Linguagens de Montagem

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

Prof. Daniel Carlos Guimarães Pedronette

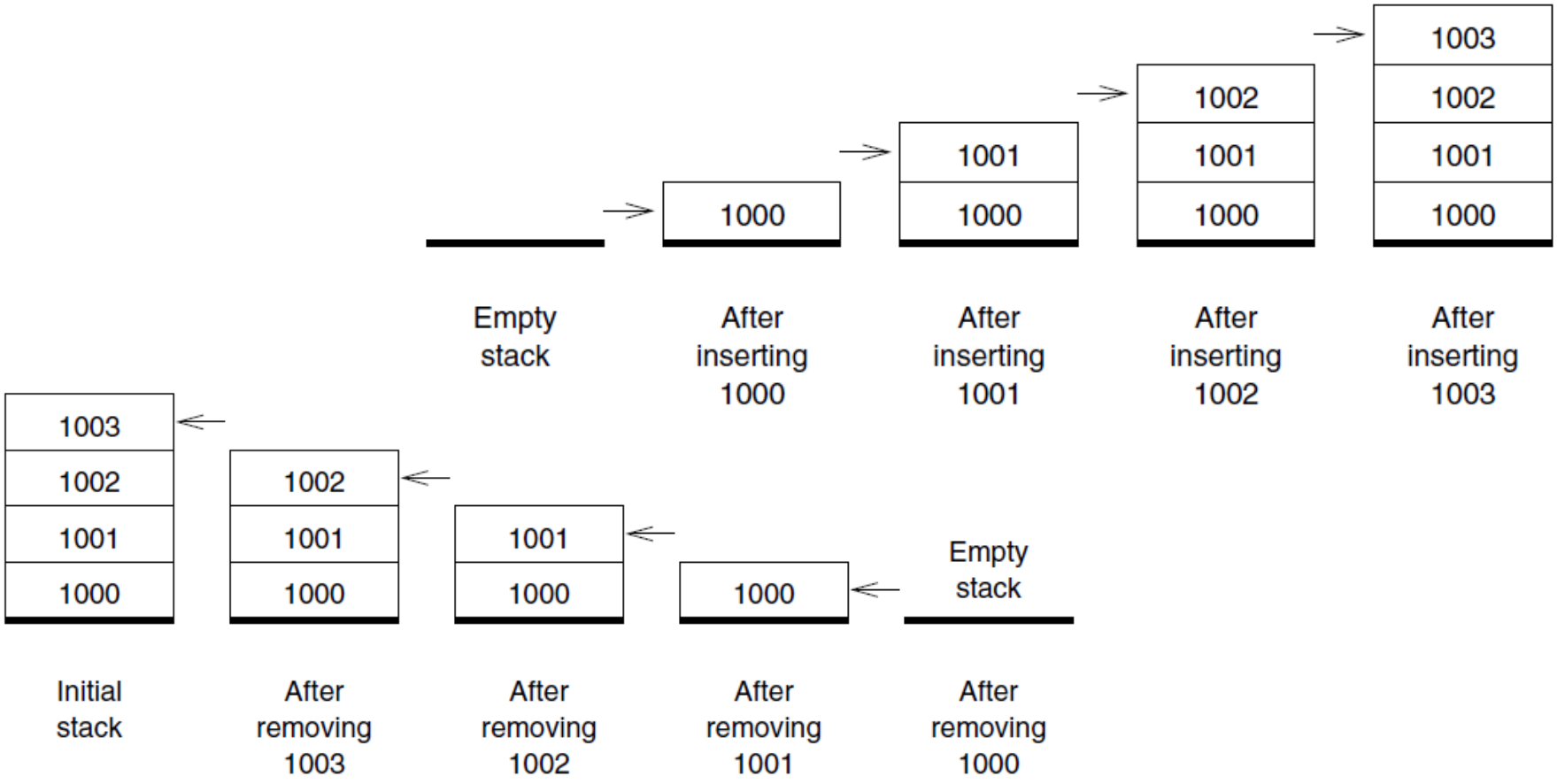
Aula 8.

