

# ORGANIZAÇÃO BÁSICA DE COMPUTADORES E LINGUAGEM DE MONTAGEM

---

## Pilha

### Organização da Pilha (*stack*)

- estrutura de dados de uma dimensão organizada em algum trecho (segmento) da Memória;
  - o primeiro item adicionado é o último a ser removido (*first-in, last-out*);
  - a posição da pilha mais recentemente acrescida é o topo da pilha.
- Declaração do segmento de pilha  
    `.STACK 100h`                      ;dimensiona o tamanho da pilha
    - SS -> aponta o início do segmento de pilha (base)
    - SP -> aponta o topo da pilha (define o deslocamento do topo em relação à base)
  - A pilha cresce do topo para baixo (endereço maior para o menor).
    - Endereço para acesso à pilha: SS:SP (no. de segmento:offset)

# ORGANIZAÇÃO BÁSICA DE COMPUTADORES E LINGUAGEM DE MONTAGEM

## Organização da Pilha (*stack*)

SP	PILHA	
	Endereço	Conteúdo
	00FAh	
	00FBh	
	00FCh	78
	00FDh	56
	00FEh	34
	00FFh	12
→	0100h	

PUSH AX  
AX = 1234h  
PUSH CX  
CX = 5678h

POP BX  
BX ← 5678h  
POP CX  
CX ← 1234h

# ORGANIZAÇÃO BÁSICA DE COMPUTADORES E LINGUAGEM DE MONTAGEM

---

## Instruções para armazenar dados na pilha

**PUSH fonte**  
**PUSHF**

- Onde fonte é:
  - um registrador de 16 bits;
  - uma palavra de memória ou variável de 16 bits (de tipo DW).
- A execução de **PUSH** resulta nas seguintes ações:
  - o registrador **SP** (*stack pointer*) é decrementado de 2;
  - uma cópia do conteúdo da fonte é armazenado na pilha de forma que:
    - posição **SS:SP** → armazena o byte baixo da fonte;
    - a posição **SS:(SP + 1)** → armazena o byte alto;
    - o conteúdo da fonte não é alterado.
- A execução de **PUSHF**, que não possui operando, resulta:
  - o registrador **SP** (*stack pointer*) é decrementado de 2
  - uma cópia do conteúdo do registrador **FLAG** é armazenado na pilha
- Observações:
  - As instruções de pilha não alteram os **FLAGS**;
  - Não é possível movimentar dados de 8 bits, nem valores imediatos.

# ORGANIZAÇÃO BÁSICA DE COMPUTADORES E LINGUAGEM DE MONTAGEM

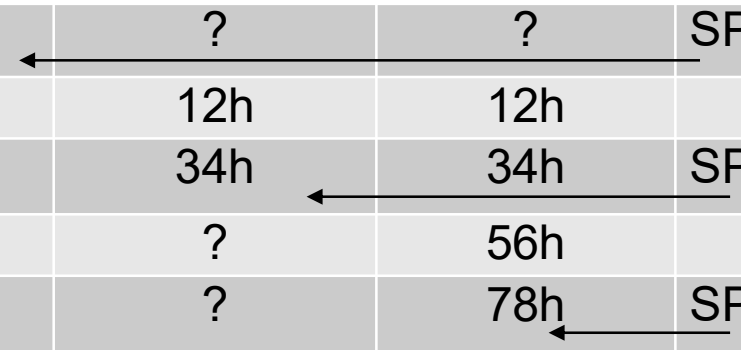
## Instruções para armazenar dados na pilha

Exemplo de operação: ...

**PUSH AX** ;instrução 1 AX = 1234h

**PUSHF** ;instrução 2 FLAGS = 5678h

Offset	Antes	Depois de 1	Depois de 2	
0100h	?	?	?	SP <sub>1</sub>
00FFh	?	12h	12h	
00FEh	?	34h	34h	SP <sub>2</sub>
00FDh	?	?	56h	
00FCh	?	?	78h	SP <sub>3</sub>
00FBh	?	?	?	
00FAh	?	?	?	
00F9h	?	?	?	



# ORGANIZAÇÃO BÁSICA DE COMPUTADORES E LINGUAGEM DE MONTAGEM

---

Instruções para retirar dados na pilha

**POP destino**

**POPF**

- Onde destino é:
  - um registrador de 16 bits;
  - uma palavra de memória ou variável de 16 bits (de tipo DW).
- A execução de POP resulta nas seguintes ações:
  - o conteúdo das posições SS:SP (byte baixo) e SS:(SP + 1) (byte alto) é movido para o destino;
  - o registrador SP (*stack pointer*) é incrementado de 2.
- A execução de POPF , que não possui operando, resulta:
  - o conteúdo das posições SS:SP (byte baixo) e SS:(SP + 1) (byte alto) é movido para o registrador de FLAGS;
  - o registrador SP (*stack pointer*) é decrementado de 2.

