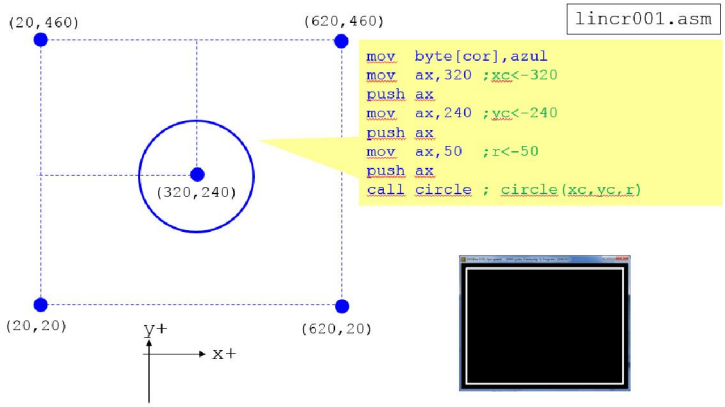
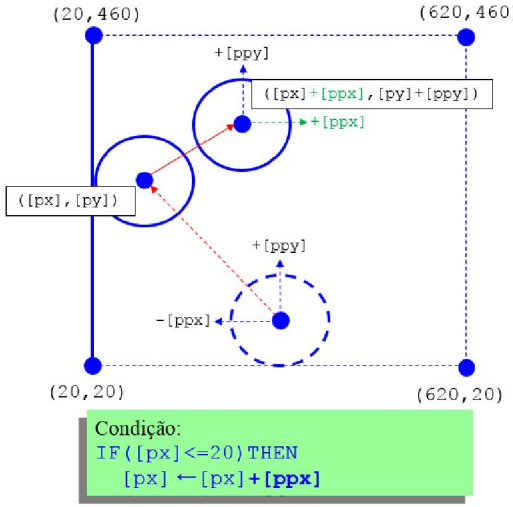
Procedimentos e passagem de parâmetros pela pilha

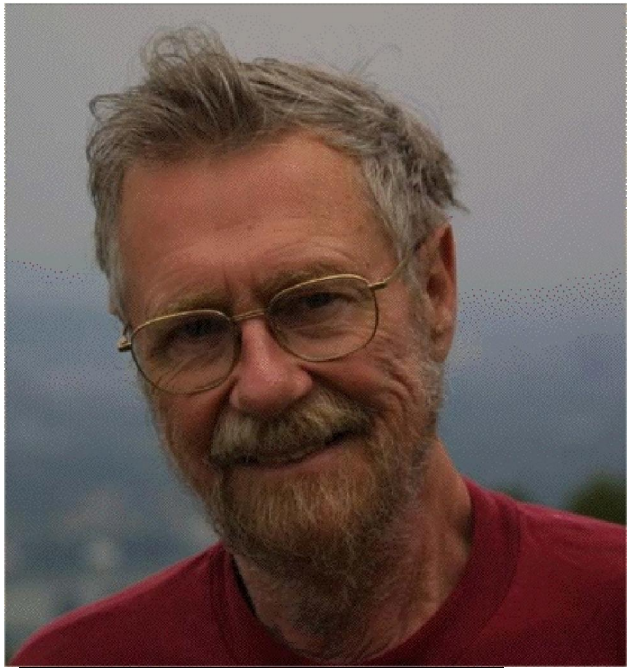
Dr. Jorge L. Aching Samatelo
jlasam001@gmail.com





"A arte de programar consiste na arte de organizar e dominar a complexidade dos sistemas"

Dijkstra



Edsger Dijkstra

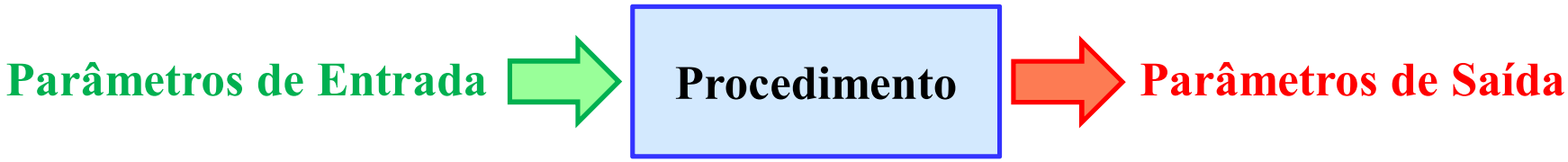
Índice

Conceitos Básicos de Programação

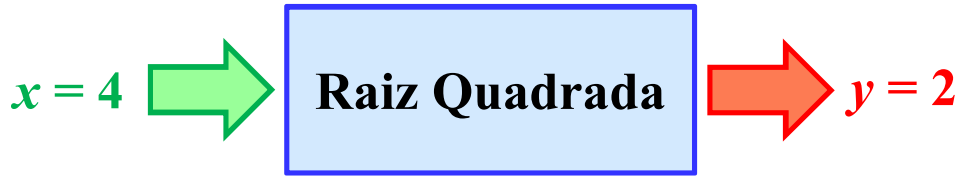
Conceitos Básicos de Programação

Procedimento

- Um **procedimento** é um **sub-rotina** que **que efetua uma tarefa especifica**. Pudendo precisar de parâmetros de entrada como de saída.



- *Por exemplo*
  - Procedimento para calcular a raiz quadrada de um número



$$y = \sqrt{x} = \sqrt{\lambda + 1} = 1 + \frac{1}{2}\lambda - \frac{1}{2}\lambda^2 + \frac{1}{16}\lambda^3 - \cdots$$

Algoritmo

Conceitos Básicos de Programação

Procedimento

- *Os procedimentos são implementados como*
  - Trechos de código independentes, executados somente quando chamados por outro(s) trecho(s) de código.

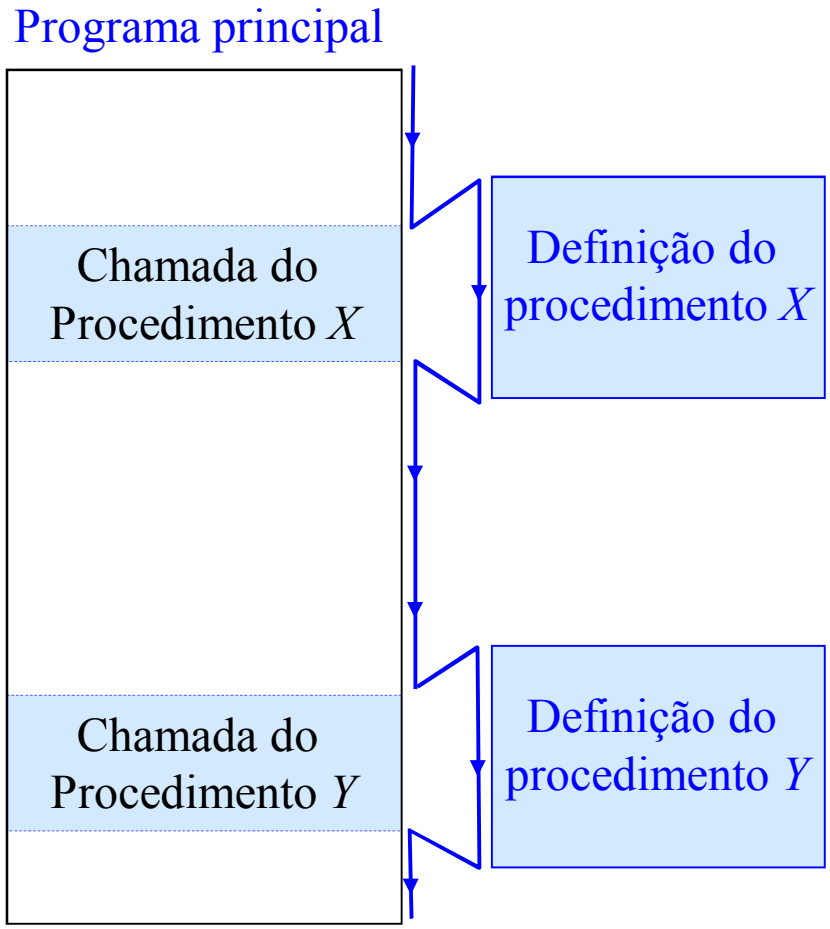
- Para poder implementar um procedimento temos que trabalhar em dois passos.
  - Definição do procedimento. É o trecho de código que implementa a tarefa efetuada pelo procedimento.
  - Chamada do procedimento. É a linha de código no programa principal que chama ao procedimento para a execução da tarefa especifica.

- Ao ser chamado o procedimento, o fluxo de execução desloca-se do fluxo principal para a executar o procedimento.
- Concluída a execução do procedimento, o fluxo de execução retorna ao ponto imediatamente após onde ocorreu a chamada no programa principal.

Conceitos Básicos de Programação

Procedimento

- Então a logica de execução de um procedimento é:
  - Ao ser chamado o procedimento, o fluxo de execução desloca-se do fluxo principal para o trecho de código onde o procedimento é definido, executando o procedimento.
  - Concluída a execução do procedimento, o fluxo de execução retorna ao ponto imediatamente após onde ocorreu a chamada do procedimento no programa principal.



Fluxo de execução

Procedimentos (*Procedures*)

Procedimentos (*Procedures*)

Instrução de chamada

Instrução **CALL**

- **Que faz:**
  - Utilizada para CHAMAR um procedimento.
  - Faz um desvio no código e guarda o endereço da próxima instrução na pilha.

- **Sintaxe:**

```
CALL <dir. destino>
```

  - Apos a execução da instrução `CALL` resulta nas seguintes ações:
    1. O registrador `IP`, que contem o deslocamento (*offset*) do endereço da próxima instrução da rotina "chamante", é armazenado na pilha;
    2. O registrador `IP` recebe o deslocamento (*offset*) do endereço da primeira instrução do procedimento chamado.

Procedimentos (*Procedures*)

Instrução de retorno

Instrução **RET**

- **Que faz:**
  - Utilizada ao final de um procedimento chamado por uma instrução `CALL`.
  - Retira o endereço que está no topo da pilha e atualiza o registrador `IP` com este.

- **Sintaxe:**

```
RET
```

Procedimentos (*Procedures*)

Instrução de retorno

Instrução **RET**

- **Observação:**
  - Se a instrução RET tem um numero inteiro como operando

```
RET <NUMBER>
```

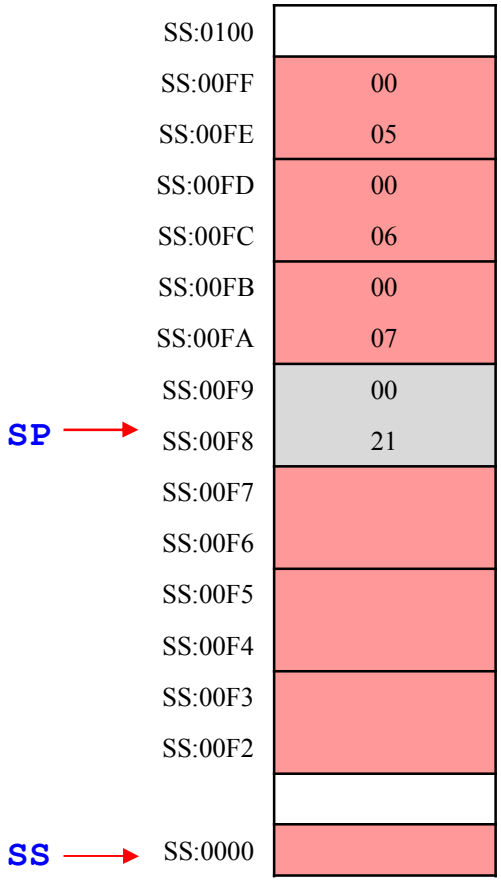
  - Então, a execução da instrução `RET` resulta nas seguintes ações:
    - Primeiro, retira o endereço que está no topo da pilha e atualiza o registrador `IP` com este valor.
    - Segundo, o registrador `SP` (***S**tack **P**ointer*) é INCREMENTADO em duas unidades + `NUMBER`.

Procedimentos (*Procedures*)

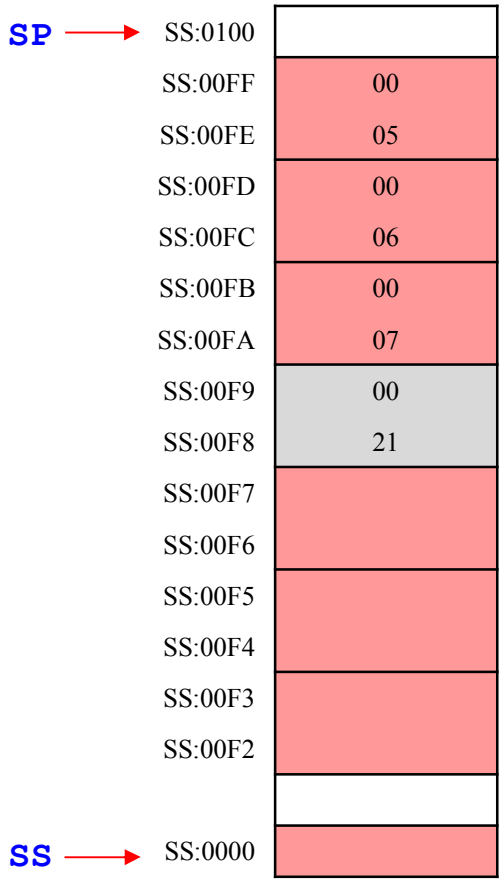
Instrução de retorno

Instrução **RET**

Antes de executar `ret 6`



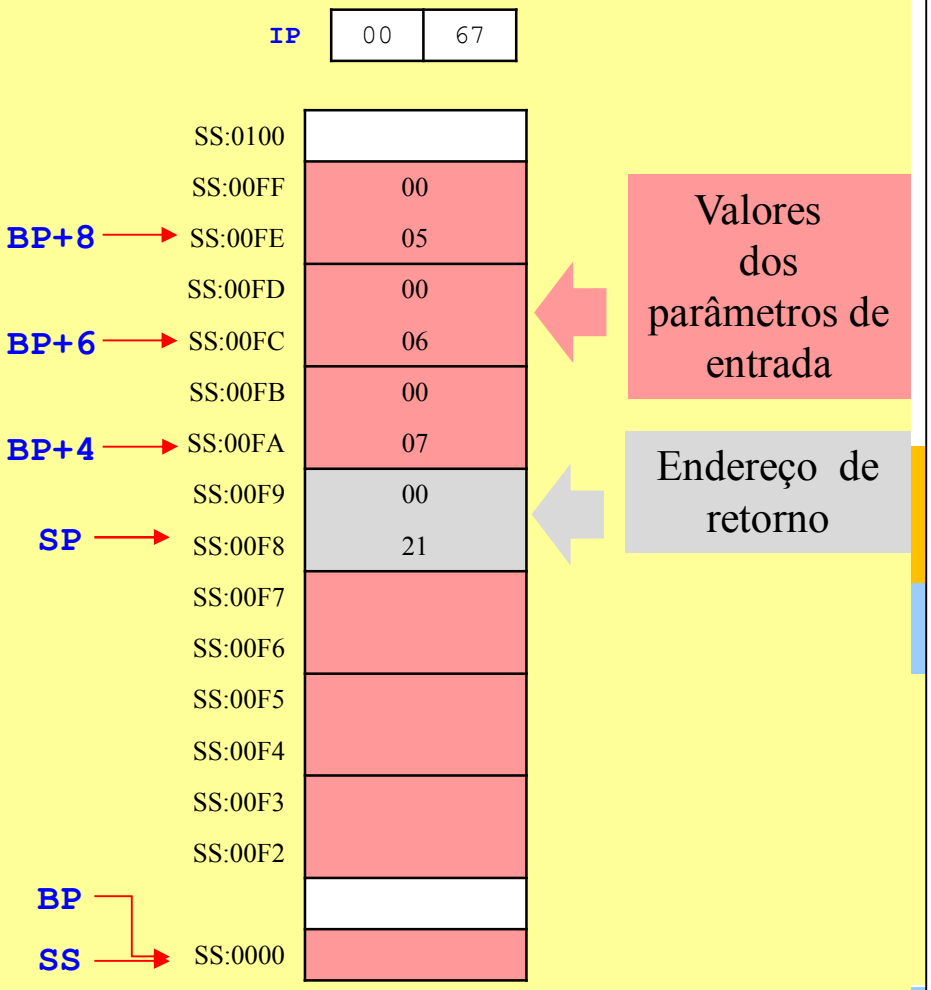
Depois de executar `ret 6`



Exemplo: Passagem de parâmetros pela pilha

exsb003.asm

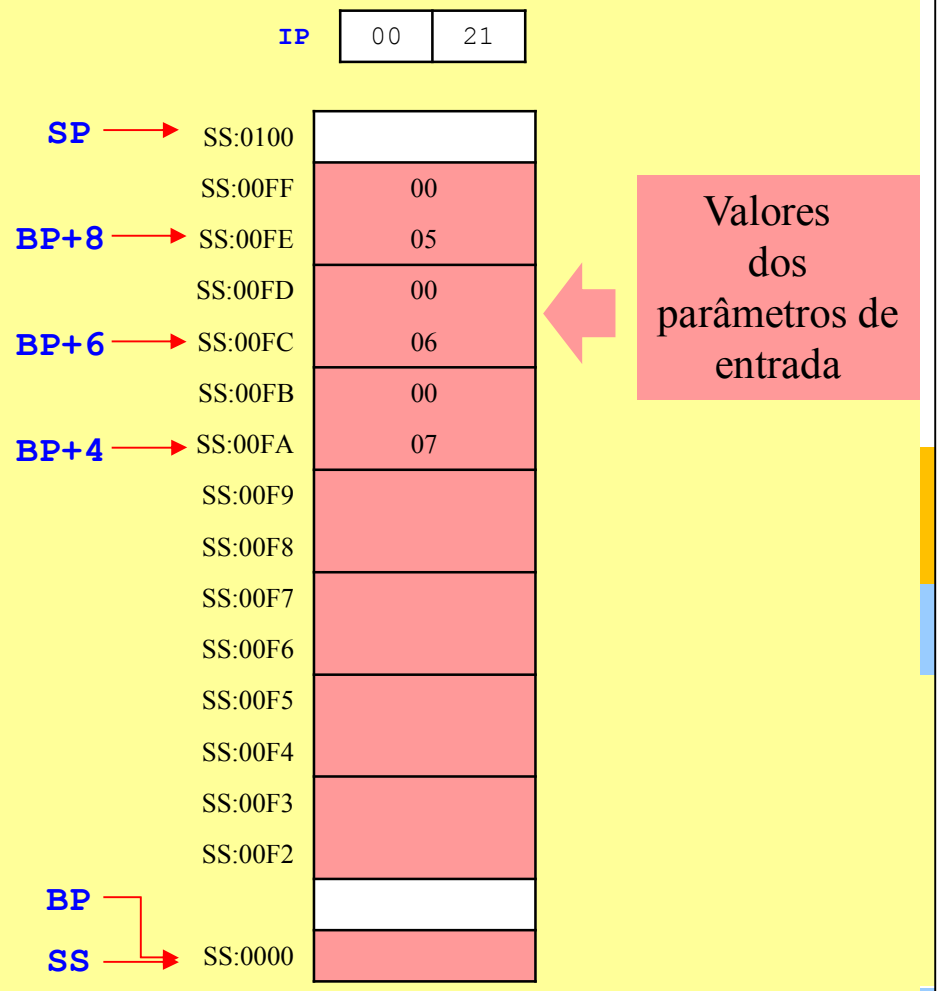
```
; --------------------CODIGO DO PROGRAMA-----------------------
mov  ax,5
push ax    ; guardamos o primeiro parametro na pilha
mov  ax,6
push ax    ; guardamos o segundo param
mov  ax,7
push ax    ; guardamos o terceiro para
call sum3values
...
; --------------------------Procedime
; este é um procedimento para somar t
sum3values:
    ; Recibe:  tres WORDS pasados pela
    ; Retorna: AX que contem o valor d
    ;1. Faz BP apontar para o topo da
    push bp
    mov  bp,sp
    ;2. salva contexto
    push bx ; salva os registradores q
    ;3. tarefa do procedimento
    ;desempilha-se os parâmetros passa
    ;no caso 3 words (16 bits cada)
    mov bx,[bp+8] ; resgata a primeira
    add bx,[bp+6] ; resgata e soma a s
    add bx,[bp+4] ; resgata e soma a t
    mov ax, bx    ; soma esta em AX
    ;4. recupera contexto
    pop bx ; recupera os registradores
    pop bp
    ret 6
IP
```



