Programação de Software Básico – Linguagem Assembly – Procedures

MATA49

Prof. Babacar Mane

2021.1 – Aula 6

Programação de Software Básico
Procedures

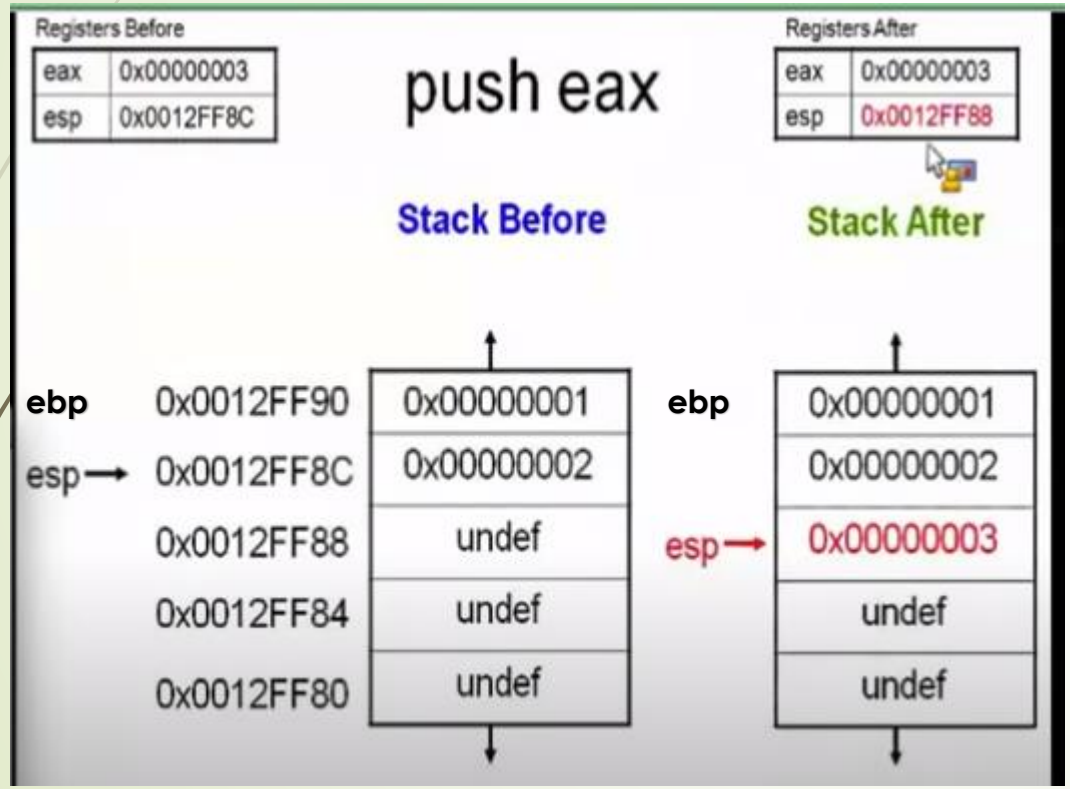
- Uso da pilha (refresh):
- Armazenamento temporário de dados
- Transferência de controle
- Passagem de parâmetros (chamada de procedimento)

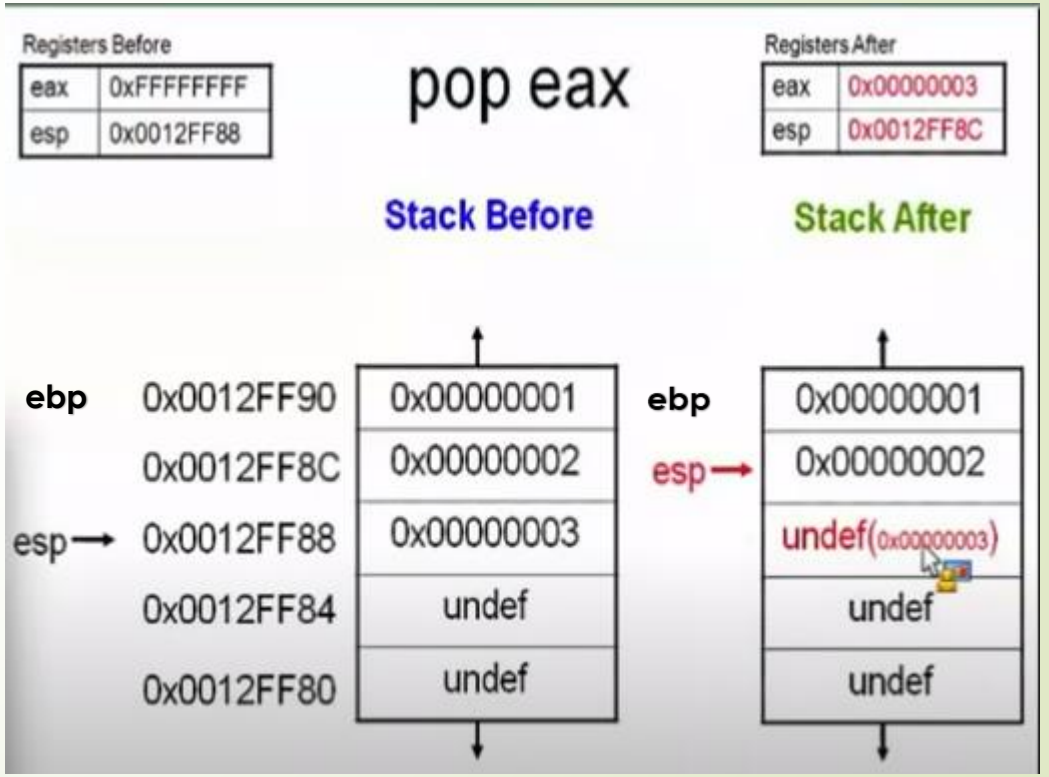
Programação de Software Básico
Procedures