# ORGANIZAÇÃO BÁSICA DE COMPUTADORES E LINGUAGEM DE MONTAGEM

## Instruções para retirar dados na pilha

Exemplo de operação:

**POPF**                    ;instrução 1  
**POP AX**                ;instrução 2

Offset	Antes	Depois de 1	Depois de 2	
0100h	?	?	?	SP <sub>2</sub>
00FFh	12h	12h	12h	
00FEh	34h	34h	34h	SP <sub>1</sub>
00FDh	56h	56h	56h	
00FCh	78h	78h	78h	SP <sub>0</sub>
00FBh	?	?	?	
00FAh	?	?	?	
00F9h	?	?	?	

# ORGANIZAÇÃO BÁSICA DE COMPUTADORES E LINGUAGEM DE MONTAGEM

---

**Exemplo de uso de pilha - ENTRADA DE CARACTERES COM SAÍDA INVERTIDA, USANDO PILHA**

- Imprimir uma “?”
- Ler n caracteres e imprimir estes caracteres na ordem inversa à lida.
- Critério de parada de leitura é o ENTER (0Dh)

# ORGANIZAÇÃO BÁSICA DE COMPUTADORES E LINGUAGEM DE MONTAGEM

## Exemplo de uso de pilha

```
TITLE ENTRADA DE CARACTERE INVERTIDA USANDO PILHA
.MODEL SMALL
.STACK 100h
.CODE
MAIN PROC
    MOV AH,2      ;exibe o Prompt para o usuario
    MOV DL,'?'    ;caractere '?' para a tela
    INT 21h       ;exibe
    XOR CX,CX     ;inicializando contador caracteres em zero
    MOV AH,1      ;prepara para ler um caractere do teclado
    INT 21h       ;caractere em AL

;while caractere nao e <CR> do

INICIO:
    CMP AL,0DH    ;e' o caractere <CR>?
    JE PT1        ;sim, entao saindo do loop

;salvando o caractere na pilha e incrementando o contador

    PUSH AX       ;AX vai para a pilha (interessa somente
AL)
    INC CX        ;contador = contador + 1
;lendo um novo caractere
    INT 21h       ;novo caractere em AL
    JMP INICIO    ;retorna para o inicio do loop
;end_while
```

```
PT1: MOV AH,2      ;prepara para exibir
      MOV DL,0DH   ;<CR>
      INT 21h      ;exibindo
      MOV DL,0AH   ;<LF>
      INT 21h      ;exibindo: mudança linha
      JCXZ FIM     ;saindo se nenhum caractere
foi digitado

;for contador vezes do

TOPO:
    POP DX         ;retira primeiro caractere da
pilha
    INT 21h        ;exibindo caractere
    LOOP TOPO      ;em loop até CX = 0
;end_for

FIM:
    MOV AH,4CH     ; saída para para o SO
    INT 21H
MAIN ENDP
END MAIN
```



# ORGANIZAÇÃO BÁSICA DE COMPUTADORES E LINGUAGEM DE MONTAGEM

---

## Sub-rotinas ou Procedimentos

- **Sintaxe para sub-rotinas:**

**nome      PROC    tipo**

**;**

**;corpo da sub-rotina - instruções**

**;**

**RET            ;transfere o controle de volta para a rotina  
                 ;principal**

**nome      ENDP**

- **Tipos possíveis**    {    **NEAR : sub-rotina no mesmo segmento de código**  
                              **FAR : em outro segmento de código**

# ORGANIZAÇÃO BÁSICA DE COMPUTADORES E LINGUAGEM DE MONTAGEM

## Sub-rotinas ou Procedimentos

- Mecanismo de chamada e retorno:

```
MAIN PROC
...
CALL SUB1
;próxima instrução
...
MAIN ENDP
```

```
SUB1 PROC
;primeira instrução
...
RET
SUB1 ENDP
```

```
CALL
SP ← SP - 2
MEM[SP] ← IP
IP ← SUB1
```

```
RET
IP ← MEM[SP]
SP ← SP + 2
```

- Comunicação de dados entre sub-rotinas:
  - Em Linguagem de Montagem, não há lista de parâmetros;
  - Se há poucos valores de entrada e saída → usar registradores

# ORGANIZAÇÃO BÁSICA DE COMPUTADORES E LINGUAGEM DE MONTAGEM

---

## Chamada e retorno de sub-rotina

- Instrução de chamada de procedimento:  
**CALL nome**
  - IP, que contem o *offset* do endereço da próxima instrução da rotina que chama (após a instrução **CALL**), é armazenado na pilha;
  - IP recebe o *offset* do endereço da primeira instrução da sub-rotina chamada.
- Instrução de retorno do procedimento:  
**RET**
  - Faz com que o *offset* do endereço da próxima instrução da rotina que chama, que está na pilha, seja recarregado em IP.
- Ambas instruções **CALL** e **RET** não afetam **FLAGS**.