Exemplo: Passagem de parâmetros pela pilha

exsb003.asm

```
; --------------------CODIGO DO PROGRAMA----------------------
mov  ax,5
push ax    ; guardamos o primeiro parametro na pilha
mov  ax,6
push ax    ; guardamos o segundo param
mov  ax,7
push ax    ; guardamos o terceiro para
call sum3values
IP ...
; -------------------------Procedime
; este é um procedimento para somar t
sum3values:
    ; Recibe:  tres WORDS pasados pela
    ; Retorna: AX que contem o valor d
    ;1. Faz BP apontar para o topo da
    push bp
    mov  bp,sp
    ;2. salva contexto
    push bx ; salva os registradores q
    ;3. tarefa do procedimento
    ;desempilha-se os parâmetros passa
    ;no caso 3 words (16 bits cada)
    mov bx,[bp+8] ; resgata a primeira
    add bx,[bp+6] ; resgata e soma a s
    add bx,[bp+4] ; resgata e soma a t
    mov ax, bx    ; soma esta em AX
    ;4. recupera contexto
    pop bx ; recupera os registradores
    pop bp
    ret 6
```



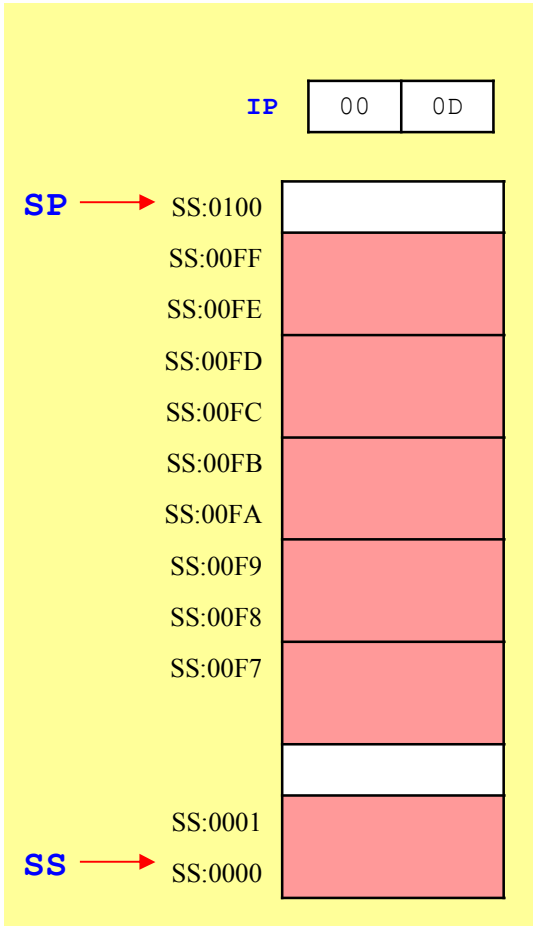
Procedimentos (*Procedures*)

Exemplo: Mecanismo de Chamada e Retorno

exf001.asm

```
                ;PROG . PRINCIPAL
                ....
IP ──► CS:000D  MOV BX,0x03E8
       CS:0010  CALL MYFUNCTION
       CS:0013  ADD AX,7
                ....
                ;procedimento
                MYFUNCTION:
       CS:001A  MOV AX,BX
       CS:001C  RET
```

```
AX=076C  BX=0000  CX=001D  DX=0000  SP=0100
DS=076B  ES=075A  SS=076C  CS=076A  IP=000D
076A:000D BBE803        MOV     BX,03E8
```



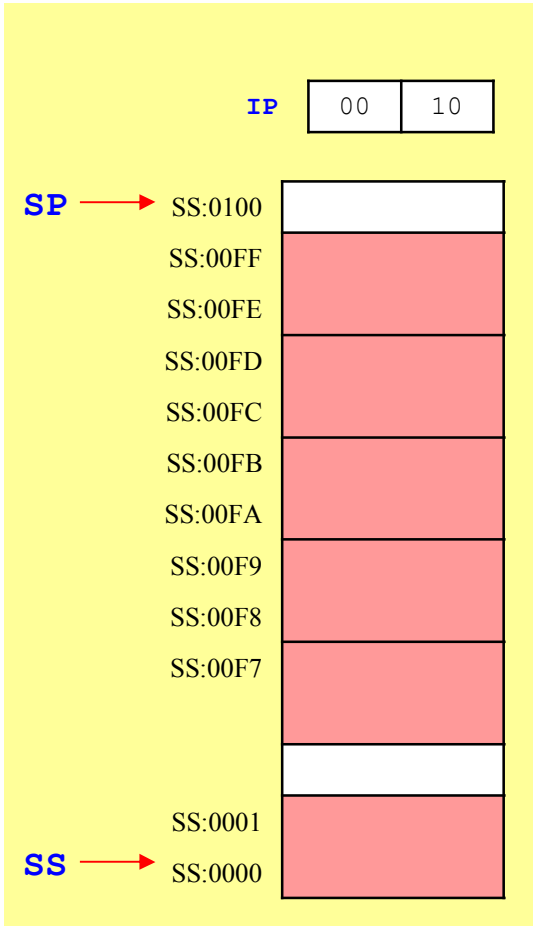
Procedimentos (*Procedures*)

Exemplo: Mecanismo de Chamada e Retorno

exf001.asm

```
                ;PROG . PRINCIPAL
                ....
CS:000D         MOV BX,0x03E8
IP → CS:0010    CALL MYFUNCTION
CS:0013         ADD AX,7
                ....
                ;procedimento
                MYFUNCTION:
CS:001A         MOV AX,BX
CS:001C         RET
```

```
-t
AX=076C  BX=03E8  CX=001D  DX=0000  SP=0100
DS=076B  ES=075A  SS=076C  CS=076A  IP=0010
076A:0010 E80700        CALL    001A
```



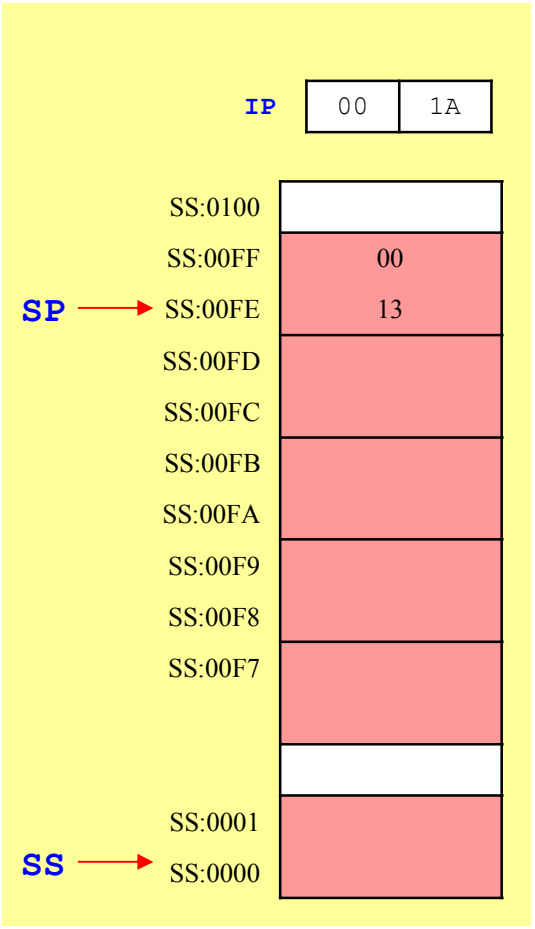
Procedimentos (*Procedures*)

Exemplo: Mecanismo de Chamada e Retorno

exf001.asm

```
                ;PROG . PRINCIPAL
                ....
CS:000D         MOV BX,0x03E8
CS:0010         CALL MYFUNCTION
CS:0013         ADD AX,7
                ....
                ;procedimento
                MYFUNCTION:
IP → CS:001A    MOV AX,BX
CS:001C         RET
```

```
-t
AX=076C  BX=03E8  CX=001D  DX=0000  SP=00FE
DS=076B  ES=075A  SS=076C  CS=076A  IP=001A
076A:001A 89D8          MOV     AX,BX
```



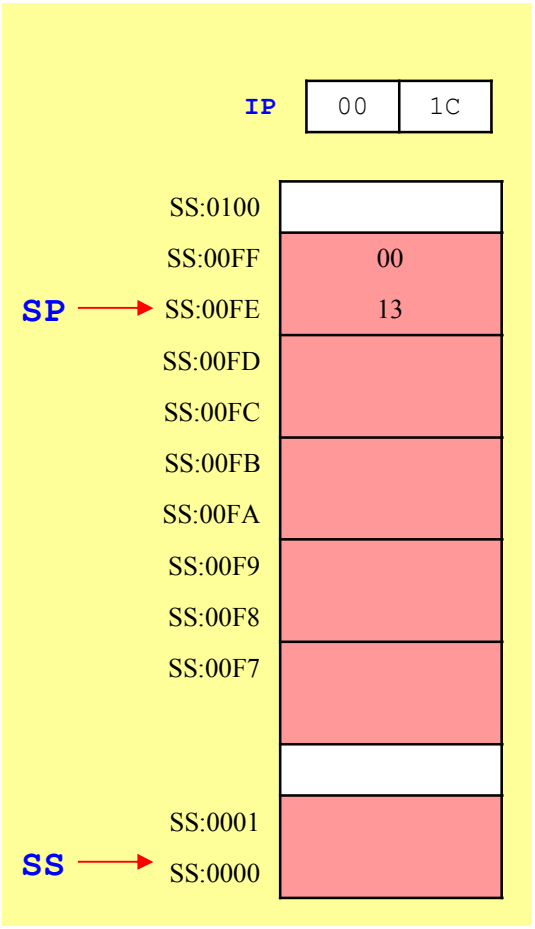
Procedimentos (*Procedures*)

Exemplo: Mecanismo de Chamada e Retorno

exf001.asm

```
                ;PROG . PRINCIPAL
                ....
CS:000D         MOV BX,0x03E8
CS:0010         CALL MYFUNCTION
CS:0013         ADD AX,7
                ....
                ;procedimento
                MYFUNCTION:
CS:001A         MOV AX,BX
IP → CS:001C    RET
```

```
-t
AX=03E8  BX=03E8  CX=001D  DX=0000  SP=00FE
DS=076B  ES=075A  SS=076C  CS=076A  IP=001C
076A:001C C3            RET
```



Procedimentos (*Procedures*)

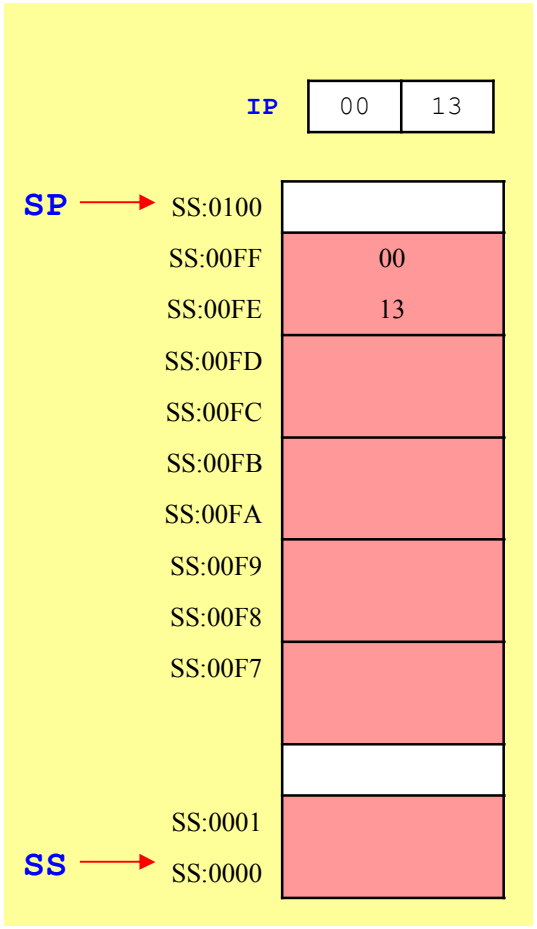
Exemplo: Mecanismo de Chamada e Retorno

exf001.asm

```
                ;PROG . PRINCIPAL
                ....
CS:000D         MOV BX,0x03E8
CS:0010         CALL MYFUNCTION
IP → CS:0013    ADD AX,7
                ....

                ;procedimento
                MYFUNCTION:
CS:001A         MOV AX,BX
CS:001C         RET
```

```
-t
AX=03E8  BX=03E8  CX=001D  DX=0000  SP=0100
DS=076B  ES=075A  SS=076C  CS=076A  IP=0013
076A:0013 050700        ADD     AX,0007
```



Passagem de parâmetros

Passagem de parâmetros

Introdução

- Vamos a estudar duas técnicas para o passagem de parâmetros a um procedimento em *assembly*:
  - Passagem por Registradores.
  - Passagem pela Pilha.

Passagem de parâmetros

Passagem por Registradores

- As vantagens deste método são a sua simplicidade e sua rapidez.

Tudo o que você precisa fazer é:

mover os parâmetros para os registradores antes de chamar o procedimento.

Passagem de parâmetros

Exemplo: Passagem de parâmetros por registradores

exsb002.asm

```
; --------------------CODIGO DO PROGRAMA-----------------------
mov  si, array  ; SI aponta ao array
mov  cx, 5      ; ECX = numero de elementos do array
call ArraySum   ; calcula a soma
mov  [theSum], ax ; retorna o valor da soma em AX
...
; --------------------------Procedimentos--------------------------
; este é um procedimento para somar os valores dos elementos de um array usando
; registradores para passagem de parâmetro.
ArraySum:
  ; Recibe:  SI : aponta ao array de WORDS,
  ;          CX : numero de elementos do array.
  ; Retorna: AX : valor da soma
  push si     ; salva os registradores que serão alterados
  push cx
  mov  ax,0   ; inicia AX a zero
  L1:  add ax,[si]   ; soma cada elemento do array
       add si,2      ; si aponta ao proximo elemento
       loop  L1      ; repete ate que cx==0
  pop cx  ; recupera registradores
  pop si
  ret     ; soma esta em AX
...
; -----------------DEF. VAR,CONST E ALOCACAO-------------------
array  DW 0x0010, 0x0020, 0x0030, 0x0040, 0x0050
theSum: resw 1
```

Salvar contexto (Prologo)

Tarefa

Recuperar contexto (Epilogo)

Passagem de parâmetros

Passagem pela Pilha

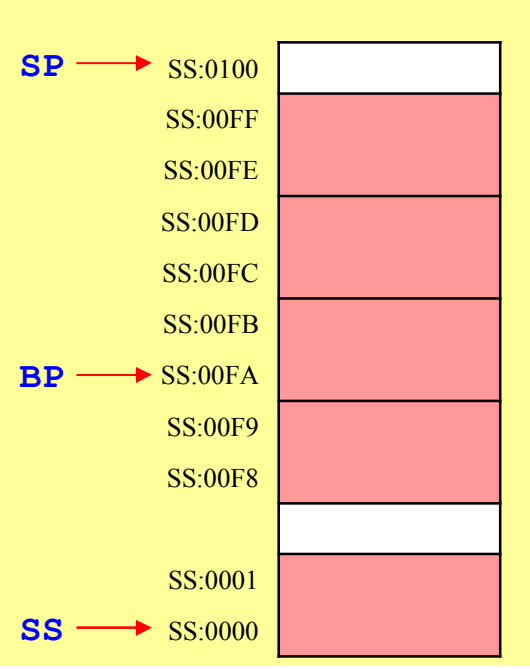
- Este é o método mais poderoso e flexível de passagem de parâmetros. Sua desvantagem é a COMPLEXIDADE.

Tudo o que você precisa fazer é:

PASO 1: movemos todos os valores dos parâmetros para a pilha

PASO 2: acessamos aos valores dos parâmetros dentro do procedimento usando o registrador `BP`.

- **O registrador `SS`:** Aponta o início do segmento de pilha (Base).
- **O registrador `SP`:** Aponta o topo da pilha (define o deslocamento do topo em relação à base). Seu valor é atualizado a cada operação de inserção (`PUSH`) ou remoção (`POP`) de valores na pilha.
- **O registrador `BP`.** Aponta a uma posição da pilha.