Pilha e Procedimentos

Pilha (*Stack*)

- Estrutura de dados:
  - LIFO: “Last-In, First Out”
  - Operações associadas: inserção/exclusão
  - Apenas o topo da pilha (TOS – *Top-Of-Stack*) é acessível
  - Insert/Delete: Push/Pop

Pilha


1003
1002
1001
1000
Empty stack
After inserting 1000
After inserting 1001
After inserting 1002
After inserting 1003
1003
1002
1001
1000
Initial stack
After removing 1003
After removing 1002
After removing 1001
After removing 1000
Empty stack

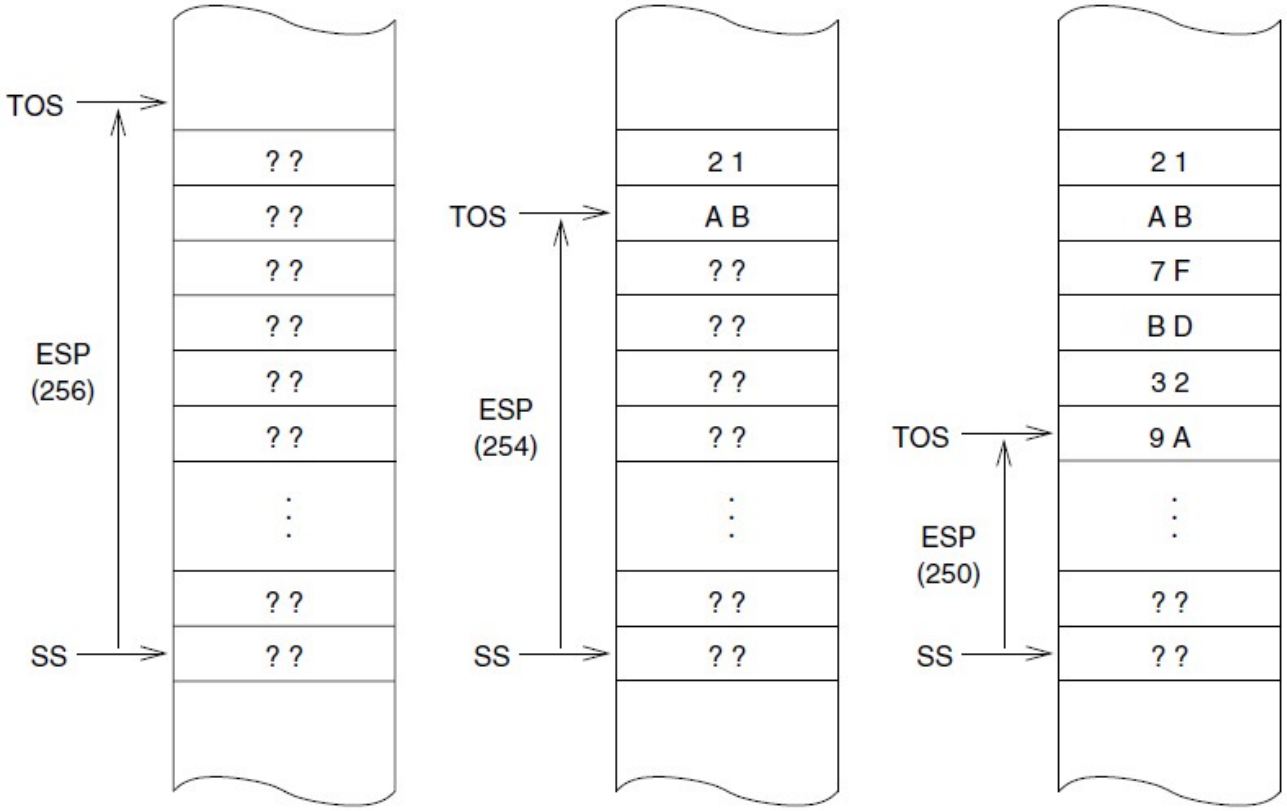
Implementação da Pilha

- Arquitetura Pentium - x86:
  - TOS: indicado por SS:ESP
    - Registrador SS: aponta para o início do segmento de pilha
    - Registrador ESP: deslocamento (offset) para o último item inserido
  - Apenas words (16 bits) ou doublewords (32 bits) são salvas na pilhas, nunca um byte (8 bits)

