

# **Linguagens de Montagem**

DEMAC – Departamento de Estatística  
Matemática Aplicada e Computação  
UNESP – Rio Claro

**Prof. Daniel Carlos Guimarães Pedronette**




## **Aula 3.**

# **Alocação e Instruções de Transferências de Memória**

# Mais Alocação de Memória

- Definição de constantes:
  - São alocadas de forma contígua:

```
sort      DB      'y'          ; ASCII of y = 79H
value     DW      25159        ; 25159D = 6247H
total     DD      542803535     ; 542803535D = 205A864FH
```

address:	x	x+1	x+2	x+3	x+4	x+5	x+6
contents:	79	47	62	4F	86	5A	20
							
	sort	value		total			

# Mais Alocação de Memória

- Abreviações:

```
message    DB    'W'
           DB    'E'
           DB    'L'
           DB    'C'
           DB    'O'
           DB    'M'
           DB    'E'
           DB    '!'
```

```
message    DB    'W','E','L','C','O','M','E','!'
```

```
message    DB    'WELCOME!'
```

# Mais Alocação de Memória

- Abreviações:

```
message    DB    'B'
           DB    'y'
           DB    'e'
           DB    0DH
           DB    0AH
```

```
message    DB    'Bye' , 0DH, 0AH
```

# Mais Alocação de Memória

- Abreviações:

marks	DW	0
	DW	0
	DW	0
	DW	0
	DW	0
	DW	0
	DW	0
	DW	0

marks	DW	0, 0, 0, 0, 0, 0, 0, 0
-------	----	------------------------

# Mais Alocação de Memória

- Inicializações Múltiplas:
  - Diretiva TIMES permite múltiplas inicializações do mesmo valor:

```
marks      TIMES    8    DW    0
```

- Útil para definir/inicializar arrays

# Tabela de Símbolos

- Alocações são realizadas de maneira contígua:

```
.DATA
value    DW    0
sum      DD    0
marks    TIMES 10 DW 0
message  DB    'The grade is:',0
char1    DB    ?
```

Name	Offset
value	0
sum	2
marks	6
message	26
char1	40



# Mais Alocação de Memória

- Definição de constantes numéricas
  - Diretiva EQU:

```
name      EQU      expression
```

```
NUM_OF_STUDENTS      EQU      90
```

```
      . . .
mov    ECX, NUM_OF_STUDENTS
      . . .
```

# Mais Alocação de Memória

- Definição de constantes numéricas que podem ser redefinidas
  - Diretiva `%assign`:

```
%assign    i    j+1
```

```
%assign    i    j+2
```

# Mais Alocação de Memória

- Definição de constantes numéricas e strings (também suporta redefinição)
  - Diretiva `%define`:

```
%define    X1    [EBP+4]
```

```
%define    X1    [EBP+20]
```

# Transferências de Memória

- Instrução MOV (move)

```
mov    destination, source
```

- Conteúdo inicial de “destination” destruído
- Conteúdo de “source” preservado

```
mov    register, register
mov    register, immediate
mov    memory, immediate
mov    register, memory
mov    memory, register
```

# MOV: Formas de Endereçamento

- Registradores:
  - Tamanho em bits compatível:

```
mov    EAX, EBX
```

```
mov    BX, CX
```

```
mov    AL, CL
```

# MOV: Formas de Endereçamento

- Immediate:
  - Tamanho em bits compatível:

```
mov     AL, 75
```

```
mov     EAX, 45h
```

```
mov     EAX, 0x45
```

# MOV: Formas de Endereçamento

- Endereçamento Indireto de Memória:
  - Sintaxe: [] indicam valor

```
response  DB      'Y'           ; allocates a byte, initializes to Y
table1    TIMES 20 DD 0         ; allocates 80 bytes, initializes to 0
name1     DB      'Jim Ray'     ; 7 bytes are initialized to Jim Ray
```

```
mov  AL, [response]  ; copies Y into AL register
mov  [response], 'N' ; N is written into response
mov  [name1], 'K'     ; write K as the first character of name1
mov  [table1], 56     ; 56 is written in the first element
```

`table1[0] = 56 in C`

# MOV: Formas de Endereçamento

- Endereçamento Direto de Memória:
  - Labels indicam enderços de memória
    - Exemplo:
      - table1 é uma array (definido utilizando TIMES e DD)

`mov EBX, [table1]`      Copia valor da primeira posição de table1 para EBX

`mov EBX, table1`      Copia endereço da primeira posição de table1 para EBX



# MOV: Formas de Endereçamento

- Exemplos:

```

mov    EBX, table1    ; copy address of table1 to EBX
mov    [EBX], 100      ; table1[0] = 100
add    EBX, 4          ; EBX = EBX + 4
mov    [EBX], 99       ; table1[1] = 99
    
```

# MOVes Ambíguos

- Podemos especificar o tipo do operando em algumas situações:

```
mov    EBX,table1
mov    ESI,name1
mov    [EBX],100
mov    [ESI],100
```

```
mov    WORD [EBX],100
mov    BYTE [ESI],100
```

```
mov    [EBX],WORD 100
mov    [ESI],BYTE 100
```

# LEA: Load Effective Address

- Carrega endereço de memória em um registrador:

```
lea    register, source
```

```
lea    EBX, [table1]           mov    EBX, table1
```

- Flexibilidade:

```
lea    EBX, [array+ESI]
```

```
mov    EBX, [array+ESI]      ; illegal
```

# Instrução XCHG

- Como podemos trocar os valores de dois registradores (EAX e EDX, por exemplo)?

```
mov    ECX, EAX
mov    EAX, EDX
mov    EDX, ECX
```

- Instrução eXCHanGe !

# Instrução XCHG

- Exemplos:

```
xchg    EAX, EDX
xchg    [response], CL
xchg    [total], DX
```

```
xchg    [response], [name1]    ; illegal
```

# Transferências de Memória

- Para lembrar:
  - Não há instruções de transferências de memória que operem diretamente com dois endereços de memória.

# Exemplos

Vamos

- Codificar,
- Montar,
- Linkar e
- Testar!

# Exercícios