- Uso da pilha (refresh):
- Armazenamento temporário de dados

```
;salva BX e CX na pilha
Push BX
Push CX
....
<< BX e CX podem ser usados agora>>
....
;recupra o valor de BX e CX
Pop CX
Pop BX
```

Programação de Software Básico
Procedures - (refresh)





Programação de Software Básico
Procedures - (refresh)

Tabela de operações 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.

Programação de Software Básico
Procedures

- Uso da pilha (refresh) :

  Transferência de controle

- Em chamadas de procedimentos e interrupções, o endereço de retorno é armazenado na pilha

  Passagem de Parâmetros

- A pilha é extensivamente usada para a passagem de parâmetros em chamadas de procedimentos

Programação de Software Básico
Procedures

- Duas instruções são utilizadas para manipular uma *procedure*: call e ret.

- A instrução call invoca uma *procedure* proc-name (_printf, _scanf):

  - section .extern _printf
    _scanf

Programação de Software Básico
Procedures

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

Programação de Software Básico
Procedures

- Como o controle do programa é transferido ?

  - Após a instrução call do main ter sido buscada o registrador EIP aponta para a próxima instrução a ser executada. EIP = 00000007h para a primeira instrução, e EIP = 0000002Dh para a segunda instrução.

  - Esta instrução deve ser executada após a execução da procedure sum

  - O processador empurra o conteúdo de EIP na pilha

Programação de Software Básico
Procedures

- Como o controle do programa é transferido ?

  - Para transferir o controle para a primeira instrução da procedure sum, teria que ser carregado no registrador EIP o offset (endereço) da instrução push EBP. (0000001Dh)

  - SP := SP - 2; reserva espaço na pilha da memória
  - (SS:SP) := IP; empilha o endereço atual de IP

Programação de Software Básico
Procedures

- A instrução ret

  - A instrução ret é utilizada para transferir o controle da procedure chamada para a procedure chamadora.
  - Transfere o controle para a instrução que sucede o call. No exemplo, mov ebx, eax
  - Como o processador sabe onde a instrução está localizada ?
  - Quando a instrução ret é executada o endereço que foi armazenado na pilha é recuperado

Programação de Software Básico
Procedures

- A instrução ret

  - As operações que ocorrem após a execução da instrução ret:

    IP := (SS:SP); desempilha o endereço de retorno

    SP := SP + 2; atualiza o topo da pilha

Programação de Software Básico
Procedures

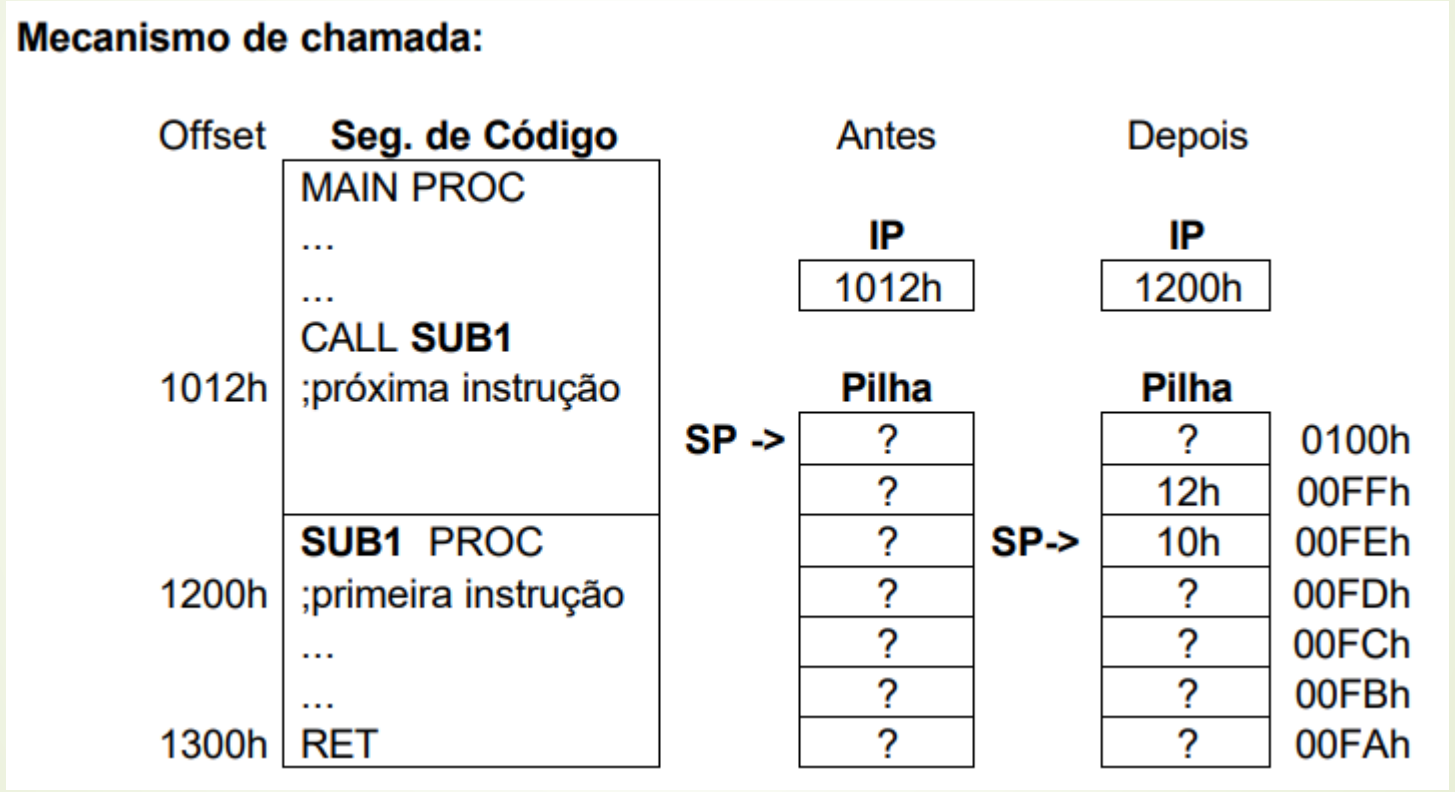
- Exemplo:

```
 7:   .DATA
 8:   prompt_msg1   DB    "Please input the first number: ",0
 9:   prompt_msg2   DB    "Please input the second number: ",0
10:   sum_msg       DB    "The sum is ",0
11:
12:   .CODE
13:           .STARTUP
14:           PutStr  prompt_msg1     ; request first number
15:           GetInt  CX              ; CX = first number
16:
17:           PutStr  prompt_msg2     ; request second number
18:           GetInt  DX              ; DX = second number
19:
20:           call    sum             ; returns sum in AX
21:           PutStr  sum_msg         ; display sum
22:           PutInt  AX
23:           nwln
```

```
27:   ;-----------------------------------------------------
28:   ;Procedure sum receives two integers in CX and DX.
29:   ;The sum of the two integers is returned in AX.
30:   ;-----------------------------------------------------
31:   sum:
32:           mov     AX,CX           ; sum = first number
33:           add     AX,DX           ; sum = sum + second number
34:           ret
```