Implementação da Pilha

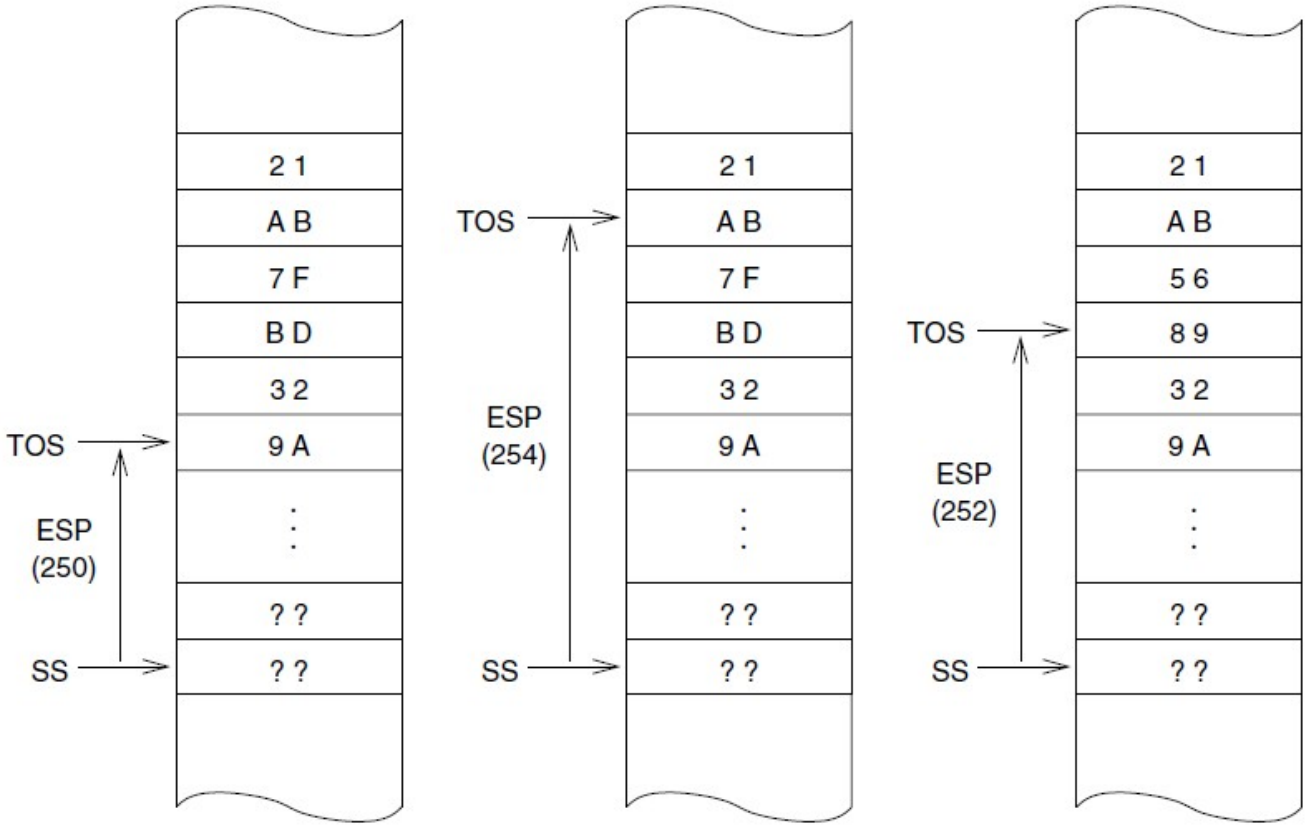
- Arquitetura Pentium – x86: (cont.)
  - Pilha cresce no sentido de menores endereços de memória.
    - Pilha cresce “para baixo”
  - TOS sempre aponta para o byte menos significativo da última “*word*” inserida na pilha

Implementação da Pilha




```
push    21ABH
push    7FBD329AH
```

Implementação da Pilha




```
pop     EBX
```

Implementação da Pilha

`push source16`	ESP = ESP − 2 SS:ESP = `source16`	ESP is first decremented by 2 to modify TOS. Then the 16-bit data from `source16` is copied onto the stack at the new TOS. The stack expands by two bytes.
`push source32`	ESP = ESP − 4 SS:ESP = `source32`	ESP is first decremented by 4 to modify TOS. Then the 32-bit data from `source32` is copied onto the stack at the new TOS. The stack expands by four bytes.
`pop dest16`	`dest16` = SS:ESP ESP = ESP + 2	The data item located at TOS is copied to `dest16`. Then ESP is incremented by 2 to update TOS. The stack shrinks by two bytes.
`pop dest32`	`dest32` = SS:ESP ESP = ESP + 4	The data item located at TOS is copied to `dest32`. Then ESP is incremented by 4 to update TOS. The stack shrinks by four bytes.