Exemplo: Passagem de parâmetros pela pilha

exsb003.asm

```
; --------------------CODIGO DO PROGRAMA-----------------------
mov  ax,5
push ax   ; guardamos o primeiro parametro na pilha
mov  ax,6
push ax   ; guardamos o segundo parametro na pilha
mov  ax,7
push ax   ; guardamos o terceiro parametro na pilha
call sum3values
...
; -------------------------Procedimentos-------------------------
; este é um procedimento para somar tres valores passados pela pilha.
sum3values:
   ; Recibe:  tres WORDS pasados pela pilha.
   ; Retorna: AX que contem o valor da soma
   ;1. Faz BP apontar para o topo da pilha, antes de salvar o contexto
   push bp
   mov  bp,sp
   ;2. salva contexto
   push bx ; salva os registradores que serão alterados
   ;3. tarefa do procedimento
   ;desempilha-se os parâmetros passados pela pilha
   ;no caso 3 words (16 bits cada)
   mov bx,[bp+8] ; resgata a primeira entrada
   add bx,[bp+6] ; resgata e soma a segunda entrada
   add bx,[bp+4] ; resgata e soma a terceira entrada
   mov ax, bx    ; soma esta em AX
   ;4. recupera contexto
   pop bx ; recupera os registradores
   pop bp
   ret 6
```

Salvar contexto (Prologo)

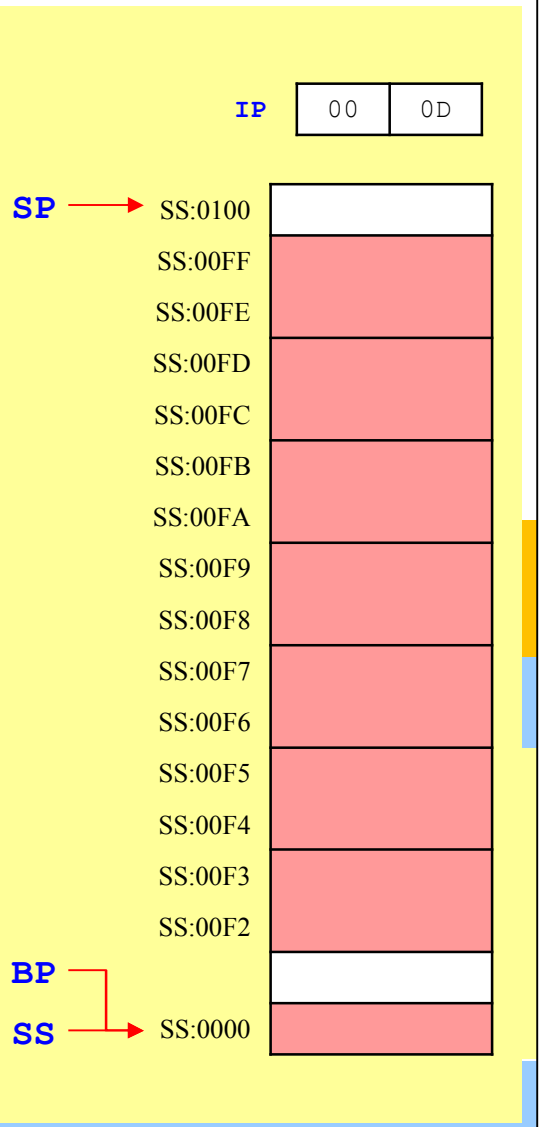
Tarefa

Recuperar contexto (Epilogo)

Exemplo: Passagem de parâmetros pela pilha

exsb003.asm

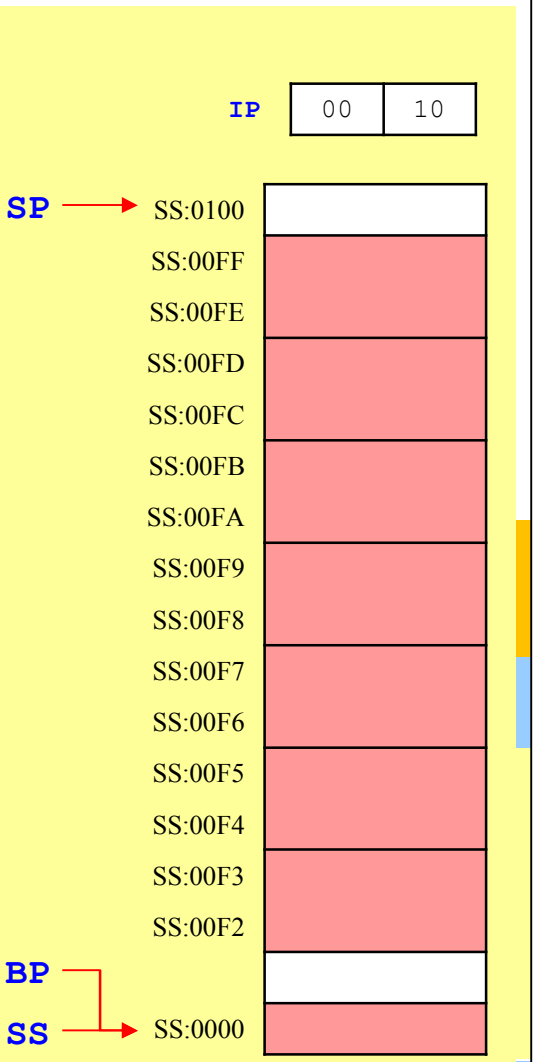
```
; --------------------CODIGO DO PROGRAMA----------------------
IP
mov  ax,5
push ax    ; guardamos o primeiro parametro na pilha
mov  ax,6
push ax    ; guardamos o segundo parametro na pilha
mov  ax,7
push ax    ; guardamos o terceiro parametro na pilha
call sum3values
...
; -------------------------Procedimentos---------------
; este é um procedimento para somar tres valores passado
sum3values:
    ; Recibe:  tres WORDS pasados pela pilha.
    ; Retorna: AX que contem o valor da soma
    ;1. Faz BP apontar para o topo da pilha, antes de sal
    push bp
    mov  bp,sp
    ;2. salva contexto
    push bx ; salva os registradores que serão alterados
    ;3. tarefa do procedimento
    ;desempilha-se os parâmetros passados pela pilha
    ;no caso 3 words (16 bits cada)
    mov bx,[bp+8] ; resgata a primeira entrada
    add bx,[bp+6] ; resgata e soma a segunda entrada
    add bx,[bp+4] ; resgata e soma a terceira entrada
    mov ax, bx    ; soma esta em AX
    ;4. recupera contexto
    pop bx ; recupera os registradores
    pop bp
    ret 6
```



Exemplo: Passagem de parâmetros pela pilha

exsb003.asm

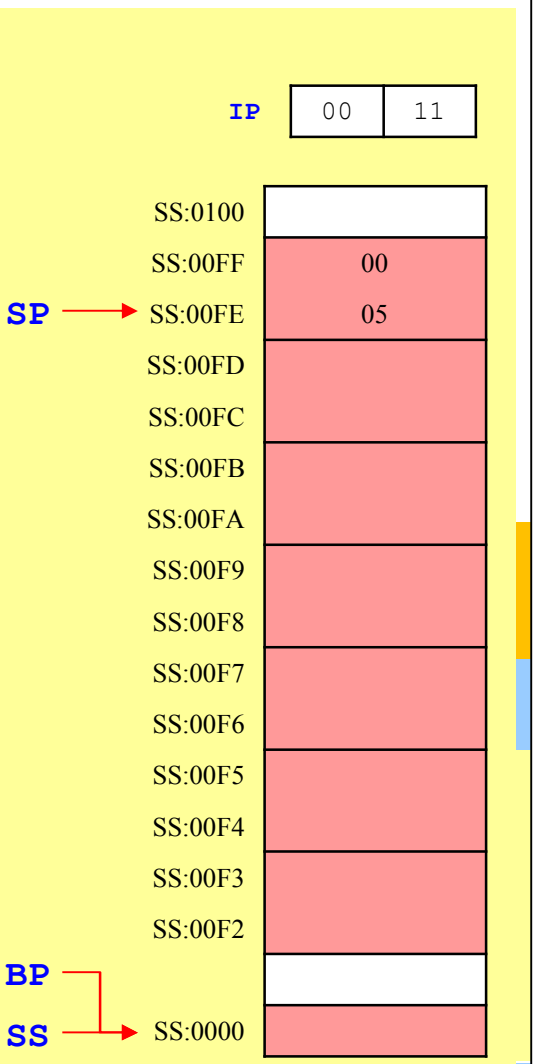
```
; --------------------CODIGO DO PROGRAMA----------------------
mov  ax,5
push ax    ; guardamos o primeiro parametro na pilha
mov  ax,6
push ax    ; guardamos o segundo parametro na pilha
mov  ax,7
push ax    ; guardamos o terceiro parametro na pilha
call sum3values
...
; -------------------------Procedimentos---------------
; este é um procedimento para somar tres valores passado
sum3values:
    ; Recibe:  tres WORDS pasados pela pilha.
    ; Retorna: AX que contem o valor da soma
    ;1. Faz BP apontar para o topo da pilha, antes de sal
    push bp
    mov  bp,sp
    ;2. salva contexto
    push bx ; salva os registradores que serão alterados
    ;3. tarefa do procedimento
    ;desempilha-se os parâmetros passados pela pilha
    ;no caso 3 words (16 bits cada)
    mov bx,[bp+8] ; resgata a primeira entrada
    add bx,[bp+6] ; resgata e soma a segunda entrada
    add bx,[bp+4] ; resgata e soma a terceira entrada
    mov ax, bx    ; soma esta em AX
    ;4. recupera contexto
    pop bx ; recupera os registradores
    pop bp
    ret 6
```



Exemplo: Passagem de parâmetros pela pilha

exsb003.asm

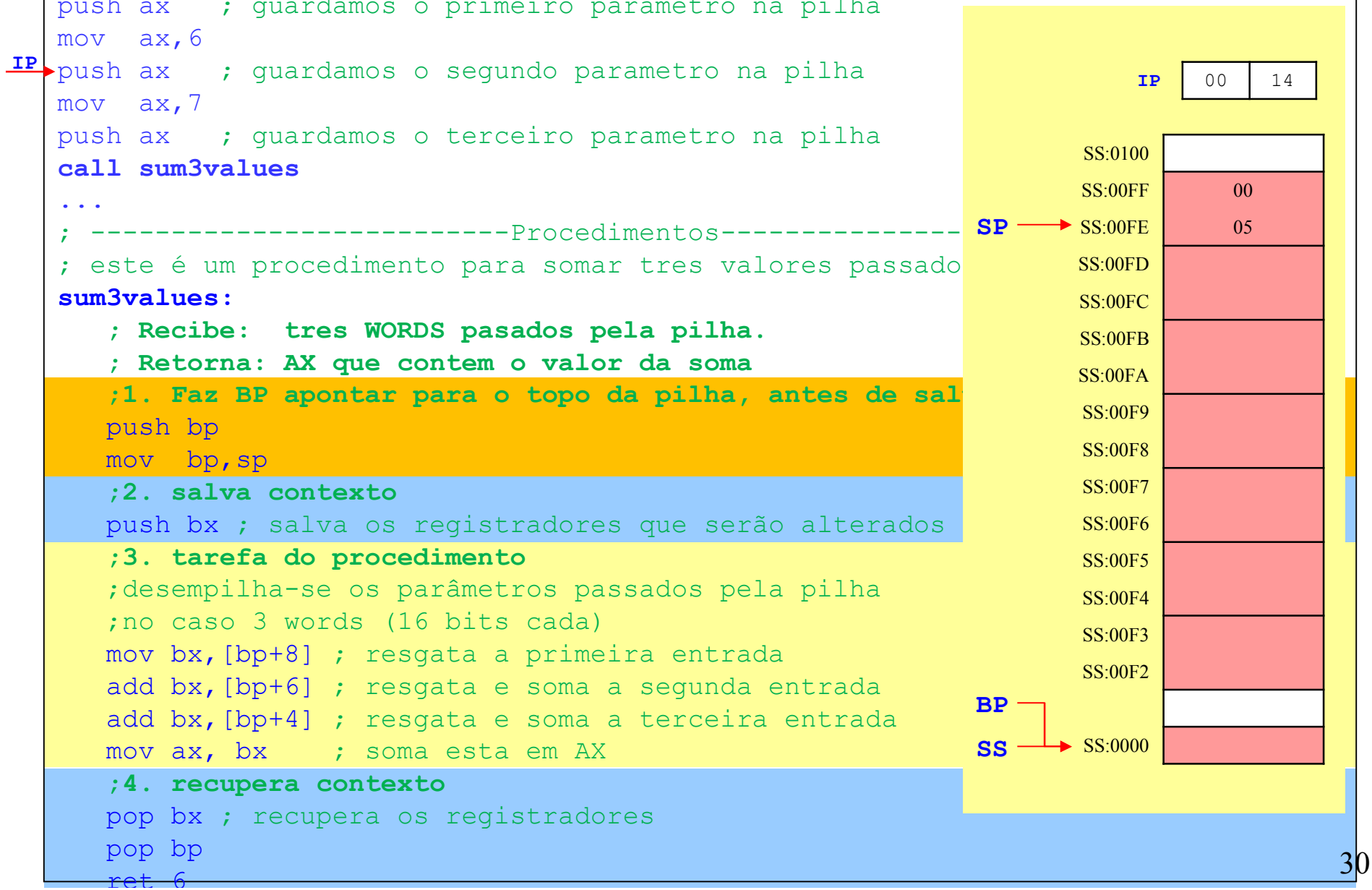
```
; --------------------CODIGO DO PROGRAMA----------------------
mov  ax,5
push ax    ; guardamos o primeiro parametro na pilha
mov  ax,6
push ax    ; guardamos o segundo parametro na pilha
mov  ax,7
push ax    ; guardamos o terceiro parametro na pilha
call sum3values
...
; -------------------------Procedimentos---------------
; este é um procedimento para somar tres valores passado
sum3values:
    ; Recibe:  tres WORDS pasados pela pilha.
    ; Retorna: AX que contem o valor da soma
    ;1. Faz BP apontar para o topo da pilha, antes de sal
    push bp
    mov  bp,sp
    ;2. salva contexto
    push bx ; salva os registradores que serão alterados
    ;3. tarefa do procedimento
    ;desempilha-se os parâmetros passados pela pilha
    ;no caso 3 words (16 bits cada)
    mov bx,[bp+8] ; resgata a primeira entrada
    add bx,[bp+6] ; resgata e soma a segunda entrada
    add bx,[bp+4] ; resgata e soma a terceira entrada
    mov ax, bx    ; soma esta em AX
    ;4. recupera contexto
    pop bx ; recupera os registradores
    pop bp
    ret 6
```



Exemplo: Passagem de parâmetros pela pilha

exsb003.asm

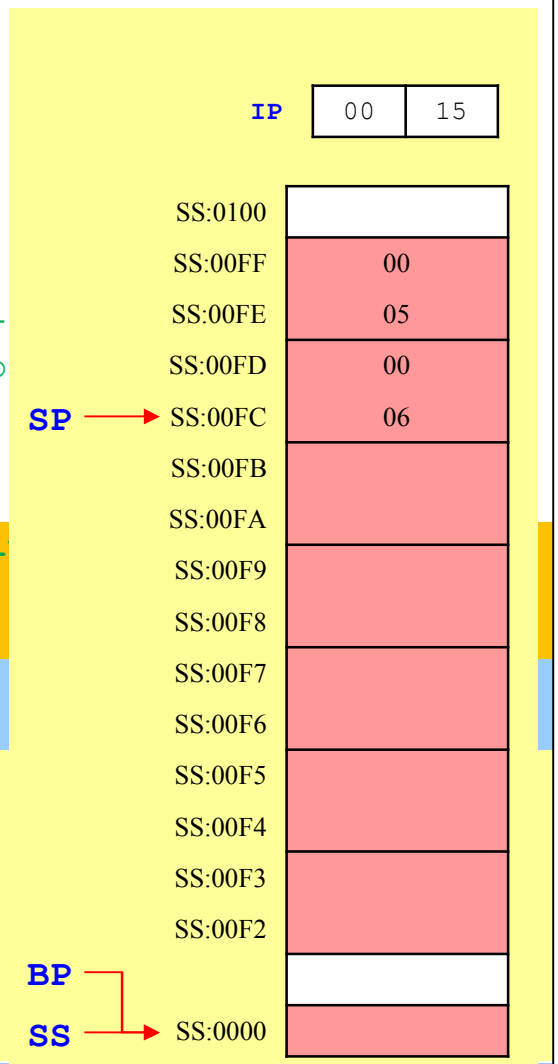
```
; --------------------CODIGO DO PROGRAMA----------------------
mov  ax,5
push ax    ; guardamos o primeiro parametro na pilha
mov  ax,6
push ax    ; guardamos o segundo parametro na pilha
mov  ax,7
push ax    ; guardamos o terceiro parametro na pilha
call sum3values
...
; -------------------------Procedimentos---------------
; este é um procedimento para somar tres valores passado
sum3values:
    ; Recibe:  tres WORDS pasados pela pilha.
    ; Retorna: AX que contem o valor da soma
    ;1. Faz BP apontar para o topo da pilha, antes de sal
    push bp
    mov  bp,sp
    ;2. salva contexto
    push bx ; salva os registradores que serão alterados
    ;3. tarefa do procedimento
    ;desempilha-se os parâmetros passados pela pilha
    ;no caso 3 words (16 bits cada)
    mov bx,[bp+8] ; resgata a primeira entrada
    add bx,[bp+6] ; resgata e soma a segunda entrada
    add bx,[bp+4] ; resgata e soma a terceira entrada
    mov ax, bx    ; soma esta em AX
    ;4. recupera contexto
    pop bx ; recupera os registradores
    pop bp
    ret 6
```



Exemplo: Passagem de parâmetros pela pilha

exsb003.asm

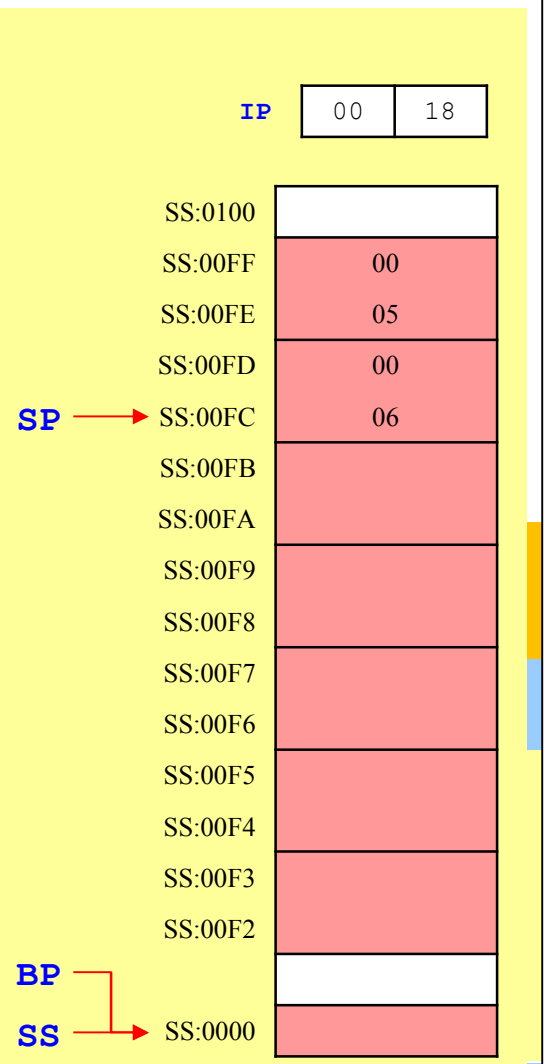
```
; --------------------CODIGO DO PROGRAMA----------------------
mov  ax,5
push ax    ; guardamos o primeiro parametro na pilha
mov  ax,6
push ax    ; guardamos o segundo parametro na pilha
IP → mov  ax,7
push ax    ; guardamos o terceiro parametro na pilha
call sum3values
...
; -------------------------Procedimentos---------------
; este é um procedimento para somar tres valores passado
sum3values:
    ; Recibe:  tres WORDS pasados pela pilha.
    ; Retorna: AX que contem o valor da soma
    ;1. Faz BP apontar para o topo da pilha, antes de sal
    push bp
    mov  bp,sp
    ;2. salva contexto
    push bx ; salva os registradores que serão alterados
    ;3. tarefa do procedimento
    ;desempilha-se os parâmetros passados pela pilha
    ;no caso 3 words (16 bits cada)
    mov bx,[bp+8] ; resgata a primeira entrada
    add bx,[bp+6] ; resgata e soma a segunda entrada
    add bx,[bp+4] ; resgata e soma a terceira entrada
    mov ax, bx    ; soma esta em AX
    ;4. recupera contexto
    pop bx ; recupera os registradores
    pop bp
    ret 6
```