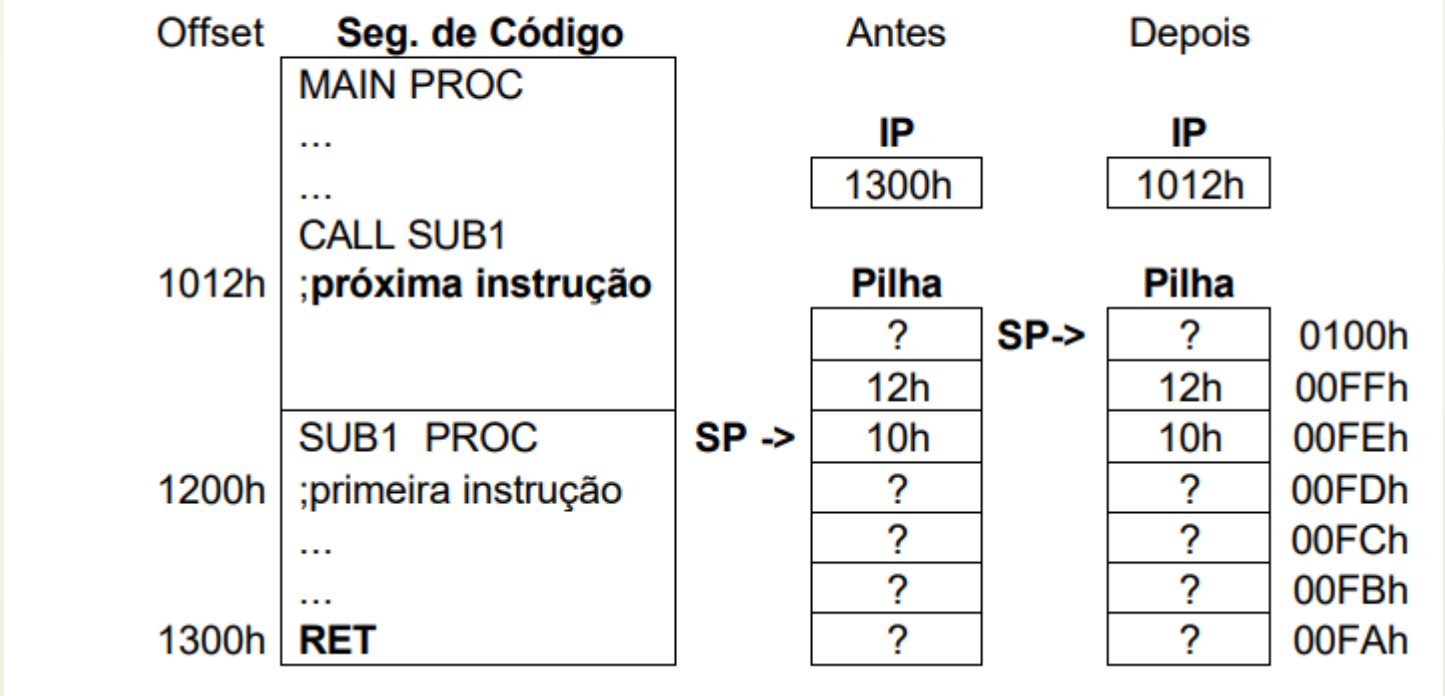
Programação de Software Básico
Procedures

- **Mecanismo de chamada:**



Programação de Software Básico
Procedures

- **Mecanismo de retorno:**



Programação de Software Básico
Procedures

- Passagem de parâmetros:
  - Passagem de parâmetro na linguagem assembly é diferente e mais complicado do que nas linguagens de alto nível.
  - Em assembly:
    - Deve-se primeiro colocar todos os parâmetros em uma área de memória mutuamente acessível
    - Então chamar o procedimento
  - Existem dois métodos para invocar as procedures: o método registrador e o método pilha

Programação de Software Básico
Procedures

- Passagem de parâmetros: Método por registrador
  - Neste método, a procedure chamadora coloca os parâmetros necessários nos registradores de propósito geral antes de invocar uma procedure
  - Vantagens do método:
    - Conveniente e mais fácil quando se passa um número pequeno de argumentos.
    - Método mais rápido por que todos os argumentos estão em registradores.

Programação de Software Básico
Procedures

- Passagem de parâmetros: Método por registrador
  - Exemplos:
    - Uma rotina simples de soma
    - Uma rotina de cálculo de tamanho de strings

Programação de Software Básico
Procedures

- Passagem de parâmetros: Método por registrador
  - Desvantagens do método:
    - Poucos parâmetros podem ser passados
    - Os registradores de propósito geral normalmente são utilizados pelo chamador para outras propostas. Provavelmente se utilizará a pilha para armazenar os seus valores

Programação de Software Básico
Procedures

- Passagem de parâmetros: Método de pilha

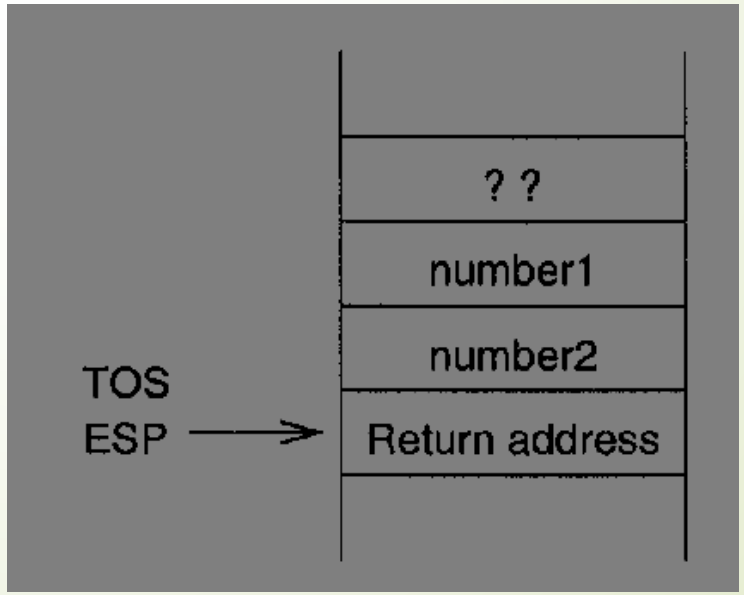
  - Todos os argumentos requeridos pela procedure são empurrados na pilha antes da sua chamada.

