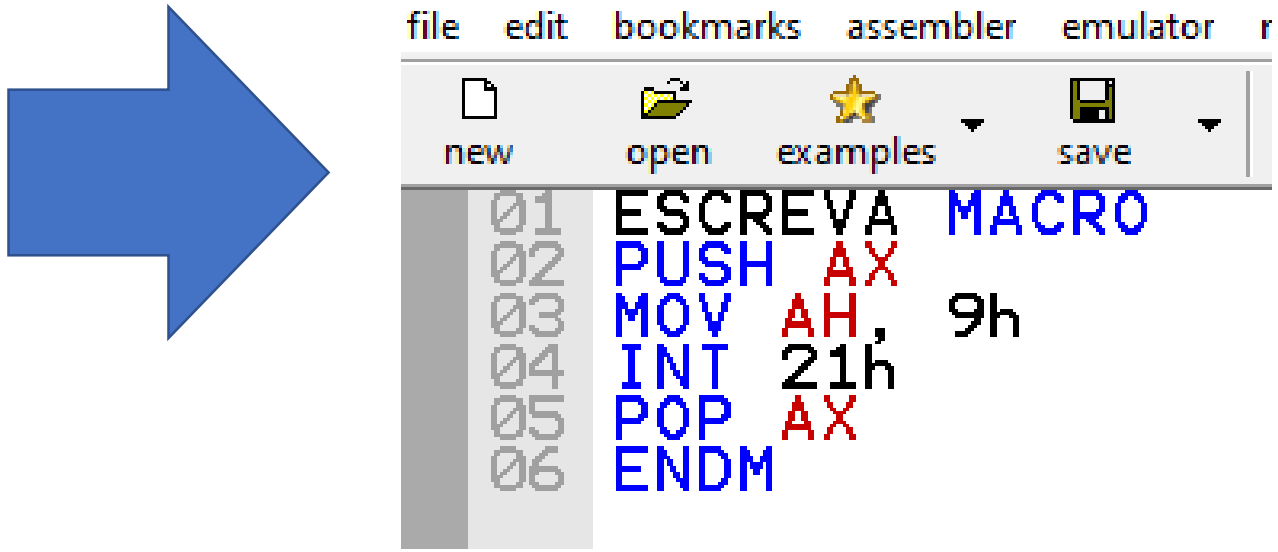
Figura 9.18 - Esboço de memória (adaptado de NORTON & SOSSA, 1988).

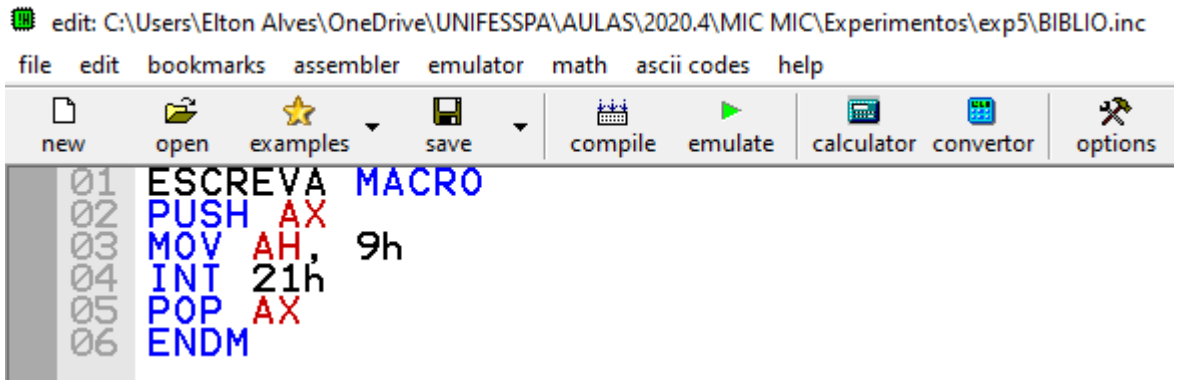
Bibliotecas em Assembly

- Um recurso de programação bastante adequado é o desenvolvimento de uma **biblioteca** de recursos genéricos que possam ser utilizados em vários outros programas.
- Com essa atitude se economiza tempo de desenvolvimento e evita-se a ocorrência de muitos erros de escrita ou de lógica, pois se faz uso da filosofia de **reaproveitamento de código**.