Exemplo: Passagem de parâmetros pela pilha

exsb003.asm

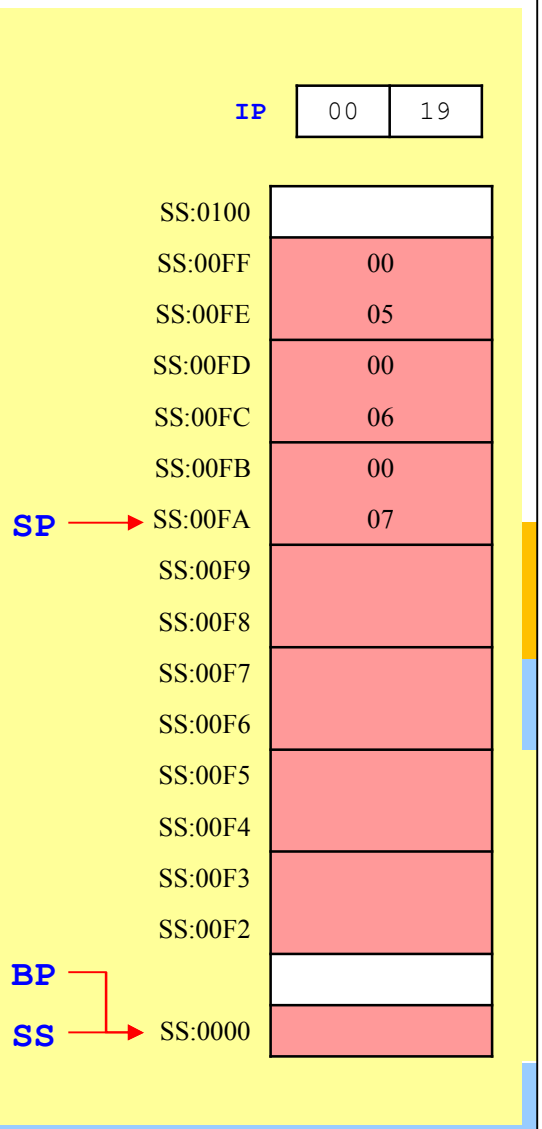
```
; --------------------CODIGO DO PROGRAMA----------------------
mov  ax,5
push ax    ; guardamos o primeiro parametro na pilha
mov  ax,6
push ax    ; guardamos o segundo parametro na pilha
mov  ax,7
IP → push ax    ; guardamos o terceiro parametro na pilha
call sum3values
...
; -------------------------Procedimentos---------------
; este é um procedimento para somar tres valores passado
sum3values:
    ; Recibe:  tres WORDS pasados pela pilha.
    ; Retorna: AX que contem o valor da soma
    ;1. Faz BP apontar para o topo da pilha, antes de sal
    push bp
    mov  bp,sp
    ;2. salva contexto
    push bx ; salva os registradores que serão alterados
    ;3. tarefa do procedimento
    ;desempilha-se os parâmetros passados pela pilha
    ;no caso 3 words (16 bits cada)
    mov bx,[bp+8] ; resgata a primeira entrada
    add bx,[bp+6] ; resgata e soma a segunda entrada
    add bx,[bp+4] ; resgata e soma a terceira entrada
    mov ax, bx    ; soma esta em AX
    ;4. recupera contexto
    pop bx ; recupera os registradores
    pop bp
    ret 6
```



Exemplo: Passagem de parâmetros pela pilha

exsb003.asm

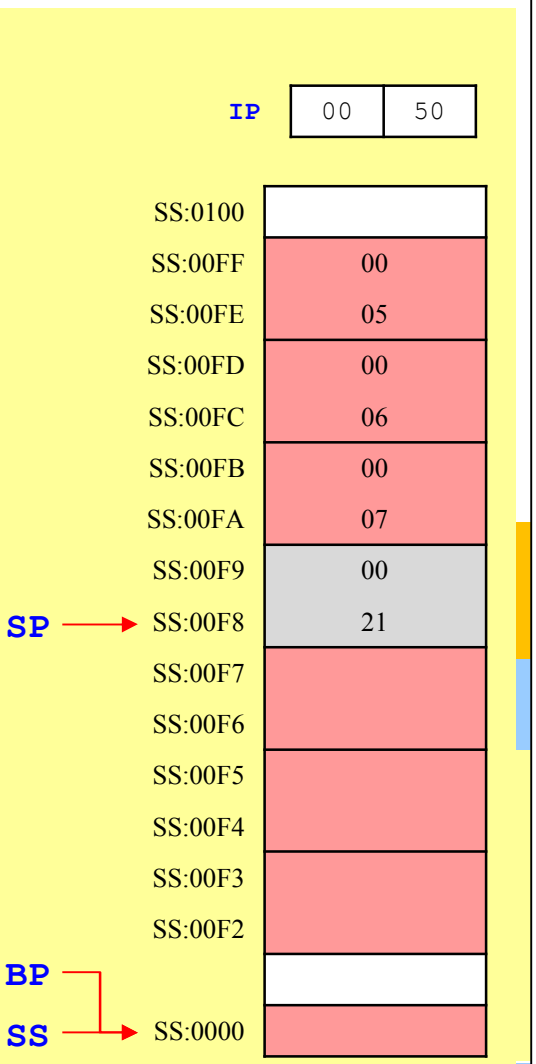
```
; --------------------CODIGO DO PROGRAMA----------------------
mov  ax,5
push ax   ; guardamos o primeiro parametro na pilha
mov  ax,6
push ax   ; guardamos o segundo parametro na pilha
mov  ax,7
push ax   ; guardamos o terceiro parametro na pilha
IP call sum3values
...
; -------------------------Procedimentos---------------
; este é um procedimento para somar tres valores passado
sum3values:
   ; Recibe:  tres WORDS pasados pela pilha.
   ; Retorna: AX que contem o valor da soma
   ;1. Faz BP apontar para o topo da pilha, antes de sal
   push bp
   mov  bp,sp
   ;2. salva contexto
   push bx ; salva os registradores que serão alterados
   ;3. tarefa do procedimento
   ;desempilha-se os parâmetros passados pela pilha
   ;no caso 3 words (16 bits cada)
   mov bx,[bp+8] ; resgata a primeira entrada
   add bx,[bp+6] ; resgata e soma a segunda entrada
   add bx,[bp+4] ; resgata e soma a terceira entrada
   mov ax, bx    ; soma esta em AX
   ;4. recupera contexto
   pop bx ; recupera os registradores
   pop bp
   ret 6
```



Exemplo: Passagem de parâmetros pela pilha

exsb003.asm

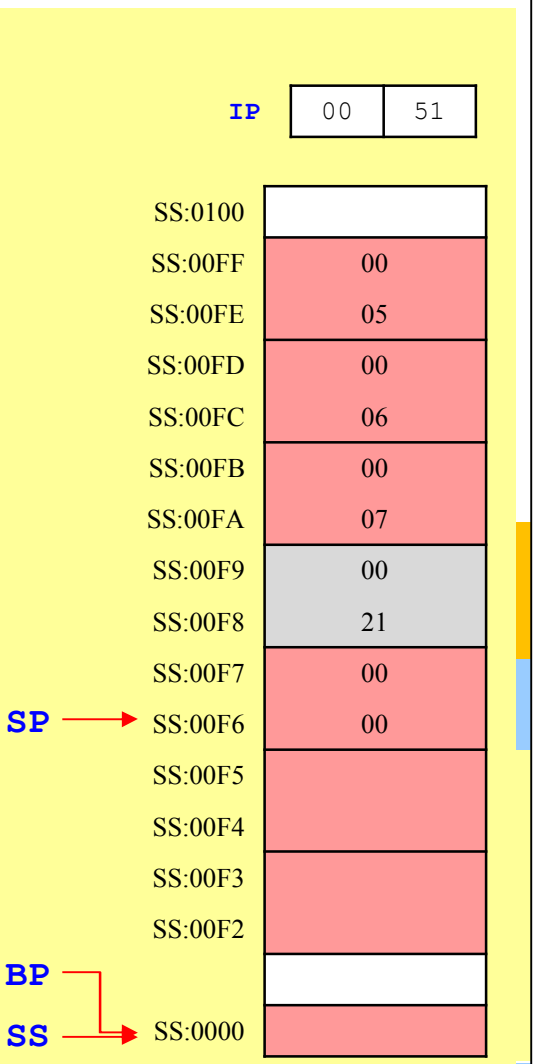
```
; --------------------CODIGO DO PROGRAMA----------------------
mov  ax,5
push ax   ; guardamos o primeiro parametro na pilha
mov  ax,6
push ax   ; guardamos o segundo parametro na pilha
mov  ax,7
push ax   ; guardamos o terceiro parametro na pilha
call sum3values
...
; -------------------------Procedimentos---------------
; este é um procedimento para somar tres valores passado
sum3values:
   ; Recibe:  tres WORDS pasados pela pilha.
   ; Retorna: AX que contem o valor da soma
   ;1. Faz BP apontar para o topo da pilha, antes de sal
IP push bp
   mov  bp,sp
   ;2. salva contexto
   push bx ; salva os registradores que serão alterados
   ;3. tarefa do procedimento
   ;desempilha-se os parâmetros passados pela pilha
   ;no caso 3 words (16 bits cada)
   mov bx,[bp+8] ; resgata a primeira entrada
   add bx,[bp+6] ; resgata e soma a segunda entrada
   add bx,[bp+4] ; resgata e soma a terceira entrada
   mov ax, bx    ; soma esta em AX
   ;4. recupera contexto
   pop bx ; recupera os registradores
   pop bp
   ret 6
```



Exemplo: Passagem de parâmetros pela pilha

exsb003.asm

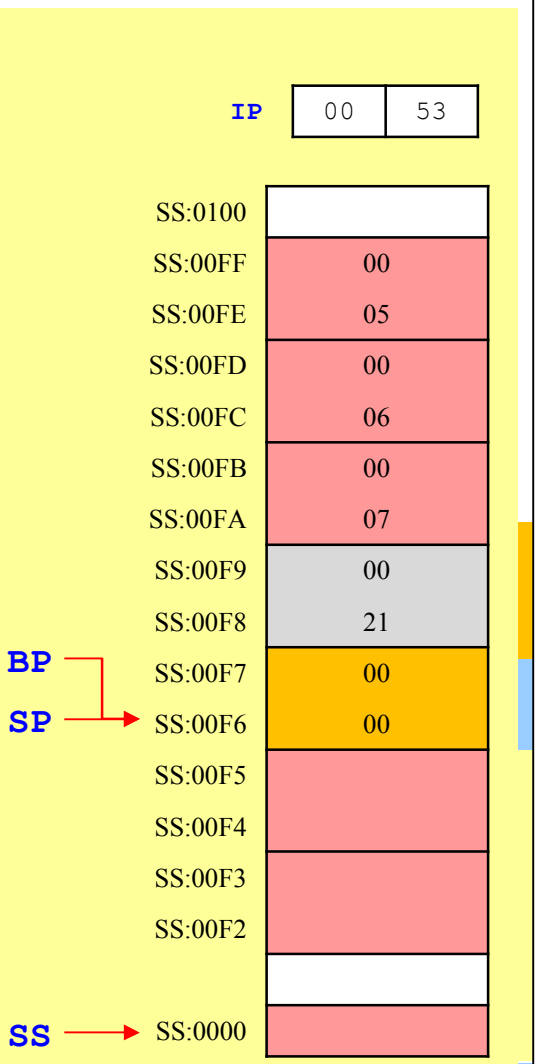
```
; --------------------CODIGO DO PROGRAMA----------------------
mov  ax,5
push ax   ; guardamos o primeiro parametro na pilha
mov  ax,6
push ax   ; guardamos o segundo parametro na pilha
mov  ax,7
push ax   ; guardamos o terceiro parametro na pilha
call sum3values
...
; -------------------------Procedimentos---------------
; este é um procedimento para somar tres valores passado
sum3values:
   ; Recibe:  tres WORDS pasados pela pilha.
   ; Retorna: AX que contem o valor da soma
   ;1. Faz BP apontar para o topo da pilha, antes de sal
   push bp
IP ▶ mov  bp,sp
   ;2. salva contexto
   push bx ; salva os registradores que serão alterados
   ;3. tarefa do procedimento
   ;desempilha-se os parâmetros passados pela pilha
   ;no caso 3 words (16 bits cada)
   mov bx,[bp+8] ; resgata a primeira entrada
   add bx,[bp+6] ; resgata e soma a segunda entrada
   add bx,[bp+4] ; resgata e soma a terceira entrada
   mov ax, bx    ; soma esta em AX
   ;4. recupera contexto
   pop bx ; recupera os registradores
   pop bp
   ret 6
```



Exemplo: Passagem de parâmetros pela pilha

exsb003.asm

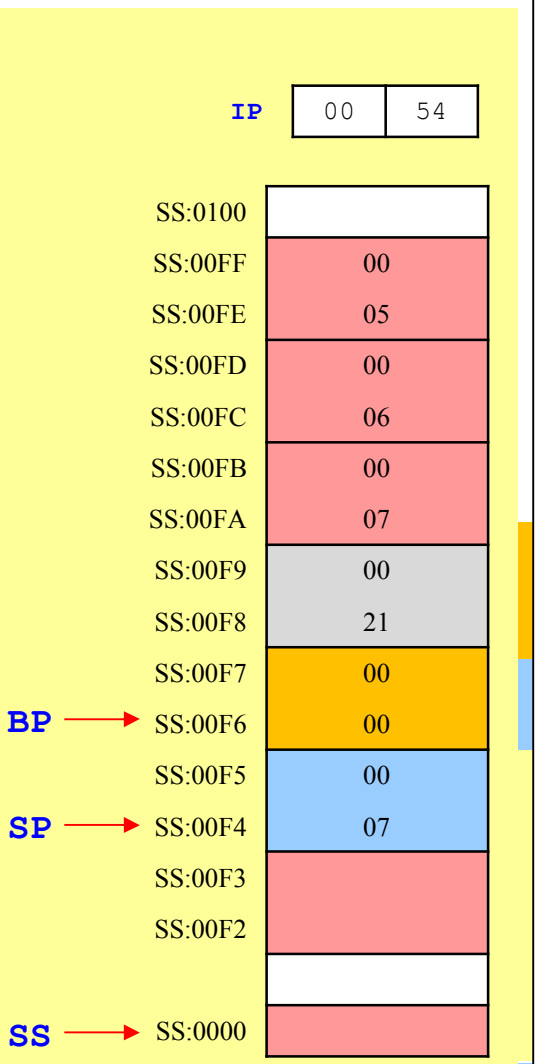
```
; --------------------CODIGO DO PROGRAMA-----------------------
mov  ax,5
push ax   ; guardamos o primeiro parametro na pilha
mov  ax,6
push ax   ; guardamos o segundo parametro na pilha
mov  ax,7
push ax   ; guardamos o terceiro parametro na pilha
call sum3values
...
; -------------------------Procedimentos---------------
; este é um procedimento para somar tres valores passado
sum3values:
   ; Recibe:  tres WORDS pasados pela pilha.
   ; Retorna: AX que contem o valor da soma
   ;1. Faz BP apontar para o topo da pilha, antes de sal
   push bp
   mov  bp,sp
   ;2. salva contexto
IP push bx ; salva os registradores que serão alterados
   ;3. tarefa do procedimento
   ;desempilha-se os parâmetros passados pela pilha
   ;no caso 3 words (16 bits cada)
   mov bx,[bp+8] ; resgata a primeira entrada
   add bx,[bp+6] ; resgata e soma a segunda entrada
   add bx,[bp+4] ; resgata e soma a terceira entrada
   mov ax, bx    ; soma esta em AX
   ;4. recupera contexto
   pop bx ; recupera os registradores
   pop bp
   ret 6
```



Exemplo: Passagem de parâmetros pela pilha

exsb003.asm

```
; --------------------CODIGO DO PROGRAMA----------------------
mov  ax,5
push ax   ; guardamos o primeiro parametro na pilha
mov  ax,6
push ax   ; guardamos o segundo parametro na pilha
mov  ax,7
push ax   ; guardamos o terceiro parametro na pilha
call sum3values
...
; -------------------------Procedimentos--------------
; este é um procedimento para somar tres valores passado
sum3values:
   ; Recibe:  tres WORDS pasados pela pilha.
   ; Retorna: AX que contem o valor da soma
   ;1. Faz BP apontar para o topo da pilha, antes de sal
   push bp
   mov  bp,sp
   ;2. salva contexto
   push bx ; salva os registradores que serão alterados
   ;3. tarefa do procedimento
   ;desempilha-se os parâmetros passados pela pilha
   ;no caso 3 words (16 bits cada)
IP mov bx,[bp+8] ; resgata a primeira entrada
   add bx,[bp+6] ; resgata e soma a segunda entrada
   add bx,[bp+4] ; resgata e soma a terceira entrada
   mov ax, bx    ; soma esta em AX
   ;4. recupera contexto
   pop bx ; recupera os registradores
   pop bp
   ret 6
```



Exemplo: Passagem de parâmetros pela pilha

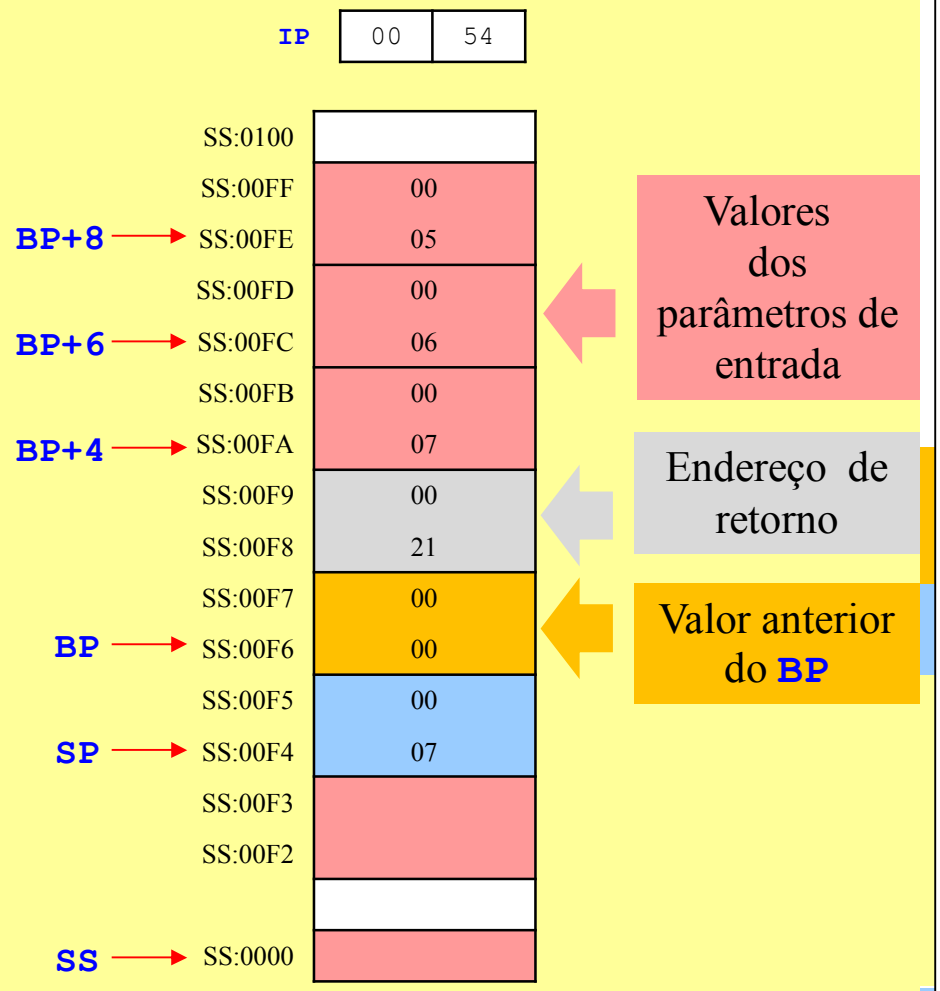
exsb003.asm

```
; --------------------CODIGO DO PROGRAMA----------------------
mov  ax,5
push ax    ; guardamos o primeiro parametro na pilha
mov  ax,6
push ax    ; guardamos o segundo param
mov  ax,7
push ax    ; guardamos o terceiro para
call
...
; --
; es
sum3
    ;
    ; Retorna: AX que contem o valor d
    ;1. Faz BP
    push bp
    mov  bp,sp
    ;2. salva
    push bx ;
    ;3. tarefa
    ;desempilha-se os parâmetros passa
    ;no caso 3
IP  mov bx,[bp
    add bx,[bp
    add bx,[bp
    mov ax, bx
    ;4. recupe
    pop bx ; recupera os registradores
    pop bp
    ret 6
```

Conclusão:
O registrador **BP** permite percorrer a pilha e acessar já seja aos parâmetros passados ao procedimento ou as variáveis locais do procedimento.