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

Programação de Software Básico
Procedures

- Passagem de parâmetros: Método de pilha

  - Após a chamada call para a procedure sum teremos o seguinte estado de pilha:

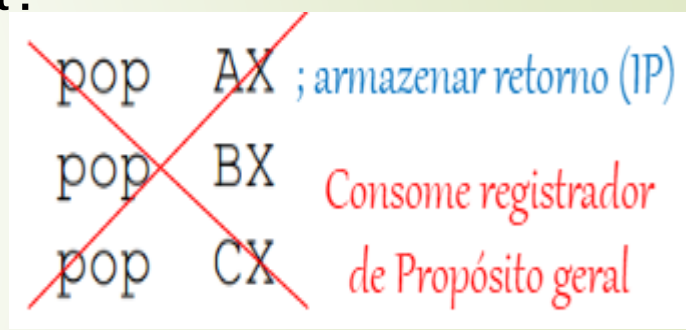


Programação de Software Básico
Procedures

- Passagem de parâmetros: Método de pilha
- Como ter acesso aos parâmetros na pilha?
  - Podemos usar:

    add SP, 2

    mov BX, [SP]



  - Problema: sujeito a erros
    - É necessário lembrar de atualizar SP para apontar para o valor de retorno do procedimento antes do seu término
    - Se teremos que retirar os parâmetros da pilha e armazená-los nos registradores como retirar 10 parâmetros, por exemplo?
    - Existe uma alternativa melhor?
      - Usar o registrador BP para acessar parâmetros na pilha;

Programação de Software Básico
Procedures

- Passagem de parâmetros: Método de pilha
  - Método preferido de acesso aos parâmetros
    - Para acessar o número 2 do exemplo anterior:

```
pop BP
mov BP, SP
mov BX,[BP+2]
```

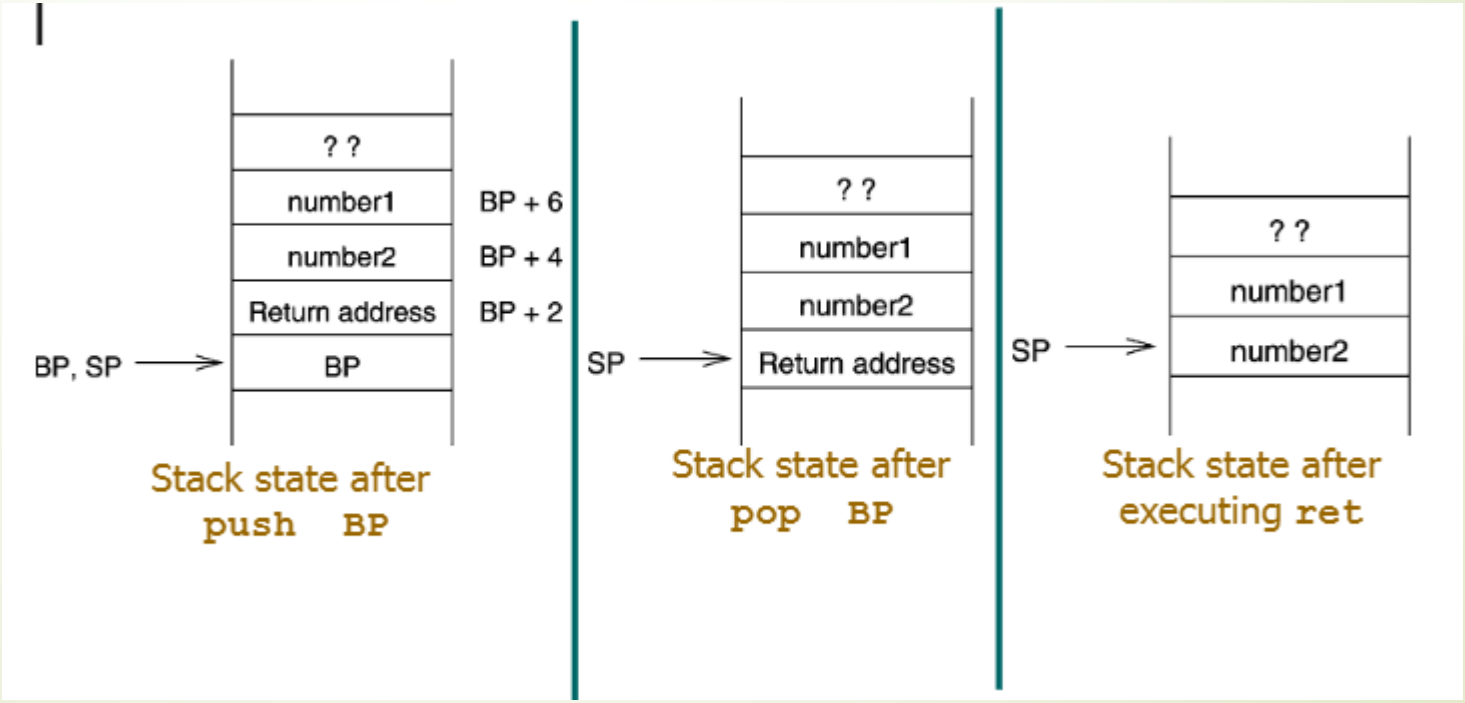
Problema: o conteúdo de BP foi perdido!