Instruções Adicionais - Pilha

```
pusha
popa
```

- Instruções para salvar e recuperar registradores de uso geral:
  - EAX, EBX, ECX, EDX, ESI, EDI
  - Para 16 bits: `pushaw, popaw`

Utilizações da Pilha

- Armazenamento temporário de dados
- Transferência de controle de execução
  - Para chamadas de Procedimentos
- Passagem de parâmetros em chamadas de *procedimentos*

Armazenamento Temporário

- Como trocar o conteúdo de duas posições de memória?

```
xchg    value1,value2        ; illegal


mov     EAX,value1
mov     EBX,value2
mov     value1,EBX
mov     value2,EAX
```

Armazenamento Temporário

- E se for necessário preservar o conteúdo dos registradores EAX, EBX?

```
        . . .
;save EAX and EBX registers on the stack
     push   EAX
     push   EBX
     ;EAX and EBX registers can now be used
     mov    EAX,value1
     mov    EBX,value2
     mov    value1,EBX
     mov    value2,EAX
;restore EAX and EBX registers from the stack
     pop    EBX
     pop    EAX
        . . .
```

```
push    value1
push    value2
pop     value1
pop     value2
```

Utilizações da Pilha

- **Transferência de Controle:**
  - Quando um procedimento é chamado, o endereço de retorno para o programa principal é armazenado na pilha, de forma que o controle possa ser devolvido para o programa principal.
- **Passagem de Parâmetros:**
  - Parâmetros pode ser empilhados antes da chamada de um procedimento.

Procedimentos

- Instruções:

```
call    proc-name

ret
```

`proc-name`: nome do procedimento

`ret`: retorna para o programa que efetuou a chamada

Transferência de Controle

- Registrador EIP
  - Instruction Pointer: “contador de programa”
- Chamadas de procedimento mudam o valor de EIP
  - Instruções CALL e RET

Transferência de Controle

```
 offset     machine code
(in hex)     (in hex)
                          main:
                                  . . .
00000002    E816000000            call    sum
00000007    89C3                  mov     EBX,EAX
                                  . . .
                          ; end of main procedure
;*************************************************
                          sum:
0000001D    55                    push    EBP
                                  . . .
                          ; end of sum procedure
;*************************************************
                          avg:
                                  . . .
00000028    E8F0FFFFFF            call    sum
0000002D    89D8                  mov     EAX,EBX
                                  . . .
                          ; end of avg procedure
.*************************************************
```

```
ESP = ESP – 4                          ; push return address onto the stack
SS:ESP = EIP
EIP = EIP + relative displacement      ; update EIP to point to the procedure
```

Instrução RET

- Transfere o controle para o programa que chamou o procedimento
  - Endereço de retorno é recuperado da pilha

```
EIP = SS:ESP      ; pop return address at TOS into IP
ESP = ESP + 4     ; update TOS by adding 4 to ESP
```

Passagem de Parâmetros

- Por Valor
  - Via Registradores
  - Via Pilha
- Por Referência
  - Via Registradores
  - Via Pilha
- Stack Frame

Passagem de Parâmetros

- **Por Valor:**
  - Via registradores de uso geral
    - Exemplo: CX,DX para os parâmetros e AX para o retorno.

```
20:         call    sum             ; returns sum in AX


31:   sum:
32:         mov     AX,CX           ; sum = first number
33:         add     AX,DX           ; sum = sum + second number
34:         ret
```

Passagem de Parâmetros

- **Por Referência:**
  - Via registradores de uso geral
  - Procedimento str_len
    - Recebe um ponteiro para uma string em EBX
    - Retorna o comprimento da string em AX