Os valores passados como parâmetros encontram-se endereçados pelo registrador **BP** nas posições BP+4,BP+6, BP+8

Os valores dos registradores a ser usados pelo procedimento estão endereçados pelo registrador **BP** nas posições BP-2,BP-4,BP-6

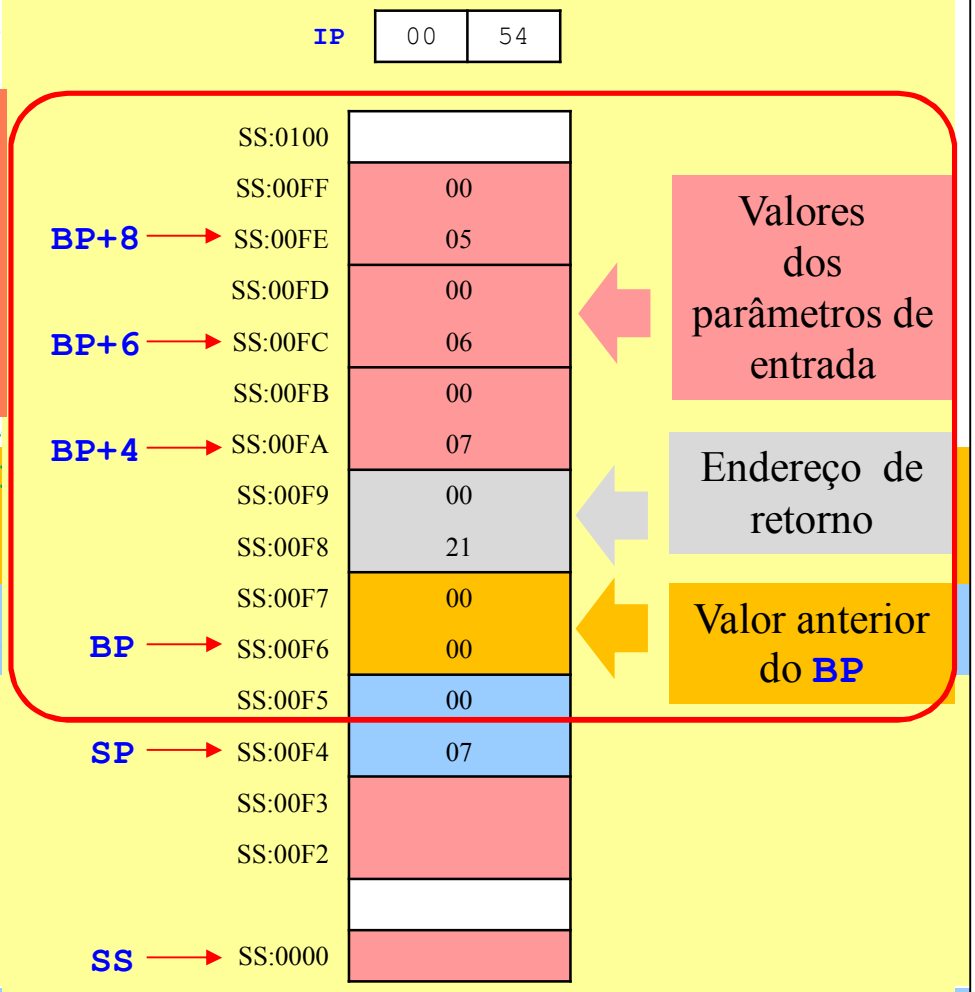


Exemplo: Passagem de parâmetros pela pilha

exsb003.asm

```
; --------------------CODIGO DO PROGRAMA----------------------
mov  ax,5
push ax    ; guardamos o primeiro parametro na pilha
mov  ax,6
push ax    ; guardamos o segundo param
mov  ax,7
push ax
call sum3v
...
; --------
; este é u
sum3values
    ; Recib
    ; Retorna: AX que contem o valor d
    ;1. Faz BP apontar para o topo da
    push bp
    mov  bp,sp
    ;2. salva contexto
    push bx ; salva os registradores q
    ;3. tarefa do procedimento
    ;desempilha-se os parâmetros passa
    ;no caso 3 words (16 bits cada)
IP  mov bx,[bp+8] ; resgata a primeira
    add bx,[bp+6] ; resgata e soma a s
    add bx,[bp+4] ; resgata e soma a t
    mov ax, bx    ; soma esta em AX
    ;4. recupera contexto
    pop bx ; recupera os registradores
    pop bp
    ret 6
```

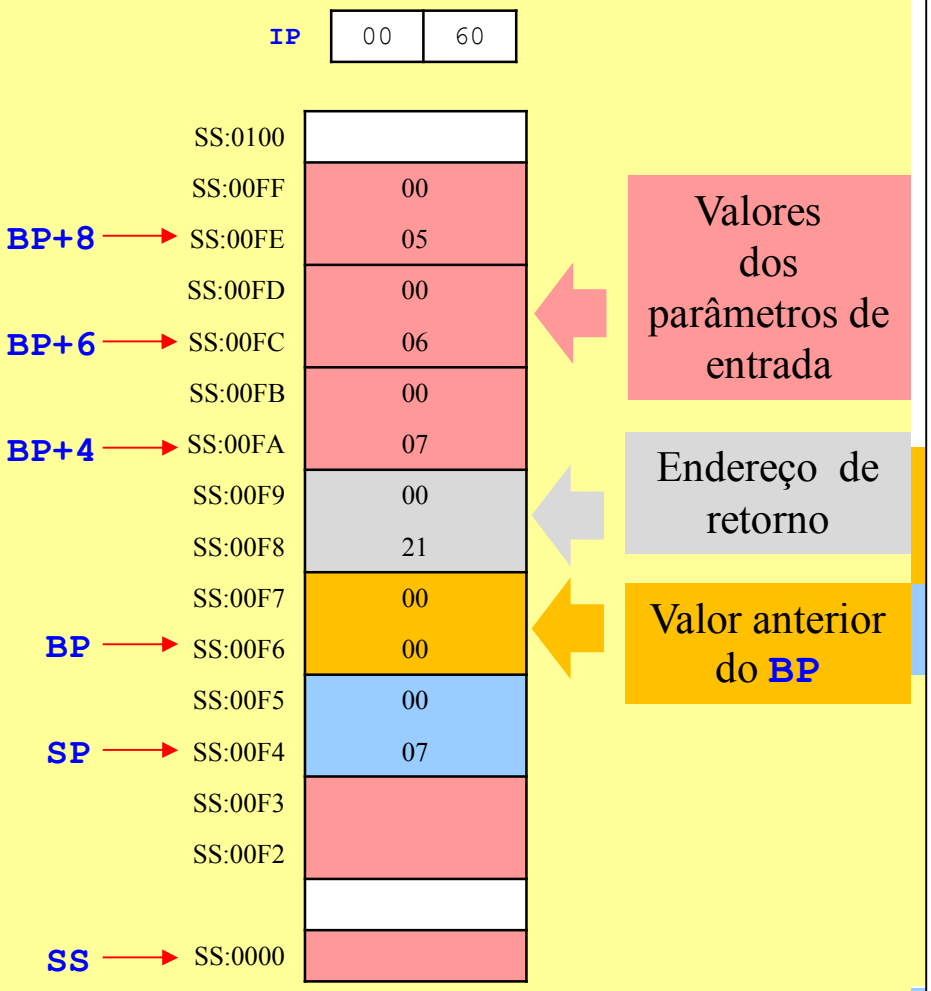
Os valores dos parâmetros, endereços de retorno e o valor anterior do registrado **BP** armazenado na pilha, são chamados de ***Stack Frame*** ou ***Resgistro de Activação.***



Exemplo: Passagem de parâmetros pela pilha

exsb003.asm

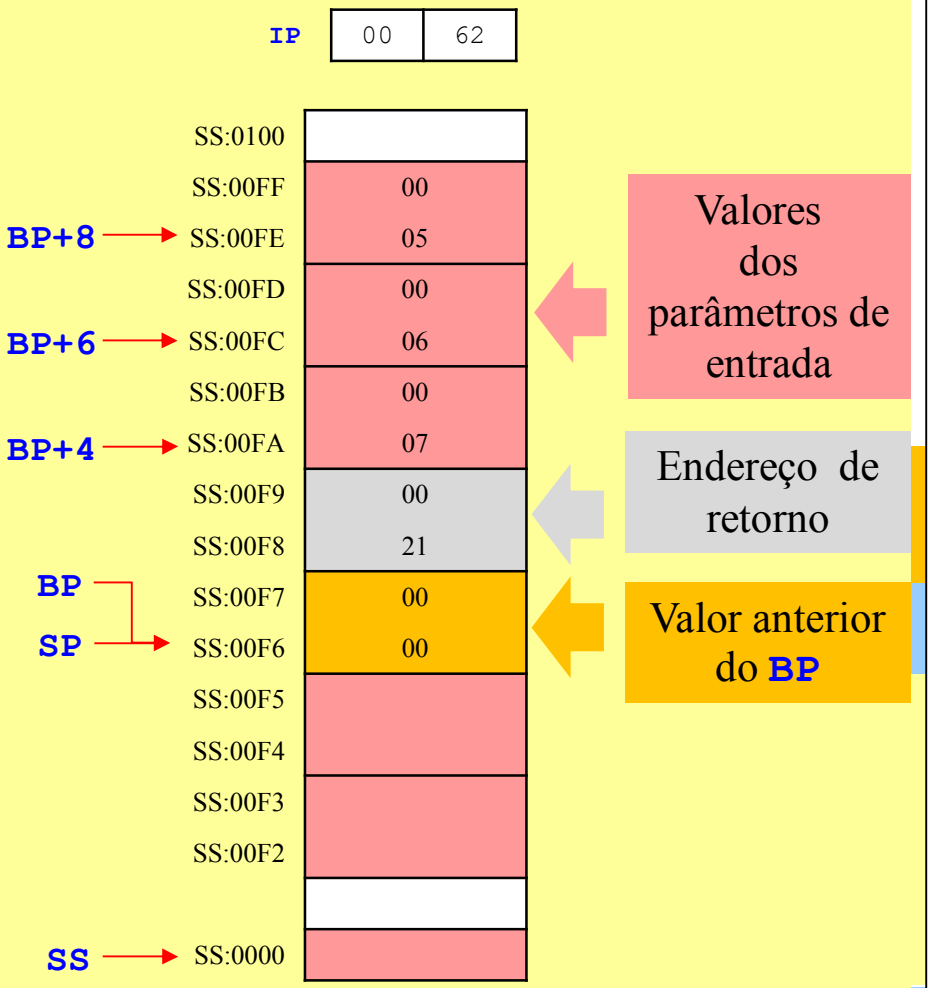
```
; --------------------CODIGO DO PROGRAMA-----------------------
mov  ax,5
push ax    ; guardamos o primeiro parametro na pilha
mov  ax,6
push ax    ; guardamos o segundo param
mov  ax,7
push ax    ; guardamos o terceiro para
call sum3values
...
; --------------------------Procedime
; este é um procedimento para somar t
sum3values:
    ; Recibe:  tres WORDS pasados pela
    ; Retorna: AX que contem o valor d
    ;1. Faz BP apontar para o topo da
    push bp
    mov  bp,sp
    ;2. salva contexto
    push bx ; salva os registradores q
    ;3. tarefa do procedimento
    ;desempilha-se os parâmetros passa
    ;no caso 3 words (16 bits cada)
    mov bx,[bp+8] ; resgata a primeira
    add bx,[bp+6] ; resgata e soma a s
    add bx,[bp+4] ; resgata e soma a t
    mov ax, bx    ; soma esta em AX
    ;4. recupera contexto
IP  pop bx ; recupera os registradores
    pop bp
    ret 6
```



Exemplo: Passagem de parâmetros pela pilha

exsb003.asm

```
; --------------------CODIGO DO PROGRAMA-----------------------
mov  ax,5
push ax    ; guardamos o primeiro parametro na pilha
mov  ax,6
push ax    ; guardamos o segundo param
mov  ax,7
push ax    ; guardamos o terceiro para
call sum3values
...
; --------------------------Procedime
; este é um procedimento para somar t
sum3values:
    ; Recibe:  tres WORDS pasados pela
    ; Retorna: AX que contem o valor d
    ;1. Faz BP apontar para o topo da
    push bp
    mov  bp,sp
    ;2. salva contexto
    push bx ; salva os registradores q
    ;3. tarefa do procedimento
    ;desempilha-se os parâmetros passa
    ;no caso 3 words (16 bits cada)
    mov bx,[bp+8] ; resgata a primeira
    add bx,[bp+6] ; resgata e soma a s
    add bx,[bp+4] ; resgata e soma a t
    mov ax, bx    ; soma esta em AX
    ;4. recupera contexto
    pop bx ; recupera os registradores
IP  pop bp
    ret 6
```

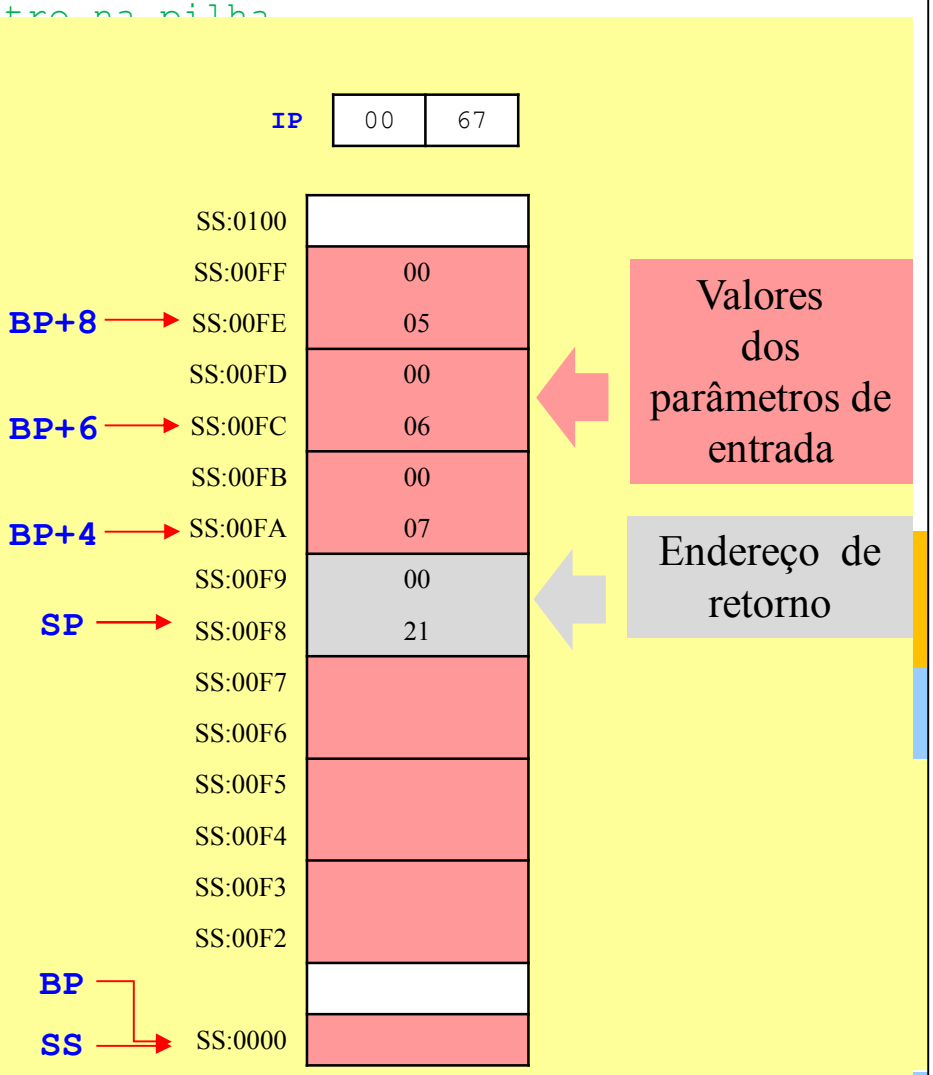


Exemplo: Passagem de parâmetros pela pilha

exsb003.asm

```
; --------------------CODIGO DO PROGRAMA-----------------------
mov  ax,5
push ax    ; guardamos o primeiro parametro na pilha
mov  ax,6
push ax    ; guardamos o segundo param
mov  ax,7
push ax    ; guardamos o terceiro para
call sum3values
...
; --------------------------Procedime
; este é um procedimento para somar t
sum3values:
    ; Recibe:  tres WORDS pasados pela
    ; Retorna: AX que contem o valor d
    ;1. Faz BP apontar para o topo da
    push bp
    mov  bp,sp
    ;2. salva contexto
    push bx ; salva os registradores q
    ;3. tarefa do procedimento
    ;desempilha-se os parâmetros passa
    ;no caso 3 words (16 bits cada)
    mov bx,[bp+8] ; resgata a primeira
    add bx,[bp+6] ; resgata e soma a s
    add bx,[bp+4] ; resgata e soma a t
    mov ax, bx    ; soma esta em AX
    ;4. recupera contexto
    pop bx ; recupera os registradores
    pop bp
    ret 6
```