- Necessário preservar o conteúdo de BP
- Usado nas operações em vários procedimentos (atenção: valor do deslocamento dos parâmetros é alterado)

```
pop BP
mov BP, SP
```

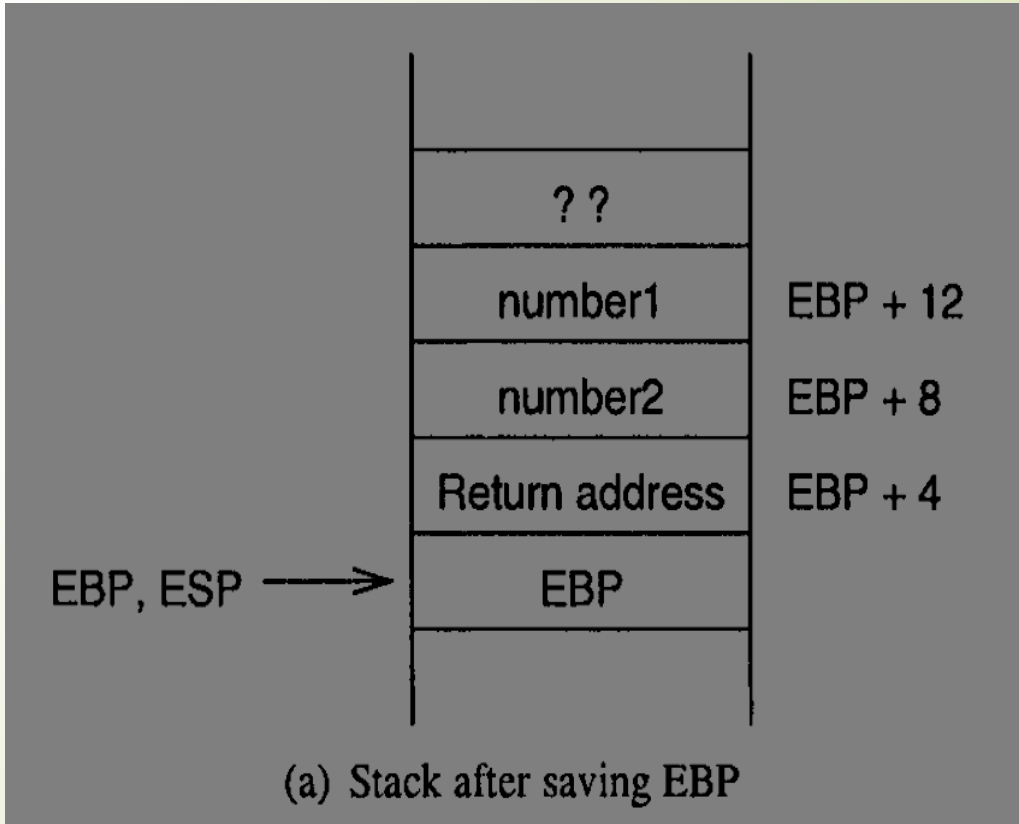
Programação de Software Básico
Procedures

- Passagem de parâmetros: Método de pilha
  - Método preferido de acesso aos parâmetros



Programação de Software Básico
Procedures

- Passagem de parâmetros: Método de pilha
  - Pilha após empurrar EBP



(a) Stack after saving EBP

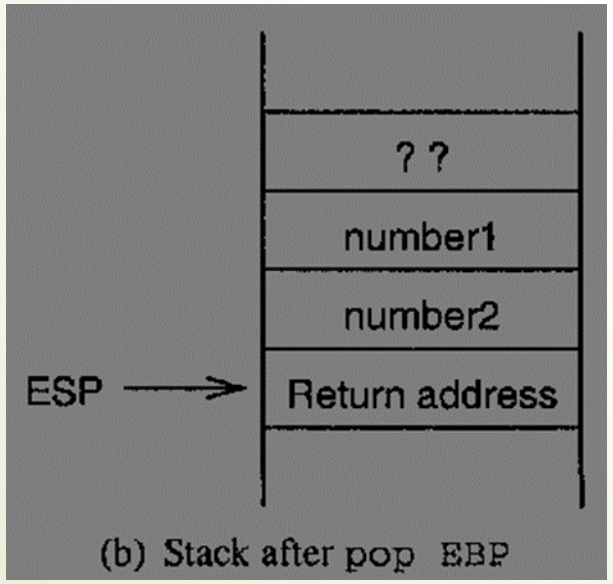
Programação de Software Básico
Procedures