Passagem de Parâmetros

```
string:    db      'String de Exemplo'

mov      EBX,string      ; EBX = string address
call     str_len         ; returns string length in AX
```

```
;-----------------------------------------------------------
;Procedure str_len receives a pointer to a string in EBX.
;String length is returned in AX.
;-----------------------------------------------------------
str_len:
      push    EBX
      sub     AX,AX           ; string length = 0
repeat:
      cmp     byte [EBX],0    ; compare with NULL char.
      je      str_len_done    ; if NULL we are done
      inc     AX              ; else, increment string length
      inc     EBX             ; point EBX to the next char.
      jmp     repeat          ;  and repeat the process

str_len_done:
       pop     EBX
       ret
```

Passagem de Parâmetros

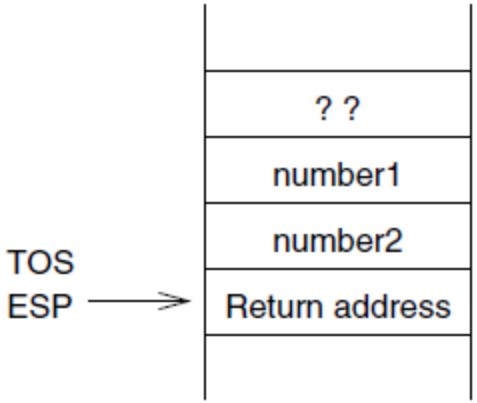
- **Utilizando Pilha:**

```
push    number1
push    number2
call    sum
```

  - Lendo os parâmetros (dentro do procedimento)

```
pop     EAX
pop     EBX
pop     ECX


push    EAX
```



Passagem de Parâmetros

- **Utilizando Pilha:**
  - Podemos deixar os parâmetros na pilha e acessá-los conforme necessário:

```
mov EBX,[ESP+4]     ;number2
mov ECX,[ESP+6]     ;number1
```

Stack Frame

- A **Stack Frame**, ou *registro de ativação*, consiste na coleção de dados associados a uma chamada de procedimento.
  - Parâmetros
  - Endereço de retorno
  - Cópias de registradores salvos
  - Variáveis locais (opcional)
- Dados são salvos na pilha
  - Inclusive parâmetros

Stack Frame

- Padrões para Stack Frame (1):
  - Podemos deixar os parâmetros na pilhas e acessá-lo por meio de:

    `[ESP+x]`
  - Mas se o procedimento usar a pilha, ESP é alterado e o valor do parâmetro é perdido.
  - **Solução:**
    - Copiar ESP em EBP ao iniciar um procedimento

Stack Frame

- Padrões para Stack Frame (2):
  - EBP é utilizado para salvar valor de ESP
  - Mas e se um procedimento fizer uma chamada a outro procedimento?
    - O valor original de EBP pode ser perdido.
  - **Solução:**
    - Antes de salvar ESP em EBP, salvar EBP na pilha

Stack Frame

- Exemplo:

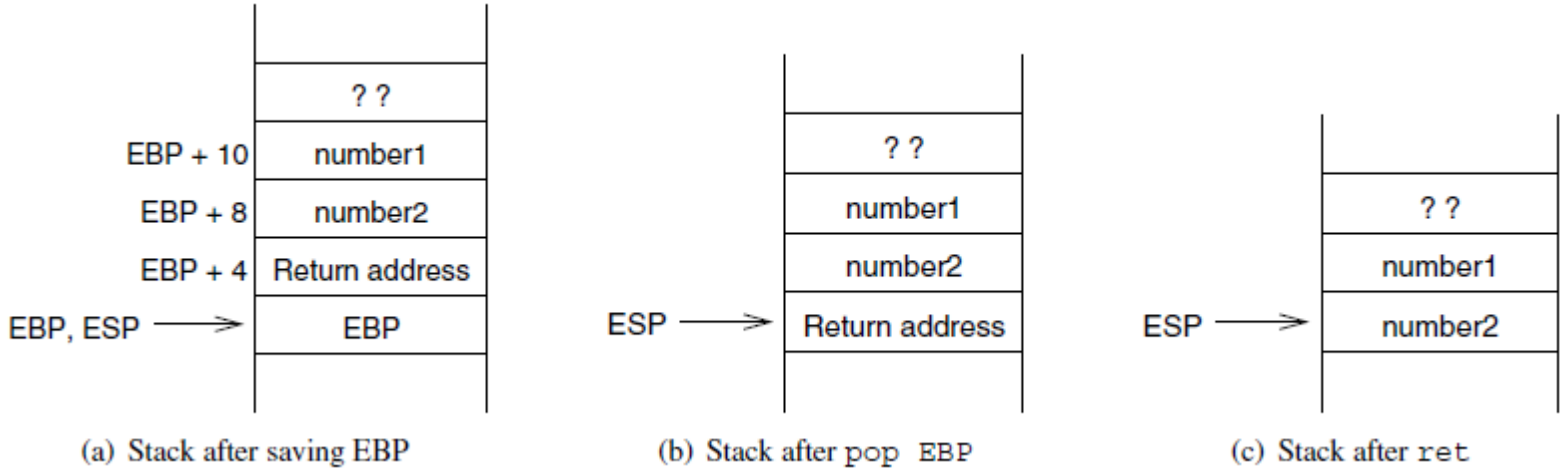
```
PROC:
        push    EBP
        mov     EBP,ESP

          ...

        mov     ESP,EBP     → Restaura estado da Pilha
        pop     EBP         → Restaura estado do EBP
        ret
```