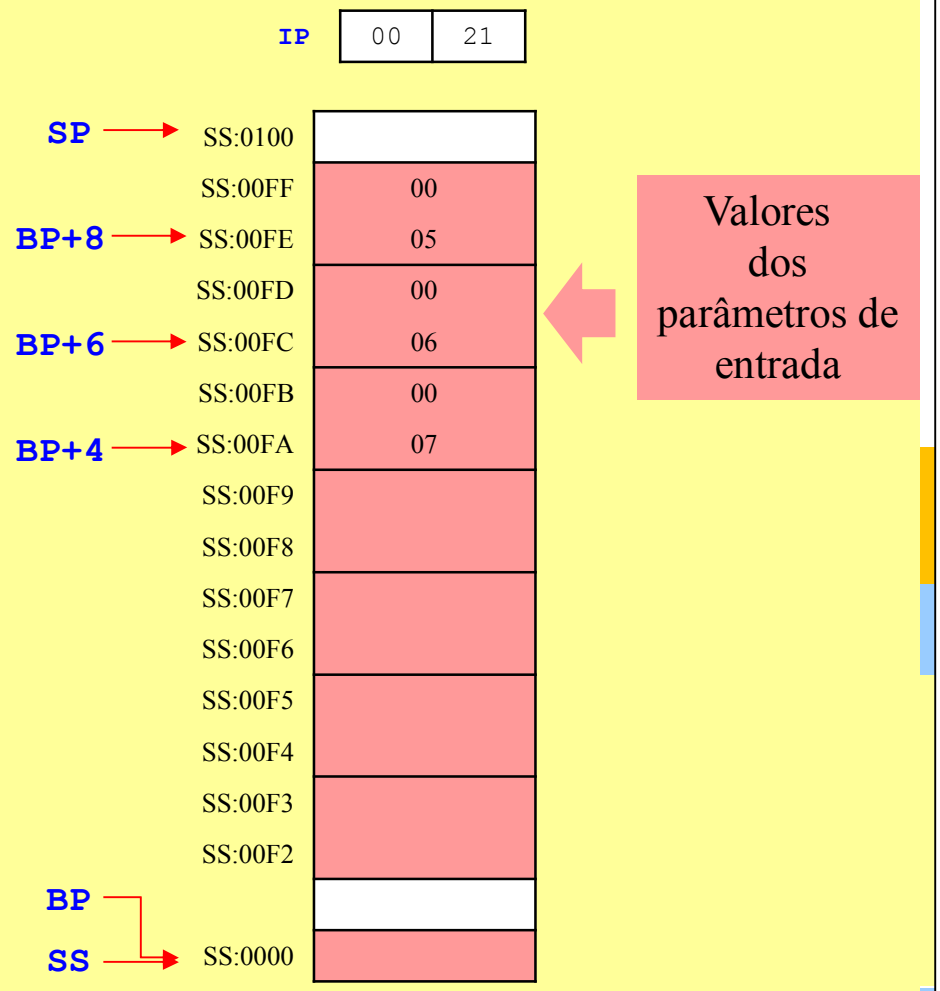
IP



Exemplo: Passagem de parâmetros pela pilha

exsb003.asm

```
; --------------------CODIGO DO PROGRAMA-----------------------
mov  ax,5
push ax    ; guardamos o primeiro parametro na pilha
mov  ax,6
push ax    ; guardamos o segundo param
mov  ax,7
push ax    ; guardamos o terceiro para
call sum3values
IP ...
; -------------------------Procedime
; este é um procedimento para somar t
sum3values:
    ; Recibe:  tres WORDS pasados pela
    ; Retorna: AX que contem o valor d
    ;1. Faz BP apontar para o topo da
    push bp
    mov  bp,sp
    ;2. salva contexto
    push bx ; salva os registradores q
    ;3. tarefa do procedimento
    ;desempilha-se os parâmetros passa
    ;no caso 3 words (16 bits cada)
    mov bx,[bp+8] ; resgata a primeira
    add bx,[bp+6] ; resgata e soma a s
    add bx,[bp+4] ; resgata e soma a t
    mov ax, bx    ; soma esta em AX
    ;4. recupera contexto
    pop bx ; recupera os registradores
    pop bp
    ret 6
```



Laboratório

Laboratório

Atenção

- Lembrar que:
  - Comando para montar a pasta de trabalho no DOSBox (`c:\sistemb1\frsm`)

```
mount c c:\sistemb1\frsm\⏎
c:⏎
```

  - Comando para gerar o arquivo `.OBJ`: `nasm nome_arquivo⏎`
  - Comando para gerar o arquivo `.EXE`: `freelink nome_arquivo⏎`
  - Depurar o arquivo `.EXE`: `debug nome_arquivo.exe⏎`
    - Para depurar linha por linha é o comando `t`.
  - Executar o `.EXE`: `nome_arquivo.exe⏎`

Laboratório

Arquivo **linec.asm**

- Para este laboratório, o aluno utilizará o conjunto de rotinas gráficas que se encontram no arquivo **linec.asm**. O arquivo é composto pelas seguintes procedimentos:

Chamada	Parâmetros a serem passados	Modo	procedimento
Cursor	dh = linha (0-29) e dl=coluna (0-79)	Gráfico (VGA)	Posiciona cursor
caracter	al= caracter a ser escrito; cor definida na variavel cor	Gráfico (VGA)	Escreve 1 caracter
plot_xy	push x; push y; call plot_xy; (x≤639, y≤479); cor definida na variavel cor	Gráfico (VGA)	Coloca um pixel na posição (x,y)
Circle	push xc; push yc; push r; call circle; (xc+r≤639,yc+r≤479)e(xc-r≥0,yc-r≥0) ; cor definida na variavel cor	Gráfico (VGA)	Desenha uma circunferência
full_circle	push xc; push yc; push r; call full_circle; (xc+r≤639,yc+r≤479) e (xc-r≥0,yc-r≥0); cor definida na variavel "cor"	Gráfico (VGA)	Desenha um círculo e o colore
Line	push x1; push y1; push x2; push y2; call line; (x≤639, y≤479)	Gráfico (VGA)	Desenha um segmento de reta

Laboratório

Arquivo **`linec.asm`**

- Tais procedimentos fazem uso da interrupção de software **`int 10h`** definida pela BIOS (*Basic Input Output System*). O modo VGA (*Video Graphics Array*) permite uma resolução de 640×480 pontos em modo gráfico, cada ponto com até 16 cores. Permite igualmente 256 cores com uma definição de 320×200 pontos.

- Observe que nas chamadas a estes procedimentos, os parâmetros desejados (posição, cor, etc.) são passados pela pilha. Por exemplo:

Chamada ao Procedimento

```
...
;A variável "cor" recebe um valor na faixa
;[0,15].  A  ;declaração  das  cores  segue
;abaixo (declarado ;em segment dados)
  mov byte [cor], azul
  ;Empilha as coordenadas (x,y)
  push ax
  push dx

  ;Chama ao procedimento.
  call plot_xy
  ...
segment dados
  preto    equ  0
  azul     equu  1
  verde    equ  2
  cyan     equ  3
  vermelho equ 4
  magenta  equ 5
  marrom   equ 6
  branco   equ 7
  cinza    equ 8
  azul_claro  equ 9
  verde_claro equ 10
  cyan_claro  equ 11
  rosa        equ 12
  magenta_claro  equ 13
  amarelo        equ 14
  branco_intenso equ 15
```

Definição do Procedimento chamado

```
plot_xy:
;Faz BP apontar para o topo da pilha, antes
;de salvar o ;contexto
  push bp
  mov bp,sp
;Salvando o contexto, empilhando registradores
  pushf
  push ax
  push bx
  push cx
  push dx
  push si
  push di
;Preparando para chamar a int 10h
;cor é uma variável global
  mov ah,0ch
  mov al,[cor]
  mov bh,0
  mov dx,479
;Aqui,  desempilha-se  os  parâmetros  passados
;pela pilha, ;no caso 2 words (16 bits cada)
  sub dx,[bp+4]
  mov cx,[bp+6]
  int 10h
; recupera-se o contexto
  pop di
  pop si
  pop dx
  pop cx
  pop bx
  pop ax
  popf
  pop bp
;O "4" como parâmetro de "ret" faz um flush na
pilha,  ;desempilhando  os  parâmetros  ;passados
pela pilha na ;chamada.
  ret
```

Salvar contexto (Prologo)

Tarefa

Recuperar contexto (Epilogo)

Laboratório

1.

- Estude a passagem de parâmetros, conforme mostrado acima. Faça um diagrama de como a passagem se processa na pilha, mostrando **`SP`** e **`BP`**. Quando ocorre um **`call`** (do tipo **`near`**), observe que é empilhado o registrador **`IP`**. Para o exemplo da tabela acima, tente explicar as instruções **`sub dx,[bp+4]`** e **`mov cx,[bp+6]`**. Veja outros exemplos para as rotinas **`line`** e **`full_circle`**.

Laboratório

2.

- Gere o código executável, usando NASM/FREELINK, e veja o comportamento do programa (são gerados vários segmentos de reta e círculos, formando um desenho). Modifique os parâmetros e veja o comportamento.

Laboratório

3.

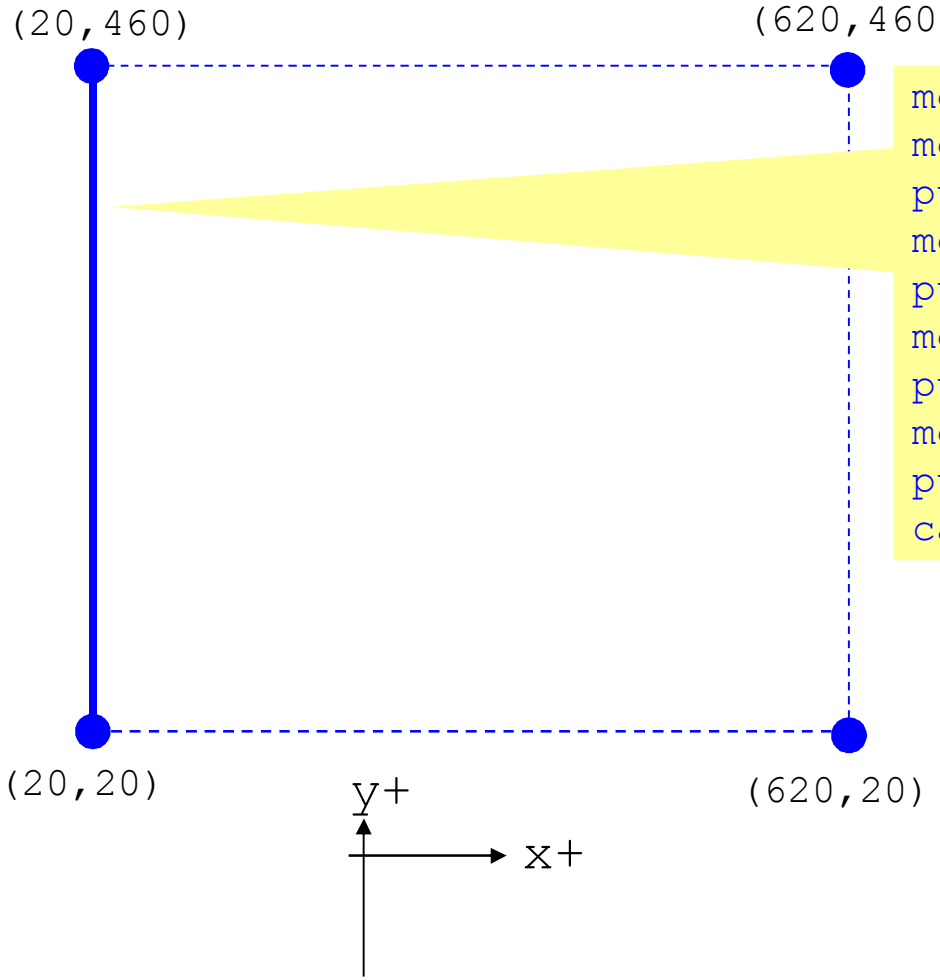
- Faça agora um programa que desenhe um quadrado de 640×480 pixels, borda branca e fundo preto. Em processamento de imagens/vídeo, o ponto (0,0) é o ponto superior esquerdo da tela. Depois, acrescente um círculo, de cor vermelha e raio = 10, no meio da tela.

- Usar como referencia o programa: **`lincr001_template.asm`**
  - **OBS**: Trocar o nome do arquivo para **`lincr001.asm`** (Os nomes dos arquivos devem ter no máximo oito caracteres sem considerar a extensão ASM)

Laboratório

3. DICA: desenho de linhas

- Disenho da linha lateral esquerda



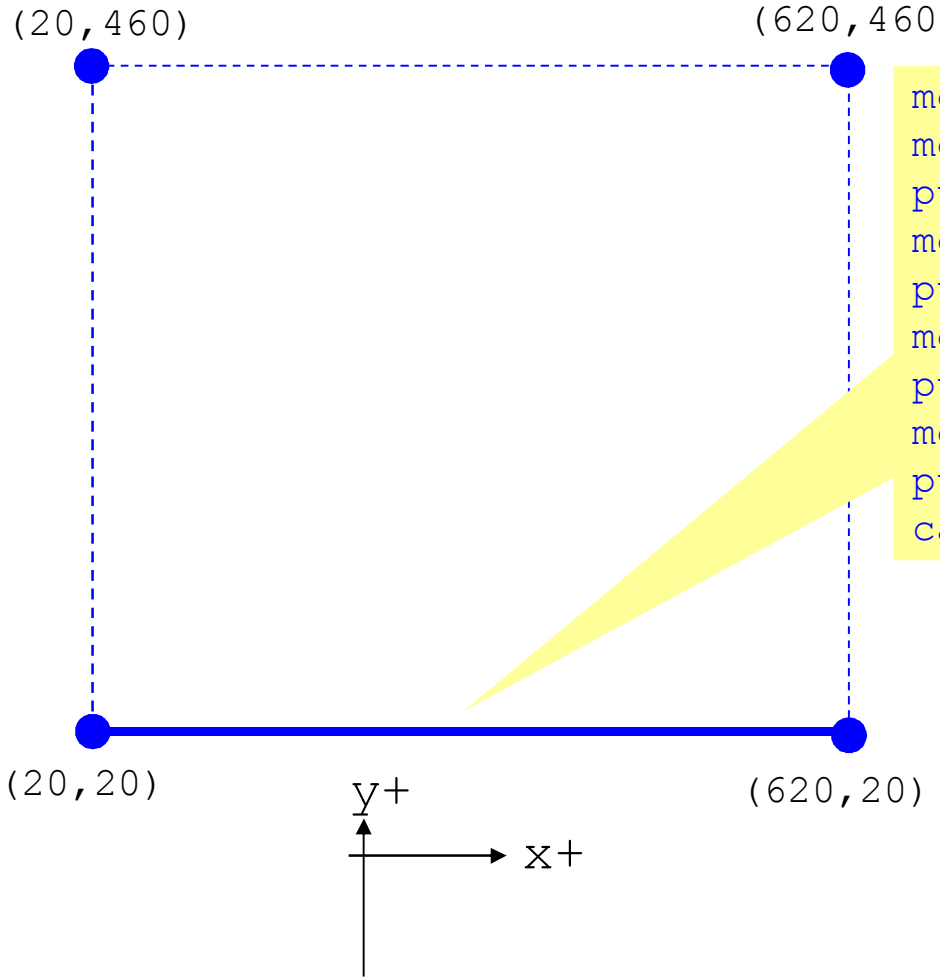
lincr001.asm

```
mov   byte[cor],branco_intenso
mov   ax,20     ;x1<-20
push  ax
mov   ax,20     ;y1<-20
push  ax
mov   ax,20     ;x2<-20
push  ax
mov   ax,460    ;y2<-460
push  ax
call  line ; line(x1,y1,x2,y2)
```

Laboratório

3. DICA: desenho de linhas

- Disenho da linha inferior



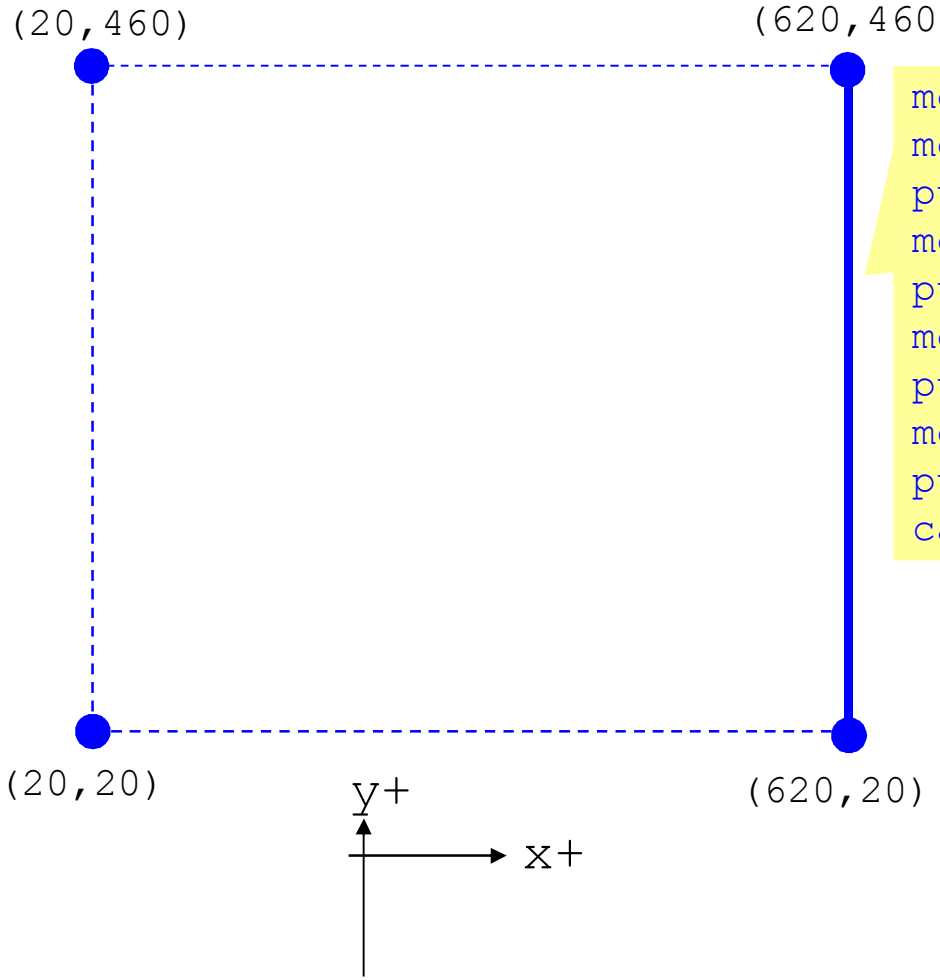
lincr001.asm

```
mov   byte[cor],branco_intenso
mov   ax,20    ;x1<-20
push  ax
mov   ax,20    ;y1<-20
push  ax
mov   ax,620   ;x2<-620
push  ax
mov   ax,20    ;y2<-20
push  ax
call  line ; line(x1,y1,x2,y2)
```