- Passagem de parâmetros: Método de pilha
  - O frame da Pilha (*Stack Frame*)
    - As informações armazenadas na pilhas tais como parâmetros, endereço de retorno e o EBP são denominados frame da pilha
    - O frame da pilha consiste também de variáveis locais se a procedure utilizá-las
    - O EBP é o ponteiro do frame. Conhecendo o ponteiro EBP nós podemos acessar todos os elementos do frame

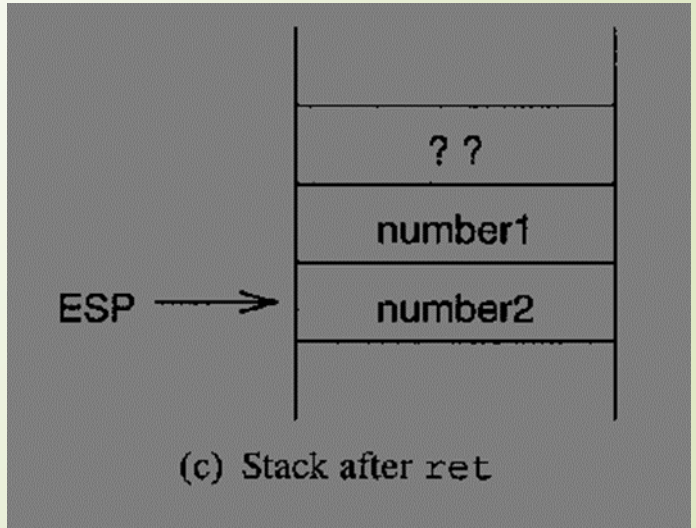
Programação de Software Básico
Procedures

- Passagem de parâmetros: Método de pilha
  - O frame da Pilha
    - Antes de retornar da procedure, o EBP precisa ser restaurado:

```
pop     EBP
```