\_\_\_\_\_

SEGMENTO DE CÓDIGO		SEGMENTO DE PILHA		
Offset no CS	Código	Offset no SS	Antes da chamada	Depois da chamada
	main PROC	0100h	?	?
	.....	00FFh	?	12h
1010h	CALL sub1	00FEh	?	10h
1012h	próxima instr.	00FDh	?	?
	.....	00FCh	?	?
	sub1 PROC	00FBh	?	?
1200h	primeira instr.	00FAh	?	?
	.....			
	.....		IP	
1300h	RET		Antes da chamada	Depois da chamada
			1012h	1200h

# ORGANIZAÇÃO BÁSICA DE COMPUTADORES E LINGUAGEM DE MONTAGEM

Mecanismo de retorno:

SEGMENTO DE CÓDIGO		SEGMENTO DE PILHA			
Offset no CS	Código	Offset no SS	Antes da chamada	Depois da chamada	
	main PROC	0100h	?	← ?	SP <sub>2</sub>
	.....	00FFh	12h	12h	
1010h	CALL sub1	00FEh	10h	← 10h	SP <sub>1</sub>
1012h	próxima instr.	00FDh	?	?	
	.....	00FCh	?	?	
	sub1 PROC	00FBh	?	?	
1200h	primeira instr.	00FAh	?	?	
	.....				
	.....		IP		
1300h	RET		Antes da chamada	Depois da chamada	
			1300h	1012h	

## **ORGANIZAÇÃO BÁSICA DE COMPUTADORES E LINGUAGEM DE MONTAGEM - um exemplo de uso sub-rotina:**

---

- **Um exemplo de uso sub-rotina - MULTIPLICAÇÃO POR SOMA E DESLOCAMENTO**
  - dois números 2 e o 3, já vão estar na memória
  - 2 procedimentos, um para multiplicar e outro para imprimir o resultado

# ORGANIZAÇÃO BÁSICA DE COMPUTADORES E LINGUAGEM DE MONTAGEM - um exemplo de uso sub-rotina:

```

TITLE  MULTIPLICACAO POR SOMA E
        DESLOCAMENTO
.MODEL SMALL
.STACK 100h
.DATA
    NUM1 DB 2
    NUM2 DB 3
.CODE
MAIN  PROC
    MOV AX,@DATA
    MOV DS,AX
    MOV AL,NUM1
    MOV BL,NUM2
    CALL MULTIPLICA
    CALL IMPRIME
    MOV AH,4Ch
    INT 21h
    MAIN  ENDP
MULTIPLICA  PROC
;multiplica dois numeros
;A e B por soma e deslocamento
;entradas:  AX = A, BX = B,
; numeros na faixa 00h - FFh
;saida:     DX = A*B (produto)
    PUSH AX
    PUSH BX ;salva os conteudos de AX e BX
    AND DX,0 ;inicializa DX em 0
;repeat if B e' impar

```

```

TOPO:
    TEST BX,1 ;LSB de BX = 1?
    JZ PT1    ;nao, (LSB = 0)
;then
    ADD DX,AX ;sim, entao
    ;produto = produto + A
;end_if
PT1:
    SHL AX,1  ;desloca A para a esquerda 1 bit
    SHR BX,1  ;desloca B para a direita 1 bit
;until
    JNZ TOPO  ;fecha o loop repeat
    POP BX
    POP AX    ;restaura os conteudos de BX e AX
    RET      ;retorno para o ponto de chamada
MULTIPLICA  ENDP
IMPRIME PROC
; imprime um numero decimal de 1 digito
; entrada DL
; saída nao ha
    PUSH AX  ; SALVA AX E DX NA PILHA
    PUSH DX
    MOV AH,02H
    OR DL,30h
    INT 21h  ;IMPRIME CARACTER NUMERICO
    POP DX   ; RESTAURA DX E AX
    POP AX
    RET      ;retorno para o ponto de chamada
IMPRIME ENDP
END MAIN

```

---

**0010**

**0011**

**0001**

**0000**

**0000**

**0010**

**0100**

**0110**