Bibliotecas em Assembly

```
ESCREVA MACRO
PUSH AX
MOV AH, 9h
INT 21h
POP AX
ENDM
```





BIBLIO.inc

Bibliotecas em Assembly

```
01 TITLE Teste de Segmento 5 ; identificacao de nome interno do programa
02
03 #MAKE_EXE# ; tipo de arquivo a ser gerado .EXE
04 ; determina area de dados  - SEGMENT e ENDS com parametro 'DATA'
05
06 INCLUDE 'BIBLIO.inc'
07 DADOS SEGMENT 'DATA'
08 mensagem DB "Ola, Mundo$"
09 DADOS ENDS
10
11 PILHA SEGMENT STACK 'STACK'  ; define o tamanho da pilha
12 DW 0100h DUP(?)
13 PILHA ENDS
14
15 CODIGO SEGMENT 'CODE'  ; identifica o trecho de codigo do programa
16 ASSUME CS:CODIGO, DS:DADOS, SS:PILHA
17 INICIO PROC FAR
18 MOV AX, DADOS
19 MOV DS, AX
20 MOV ES, AX
21 MOV DX, OFFSET mensagem
22 ESCREVA
23 MOV AH, 4Ch
24 INT 21h
25 RET
26 INICIO ENDP
27 CODIGO ENDS
28 END INICIO
```

Bibliotecas em Assembly

Apresentando valores negativos

- Instrução **NEG** (***negate***) – transforma valores negativos em positivos ou vice-versa.
- Quanto aos valores negativos, são calculados e armazenados na memória com o complemento por dois.
- A instrução **NEG** afeta o registrador de estado **SF**.
- Exemplo:

- Subtração de 9-7 = 2 (em memória), **SF = 0** (sem efeito)
- Subtração de 7-9 = Feh (typo byte) ou FFFEh (tipo word), **SF = 1** (valor negativo com base no complemente por dois).

Bibliotecas de funções externas

- A biblioteca padrão do programa denominada **emu8086.inc** é um arquivo de programa em formato texto com a definição **interna de macros** e **procedimentos**.
- Para o efetivo uso da biblioteca dentro de um programa em desenvolvimento, é necessário usar a diretiva **INCLUDE** (como demonstrado anteriormente).

Bibliotecas de funções externas

❑Algumas funções de macro são as seguintes:

- **PUTC caractere** - macro com um parâmetro, que apresenta como saída o caractere informado como parâmetro na posição atual do cursor.
- **PRINT mensagem** - macro com um parâmetro, que apresenta uma mensagem na tela, mantendo o cursor na mesma linha de apresentação.
- **PRINTN mensagem** - macro com um parâmetro, que mostra uma mensagem na tela. Após a apresentação o cursor é posicionado na próxima linha.

Bibliotecas de funções externas

❑Algumas funções de procedimento são as seguintes:

- **SCAN_NUM** - obtém um valor numérico com mais de um dígito, podendo ser positivo ou negativo. Antes da diretiva **END** é necessário utilizar a declaração **DEFINE_SCAN_NUM**.
- **PRINT_STRING** - procedimento usado para imprimir mensagem terminada com caractere nulo na posição atual do cursor. Antes da diretiva **END** é necessário utilizar a declaração **DEFINE_PRINT_STRING**.
- **PRINT_NUM** - apresenta um valor numérico negativo. Antes da diretiva END é necessário utilizar as declarações **DEFINE_PRINT_NUM** e **DEFINE_PRINT_NUM_UNS**.

Tomada de Decisão

- A linguagem de programação Assembly **não** possui de forma explícita, como ocorre nas linguagens de alto nível, instruções de ação direta para tomadas de decisão.
- Para que esse procedimento ocorra, é necessário fazer uso de **duas ou mais** instruções para tal finalidade.
- São usados os **operadores relacionais**, **operadores lógicos**, **decisões simples, decisões compostas e decisões seletivas**.