Laboratório

3. DICA: desenho de linhas

- Disenho da linha lateral dereita



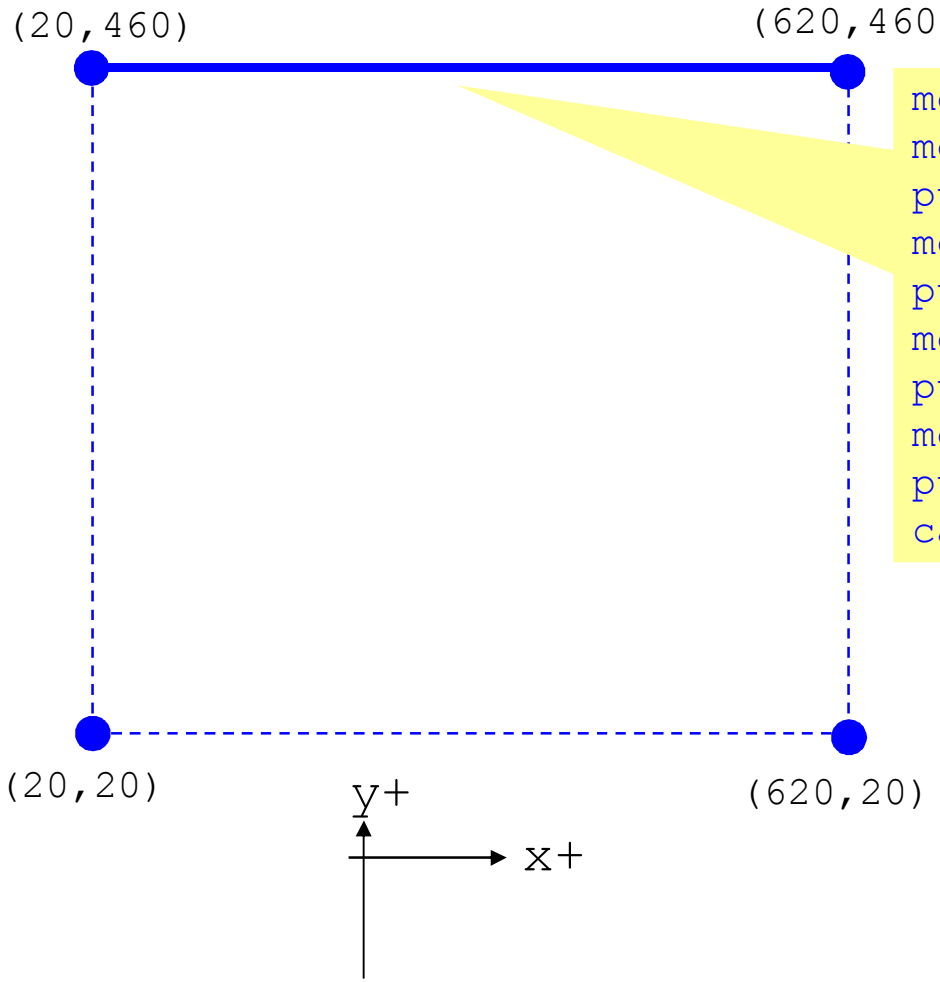
lincr001.asm

```
mov   byte[cor],branco_intenso
mov   ax,620   ;x1<-620
push  ax
mov   ax,20    ;y1<-20
push  ax
mov   ax,620   ;x2<-620
push  ax
mov   ax,460   ;y2<-460
push  ax
call  line ; line(x1,y1,x2,y2)
```

Laboratório

3. DICA: desenho de linhas

- Disenho da linha superior



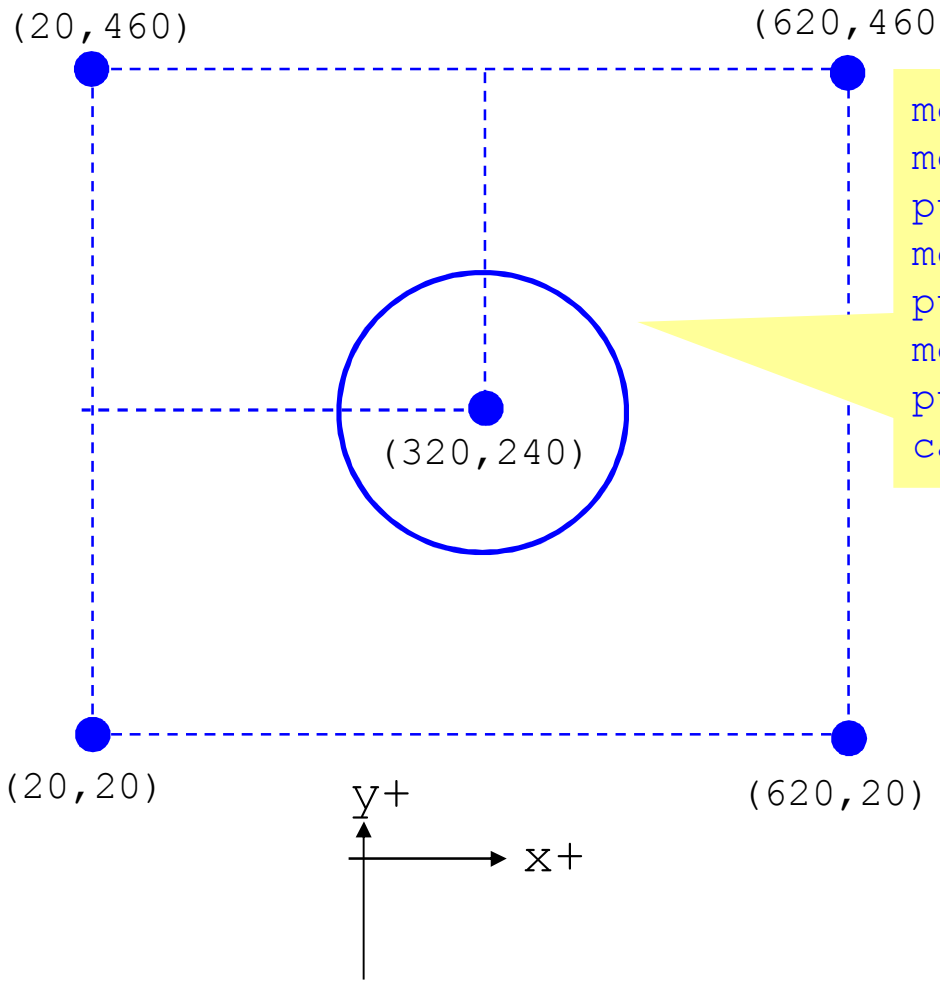
lincr001.asm

```
mov   byte[cor],branco_intenso
mov   ax,620   ;x1<-620
push  ax
mov   ax,460   ;y1<-460
push  ax
mov   ax,20    ;x2<-20
push  ax
mov   ax,460   ;y2<-460
push  ax
call  line ; line(x1,y1,x2,y2)
```

Laboratório

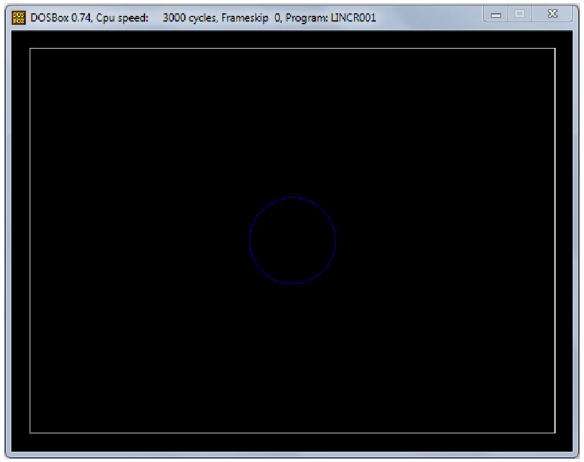
3. DICA: desenho do circulo

- Disenho do circulo



lincr001.asm

```
mov  byte[cor],azul
mov  ax,320 ;xc<-320
push ax
mov  ax,240 ;yc<-240
push ax
mov  ax,50  ;r<-50
push ax
call circle ; circle(xc,yc,r)
```



Laboratório

4.

- Agora, faça uma animação com o círculo vermelho (bola vermelha), de modo que, logo no início da animação, a bola se desloque a 45°, para cima, pela tela e ao se chocar com as laterais, esta deve desviar de trajetória da mesma forma que um raio de luz o faria ao ser refletido por uma superfície reflexiva especular.
- Para fazer o tempo de animação, use a procedimento **`delay`**, ajustando seus parâmetros para o seu programa.

```
delay:  ; Esteja atento pois talvez seja importante salvar  contexto
        ; (no caso, CX, o que NÃO foi feito aqui).
        mov cx, word [velocidade]; Carrega "velocidade" em cx (contador para loop)
del2:
        push cx                 ; Coloca cx na pilha para usa-lo em outro loop
        mov  cx, 0800h          ; Teste modificando este valor
del1:
        loop del1               ; No loop del1, cx  é decrementado até que volte a ser zero
        pop  cx                 ; Recupera cx da pilha
        loop del2               ; No loop del2, cx é decrementado até que seja zero
        ret
```

Laboratório

4.

- Observe que, no início, deve-se escolher o modo gráfico do vídeo. Portanto, é necessário armazenar o modo inicial. Isto é feito usando **AH=0Fh** e chamando-se **INT 10H**. O valor de retorno em **AL** deve ser guardado em uma variável, por exemplo **modo_anterior**, para ser restaurado ao sair do programa. Depois, para por no modo VGA, usa-se **AX=12H** e chama-se **INT 10h**. Ao sair, com **AL=[modo anterior]**, ao se fazer **AH=0Fh** e chamando-se **INT 10H**, restaura-se o modo de vídeo original. Então, as partes inicial e final de seu código devem ser, como se segue:

```
segment code
..start:
 mov ax,data
 mov ds,ax
 mov ax,stack
 mov ss,ax
 mov sp,stacktop
; salvar modo corrente de vídeo
 mov ah,0Fh
 int 10h
 mov [modo_anterior],al
; alterar modo de video para gráfico
; 640x480 16 cores
 mov al,12h
 mov ah,0
 int 10h
; Aqui entra seu código (loop infinito)
; para fazer a animação.
;--------------------------------------

;--------------------------------------
;Para sair, faça
sai:
 mov ah,0 ; set video mode
 mov al,[modo_anterior] ; recupera o modo
                        ; anterior
 int 10h
 mov ax,4c00h
 int 21h
```

Laboratório

4.

- Como seu programa será um *loop* infinito, é necessário colocar uma forma de sair do programa. Assim, dentro de seu *loop* infinito, o seguinte trecho de código, baseado na **`int 21h`**, deve aparecer:

```
  mov  ah,0bh
  int  21          ; Le buffer de teclado
  cmp  al,0        ; Verifica se AL foi 0
  jne  adelante    ; se AL != 0 então há algum caractere na STDIN
  jmp segue        ; se AL  = 0 então nada foi digitado e a animação do jogo deve continuar
adelante:
  mov ah, 08H      ; Ler caractere da STDIN
  int 21H
  cmp al, 's'      ; Verifica se AL foi 's'
  jne adianta      ; se AL != 's' então a finaliza é adiantada
  jmp sai          ; se AL  = 's' então finaliza o programa
segue:             ; a animação deve seguir a partir daqui
```

- Observe que para **`AH=0bh`**, a **`INT 21H`** apenas averigua se o *buffer* do teclado foi carregado com algum valor de tecla (e não fica esperando pela digitação da tecla). Se **`AL = 0`** então nada foi digitado; se **`AL =255`** então há algum caractere no *buffer* que precisa ser lido. Caso haja caractere, fazendo-se **`AH=8`** e chamando-se a **`int 21h`** o resultado da tecla digitada aparece em **`AL`**. Observe que **`AH=8`** não mostra (ecoa) o caractere na tela.

Laboratório

4.

- Lembre-se que saltos condicionais (**JNE, JC, JE, JZ, JNZ**, etc) deslocam o **IP** na faixa máxima **[-128, 127]**. Para saltos maiores que esta faixa, a versão do NASM16 que usamos no laboratório, gerara um erro de compilação. Porém, é comum nos programas acontecerem saltos que extrapolam esta faixa. Portanto, faça conforme o exemplo:

O que gostaria de fazer (o rótulo " igual" indica uma posição de memória que extrapola a faixa [-128, 127]. Situação onde "igual" está muito longe.

```
.
.
.
cmp ax,FFFFh  ; Verifica se ax foi FFFFh
je  igual     ; ax == FFFFh
jmp diferente ; ax != FFFFh
.
.
.
igual:
```

Mas deu erro de estouro de salto. Então, deve-se fazer (OBSERVE QUE USA-SE A **INSTRUÇÃO JMP**, QUE NÃO SOFRE DESTE PROBLEMA).

```
.
.
.
cmp ax, FFFFh ; Verifica se ax foi FFFFh
jne diferente ; ax != FFFFh
jmp igual     ; ax == FFFFh
diferente:
; aqui vêm as instruções que tratam
; quando for diferente
.
.
.
igual:
```

Laboratório

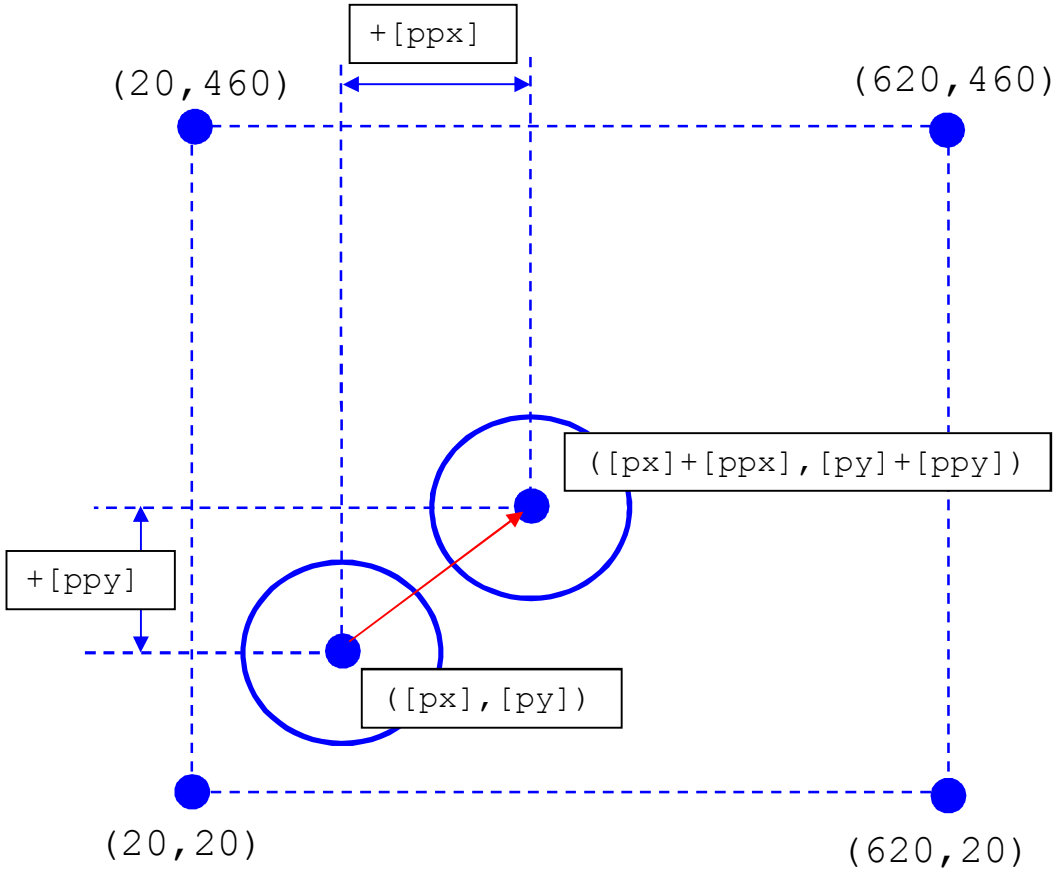
4.