Passagem de Parâmetros

- **Utilizando Pilha: Stack Frame**
  - Utilizando EBP:



(a) Stack after saving EBP (b) Stack after `pop EBP` (c) Stack after `ret`

Stack Frame

- Para compiladores C:

```
push    number1
push    number2
call    sum
add     ESP,4
```

Empilhar parâmetros number1, number2

Valor de retorno: EAX

Liberar espaço ocupado por number1, number2

```
sum(number2,number1);
```

  - gcc, por exemplo retorna inteiros em EAX

Stack Frame: Instrução RET

- Ao invés de deixar para que o procedimento que realizou a chamada "limpe" a pilha, a instrução RET pode fazê-lo:

```
ret     optional-value
```

```
EIP = SS:ESP
ESP = ESP + 4 + optional-value
```

Stack Frame

- **Stack Frame** (*parâmetros por referência*):
  - "Empilhar endereços"
  - Exemplo:
    - swap (troca) caracteres
    - Chamada:

```
mov     EAX,string      ; EAX = string[0] pointer
push    EAX
inc     EAX             ; EAX = string[1] pointer
push    EAX
call    swap            ; swaps the first two characters
```

Stack Frame

- **Stack Frame** (*parâmetros por referência*):

```
38:   .CODE
39:   swap:
40:         enter   0,0
41:         push    EBX             ; save EBX - procedure uses EBX
42:         ; swap begins here. Because of xchg, AL is preserved.
43:         mov     EBX,[EBP+12]    ; EBX = first character pointer
44:         xchg    AL,[EBX]
45:         mov     EBX,[EBP+8]     ; EBX = second character pointer
46:         xchg    AL,[EBX]
47:         mov     EBX,[EBP+12]    ; EBX = first character pointer
48:         xchg    AL,[EBX]
49:         ; swap ends here
50:         pop     EBX             ; restore registers
51:         leave
52:         ret     8               ; return and clear parameters
```

Instruções ENTER e LEAVE

- ENTER:

```
enter     XX,0
```

is equivalent to

```
push      EBP
mov       EBP,ESP
sub       ESP,XX
```

XX: Número em bytes para a armazenamento de variáveis locais

Instruções ENTER e LEAVE

- LEAVE:

```
mov     ESP,EBP
pop     EBP
```

Stack Frame

```
38:   .CODE
39:   swap:
40:         enter    0,0
41:         push     EBX              ; save EBX - procedure uses EBX
42:         ; swap begins here. Because of xchg, AL is preserved.
43:         mov      EBX,[EBP+12]     ; EBX = first character pointer
44:         xchg     AL,[EBX]
45:         mov      EBX,[EBP+8]      ; EBX = second character pointer
46:         xchg     AL,[EBX]
47:         mov      EBX,[EBP+12]     ; EBX = first character pointer
48:         xchg     AL,[EBX]
49:         ; swap ends here
50:         pop      EBX              ; restore registers
51:         leave
52:         ret      8                ; return and clear parameters
```

```
push    EBP
mov     EBP,ESP
```

```
mov     ESP,EBP
pop     EBP
```

Dúvidas?

Exemplos

Prática:

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