Tomada de Decisão

Tabela 11.1 - Comparação de uso das relações lógicas

Operador Relacional	Instrução Assembly
=	JE
>	JG
>=	JGE
<	JL
<=	JLE
<>	JNE

Tabela 11.2 - Operador lógico AND

Tabela-Verdade do Operador Lógico de Conjunção - AND		
Condição 1	Condição 2	Resultado Lógico
Verdadeiro	Verdadeiro	Verdadeiro
Verdadeiro	Falso	Falso
Falso	Verdadeiro	Falso
Falso	Falso	Falso

Tabela 11.3 - Operador lógico OR

Tabela-Verdade do Operador Lógico de Disjunção Inclusiva - OR		
Condição 1	Condição 2	Resultado Lógico
Verdadeiro	Verdadeiro	Verdadeiro
Verdadeiro	Falso	Verdadeiro
Falso	Verdadeiro	Verdadeiro
Falso	Falso	Falso

Tabela 11.4 - Operador lógico XOR

Tabela-Verdade do Operador Lógico de Disjunção Exclusiva - XOR		
Condição 1	Condição 2	Resultado Lógico
Verdadeiro	Verdadeiro	Falso
Verdadeiro	Falso	Verdadeiro
Falso	Verdadeiro	Verdadeiro
Falso	Falso	Falso

Tomada de Decisão

Tabela 11.5 - Operador lógico NOT

Tabela-Verdade do Operador Lógico de Negação - NOT	
Condição	Resultado Lógico
Verdadeiro	Falso
Falso	Verdadeiro

Decisão Simples

- Tomada de decisão em alto nível:

```
se ( <elemento 1> <operador relacional> <elemento 2> ) então
  [instruções executadas após condição ser verdadeira]
fim_se
[instruções executadas após condição ser falsa ou ser verdadeira]
```

- Tomada de decisão simples em Assembly:

```
se:
  CMP <elemento 1>, <elemento 2>
  <instrução condicional de salto> fim_se
entao:
  [instruções executadas após condição ser verdadeira]
fim_se:
[instruções executadas após condição ser falsa ou ser verdadeira]
```

Decisão Composta

- Tomada de decisão composta em alto nível

```
se ( <elemento 1> <operador relacional> <elemento 2> ) então
  [instruções executadas após condição ser verdadeira]
senão
  [instruções executadas após condição ser falsa]
fim_se
[instruções executadas após condição ser falsa ou ser verdadeira]
```

- Tomada de decisão composta em Assembly

```
se:
  CMP <elemento 1>, <elemento 2>
  <instrução condicional de salto> senao
entao:
  [instruções executadas após condição ser verdadeira]
  JMP fim_se
senão:
  [instruções executadas após condição ser falsa]
fim_se:
[instruções executadas após condição ser falsa ou ser verdadeira]]
```

Decisões com duas condições

- Tomada de decisão com operador AND em alto nível

```
se (condição1) .e. (condição2) então
  [instruções executadas após condição1 e condição2 serem verdadeiras]
senão
  [instruções executadas caso uma ou ambas as condições sejam falsas]
fim se
```

Decisões com duas condições

- Tomada de decisão com operador AND em Assembly.

se:
CMP <condição1>
<instrução condicional de salto> senao
e:
CMP <condição2>
<instrução condicional de salto> senao
entao:
[instruções executadas após condição1 e condição2 serem verdadeiras]
JMP fim_se
senão:
[instruções executadas caso uma ou ambas as condições sejam falsas]
fim_se:

Decisões com duas condições

- Tomada de decisão composta com operador OR alto nível

```
se (condição1) .ou. (condição2) então
  [instruções executadas quando pelo menos uma das condições for verdadeira]
senão
  [instruções executadas caso ambas as condições sejam falsas]
fim_se
```

- Tomada de decisão composta OR em Assembly

```
  se:
    CMP <condição1>
    <instrução condicional de salto> senao
  ou:

  CMP <condição2>
  <instrução condicional de salto> senao
entao:
  [instruções executadas quando pelo menos uma das condições for verdadeira]
  JMP fim_se
senão:
  [instruções executadas caso ambas as condições sejam falsas]
fim_se:
```