- Usar como referencia o programa: **`linecev_template.asm`**
  - **OBS**: Trocar o nome do arquivo para **`linecev.asm`** (Os nomes dos arquivos devem ter no máximo oito caracteres sem considerar a extensão ASM)

Laboratório

4. DICA: Algoritmo para desenhar o deslocamento da bola

- Inicio



Condição:

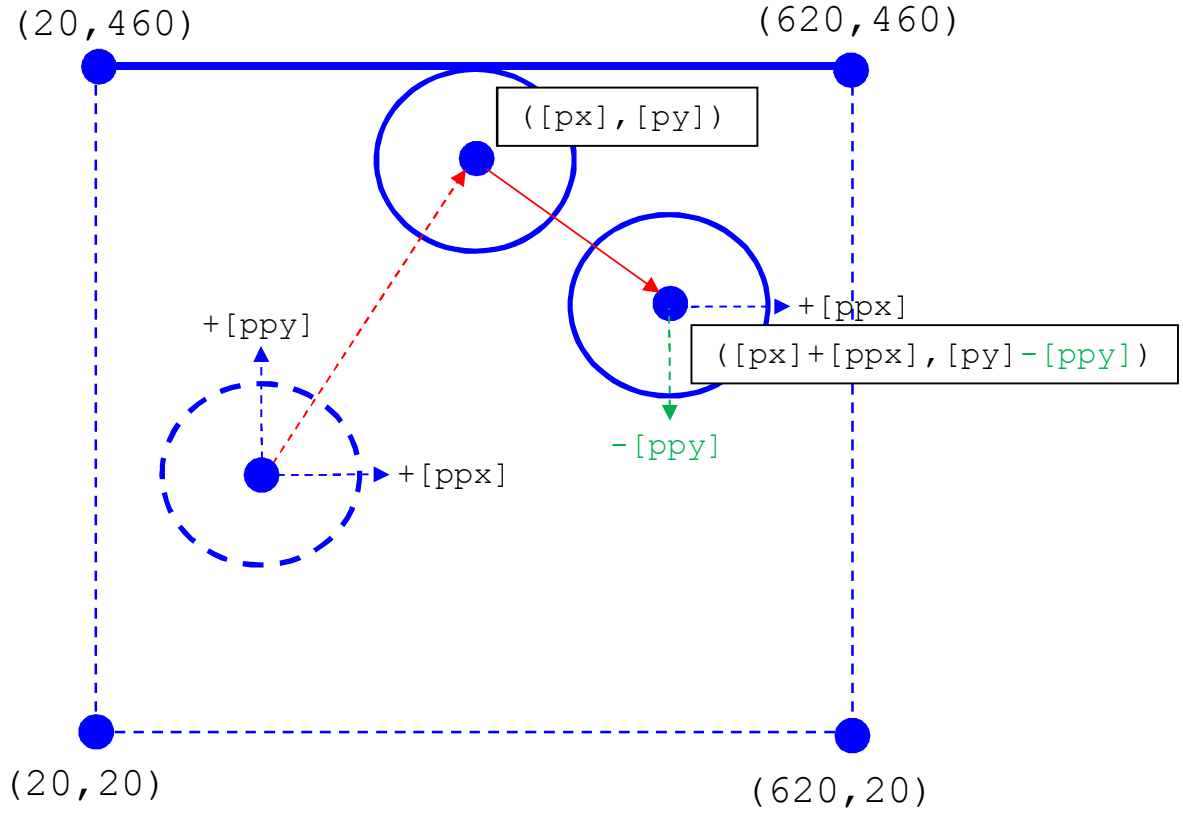
[px] ← [px]+[ppx]

[py] ← [py]+[ppy]

Laboratório

4. DICA: Algoritmo para desenhar o deslocamento da bola

- Colisão na linha superior



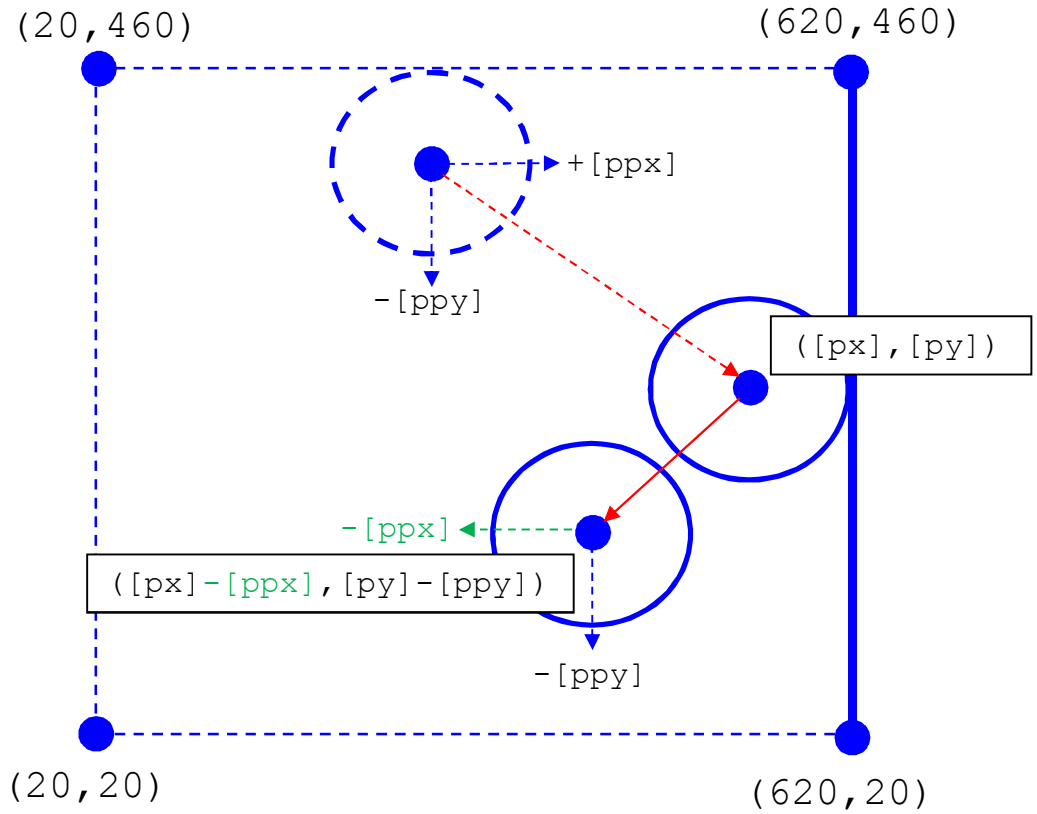
Condição:

```
IF([py]>=460)THEN
   [py] ←[py]-[ppy]
```

Laboratório

4. DICA: Algoritmo para desenhar o deslocamento da bola

- Colisão na linha direita



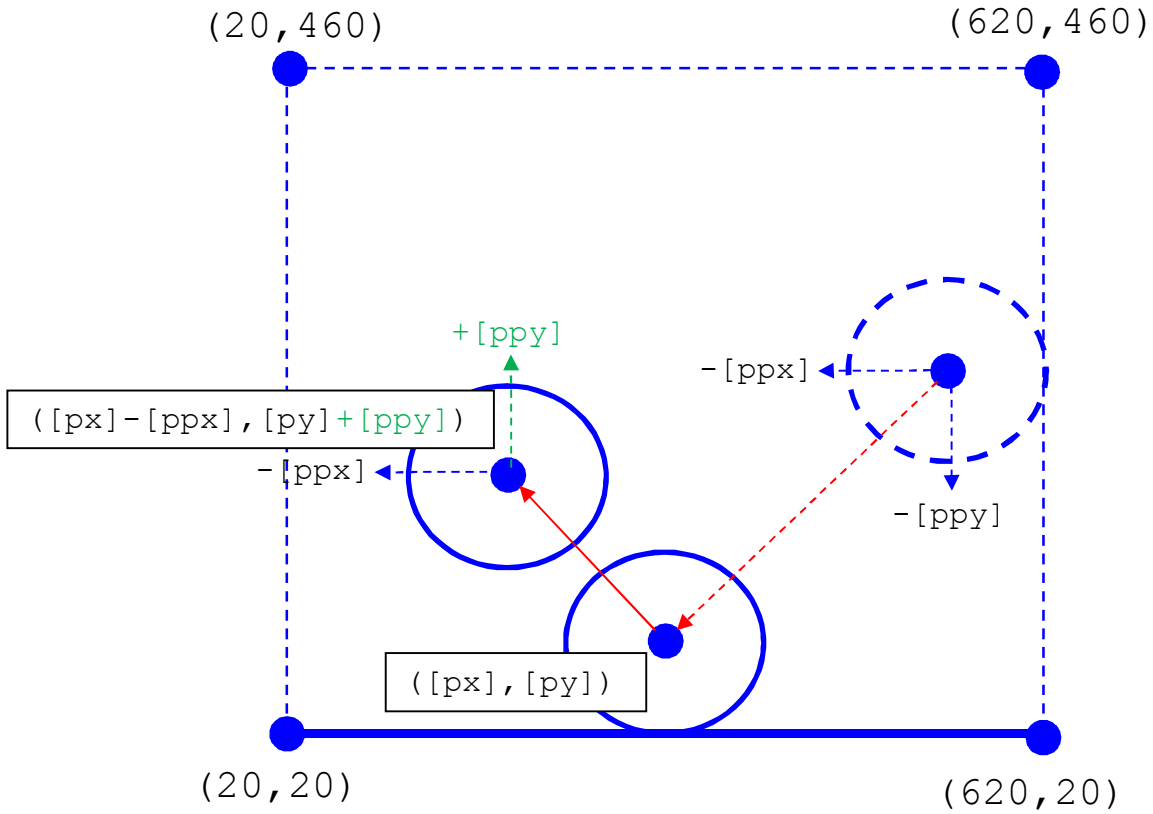
Condição:

```
IF([px]>=620)THEN
   [px] ←[px]-[ppx]
```

Laboratório

4. DICA: Algoritmo para desenhar o deslocamento da bola

- Colisão na linha inferior



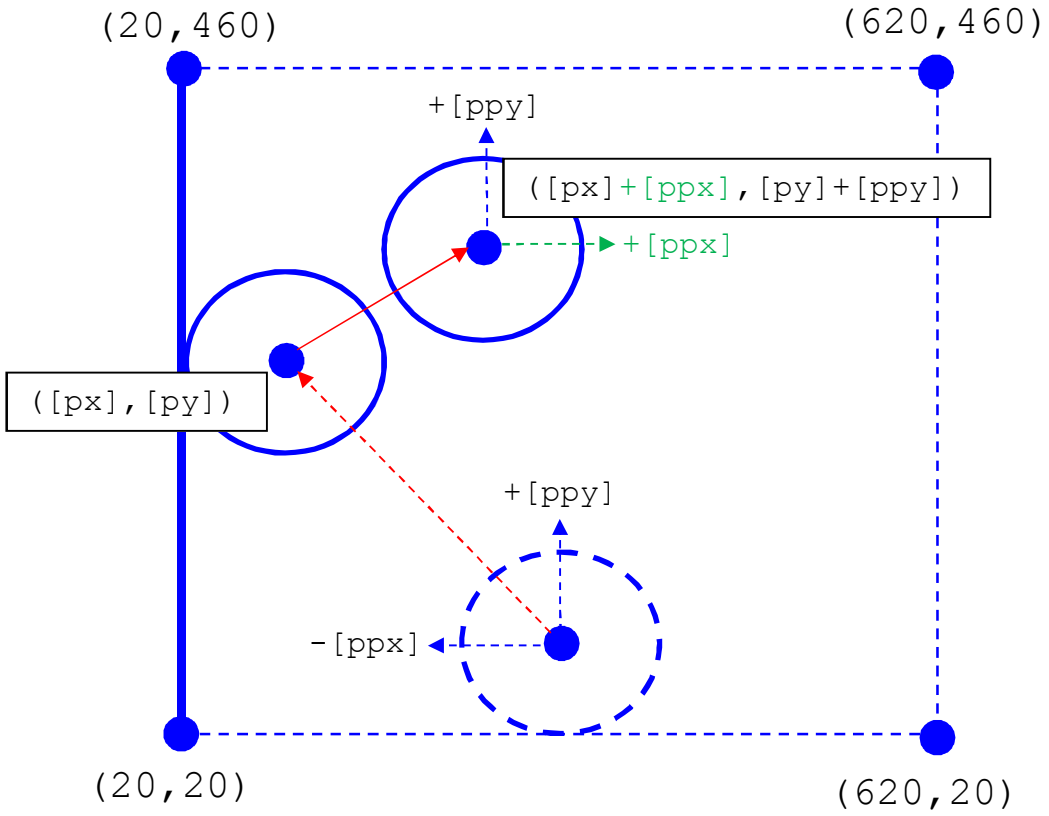
Condição:

```
IF([py]<=20)THEN
   [py] ←[py]+[ppy]
```

Laboratório

4. DICA: Algoritmo para desenhar o deslocamento da bola

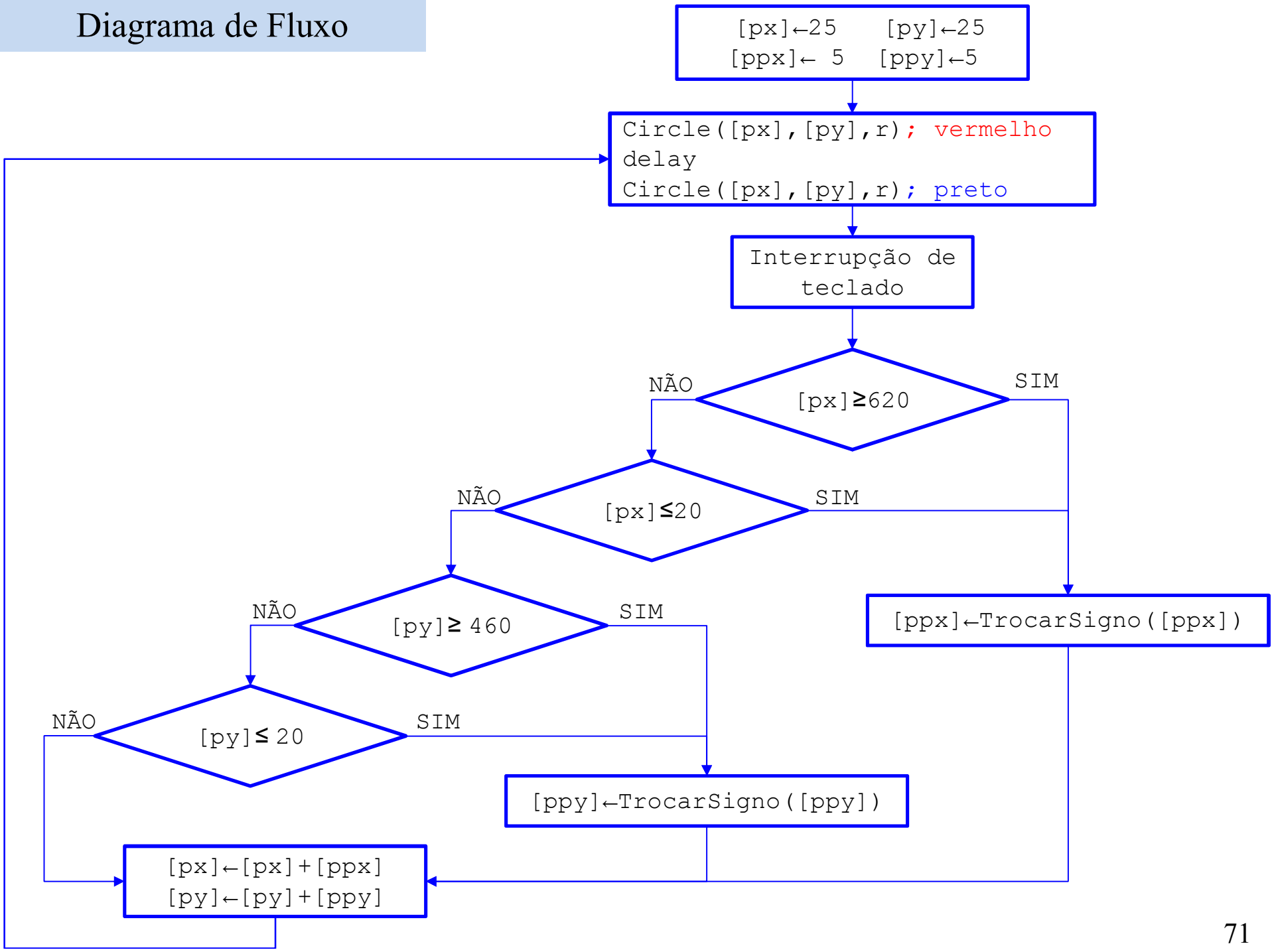
- Colisão na linha esquerda



Condição:

```
IF([px]<=20)THEN
   [px] ←[px]+[ppx]
```

Diagrama de Fluxo



Laboratório

4. DICA: Converter números positivos a negativos

- Para converter um número positivo para negativo use o seguinte algoritmo baseado no COMPLEMENTO A DOIS:
  1. Inverta todos os bits do número com a instrução **NOT**.
  2. Adicione 1 ao resultado com a instrução **INC**.

- No exemplo a seguir faremos a conversão de `-5` para `+5`:

posneg01.asm

```
segment code
..start:
...
; --------------------CODIGO DO PROGRAMA-----------------------
  mov ax, -5
  not ax ; paso 1
  inc ax ; paso 2
; -----------------------SAIDA DO PROGRAMA---------------------
...
```

- Para maior informação ver: http://stackoverflow.com/questions/4534503/how-to-convert-a-positive-number-to-negative-in-assembly

Laboratório

4. DICA: Converter números positivos

- Para converter um número positi
  baseado no COMPLEMENTO A I
  1. Inverta todos os bits do núme
  2. Adicione 1 ao resultado com

```
+5 ⇒ 0005h
+4 ⇒ 0004h
+3 ⇒ 0003h
+2 ⇒ 0002h
+1 ⇒ 0001h
 0 ⇒ 0000h
-1 ⇒ FFFFh (representa  o número -1)
-2 ⇒ FFFEh
-3 ⇒ FFFDh
-4 ⇒ FFFCh
-5 ⇒ FFFBh
```

- No exemplo a seguir faremos a co

```
segment code
..start:
...
; --------------------COD
  mov ax, -5
  not ax ; paso 1
  inc ax ; paso 2
; ----------------------
...
```

```
DOSBox 0.74, Cpu speed:   3000 cycles, Frameskip  0, Program:   DEBUG
AX=076C  BX=0000  CX=0017  DX=0000  SP=0100  BP=0000  SI=0000  DI=0000
DS=076B  ES=075A  SS=076C  CS=076A  IP=000D   NV UP EI PL NZ NA PO NC
076A:000D B8FBFF        MOV     AX,FFFB
-t

AX=FFFB  BX=0000  CX=0017  DX=0000  SP=0100  BP=0000  SI=0000  DI=0000
DS=076B  ES=075A  SS=076C  CS=076A  IP=0010   NV UP EI PL NZ NA PO NC
076A:0010 F7D0          NOT     AX
-t

AX=0004  BX=0000  CX=0017  DX=0000  SP=0100  BP=0000  SI=0000  DI=0000
DS=076B  ES=075A  SS=076C  CS=076A  IP=0012   NV UP EI PL NZ NA PO NC
076A:0012 40            INC     AX
-t

AX=0005  BX=0000  CX=0017  DX=0000  SP=0100  BP=0000  SI=0000  DI=0000
DS=076B  ES=075A  SS=076C  CS=076A  IP=0013   NV UP EI PL NZ NA PE NC
076A:0013 B44C          MOV     AH,4C
-
```

- Para maior inform
  to-convert-a-positiv

Laboratório

4. DICA: Instrução **IF - THEN**

- É necessario usar INSTRUÇÕES DE SALTO:
  - Tomando em conta que serão usados números sinalizados devem-se usar as instruções de salto JGE (OP1 ≥ OP2) e JLE (OP1 ≤ OP2).
- No exemplo é implementado uma sentencia **IF-THEN** aninhado:

exif002.asm

```
segment code
...
; -----CODIGO DO PROGRAMA---------
mov ax,250
;IF(AX menor ou igual a 100)
cmp ax, 100 ;ax <= 100
jle incAX
;ELSEIF(AX maior ou igual a 200)
cmp ax, 200 ;ax >= 200
jge decAX
;THEN1
incAX: add ax,2
       mov dx,stringInc
       jmp fim
;THEN2
decAX: sub ax,2
       mov dx,stringDec
       jmp fim

fim: mov ah,0x9
     int 21h
...
segment data
; ----DEF. VAR,CONST E ALOCACAO------
stringInc db 'incrementou',13,10,'$'
stringDec db 'decrementou',13,10,'$'
```