(b) Stack after pop EBP



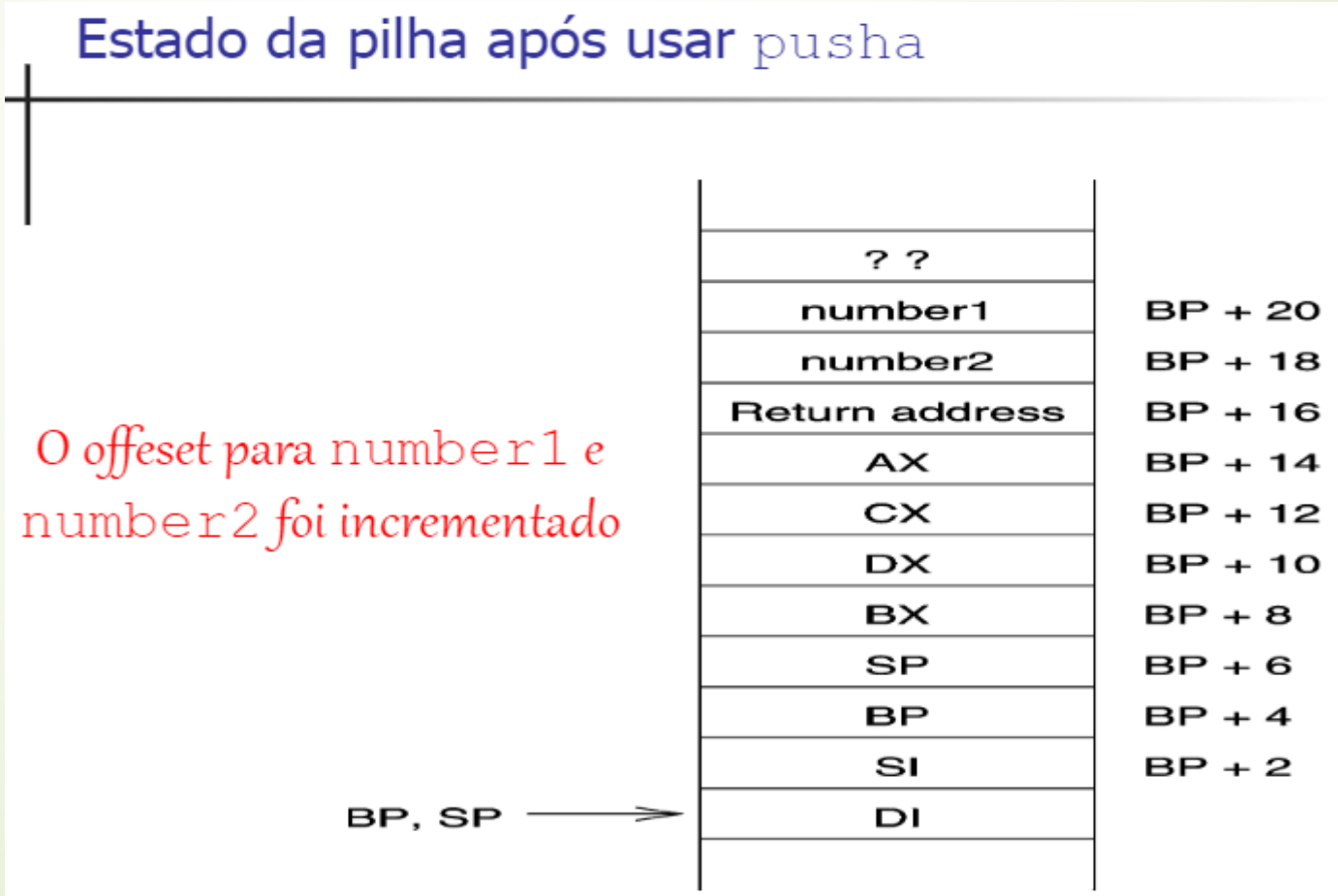
(c) Stack after ret

Programação de Software Básico
Procedures

- Passagem de parâmetros: Método de pilha
  - Como preservar o estado para cada chamada de procedimento?
    - Pilha
    - Salvar todos os registradores – pusha
      - leva 5 clocks para salvar todos os 8 registradores

Programação de Software Básico
Procedures

- **Passagem de parâmetros:** Método de pilha



Programação de Software Básico
Procedures

- Passagem de parâmetros: Método de pilha
  - Instruções para Stack Frame
    - Instrução ENTER
    - Facilita a alocação de stack frame

enter bytes, level

bytes = espaço de armazenamento local

level = nível de aninhamento (nós usamos 0)

Exemplo

```
enter XX, 0
```

Equivalente a

```
push BP
mov BP, SP
sub SP, XX
```

Programação de Software Básico
Procedures

- Passagem de parâmetros: Método de pilha
  - Instruções para Stack Frame
    - Instrução LEAVE

      Libera stack frame leave
    - Não tem operandos

Equivalente a

```
mov SP, BP
pop BP
```

Programação de Software Básico
Procedures

- Passagem de parâmetros: Método de pilha

```
proc-name     PROC
              enter   XX,0
                . . . . . .
              <procedure body>
                . . . . . .
              leave
              ret
proc-name     ENDP
```

Programação de Software Básico – Linguagem Assembly – Operações com bits

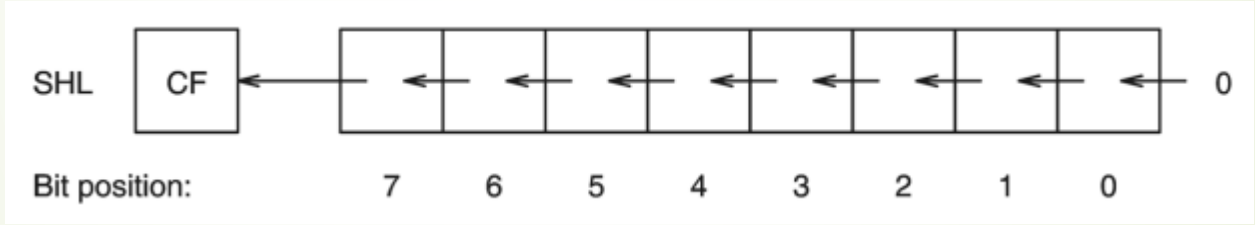
MATA49

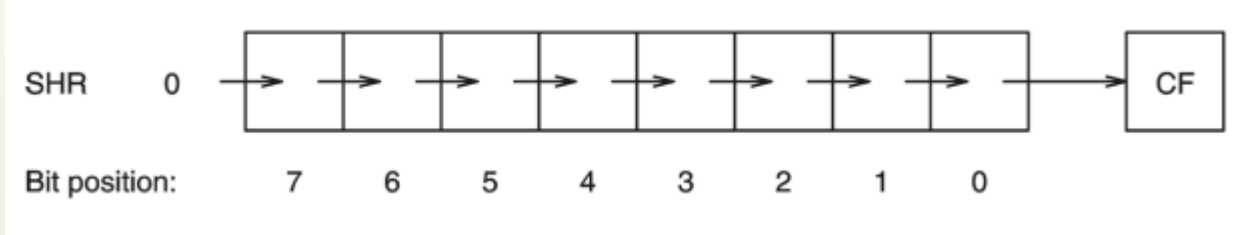
Prof. Babacar Mane

2021.1 – Aula 6

Programação de Software Básico
Operações com bits

- Deslocamento (shift)
  - Permite o deslocamento de uma cadeia de bits para direita ou para esquerda





Programação de Software Básico
Operações com bits

- Deslocamento (shift)
  - SHL e SHR
    - – SHL: deslocamento para esquerda (left)
    - – SHR: deslocamento para direita (right)
  - – Sintaxe:
    - SHL/SHR <destino>,<num_deslocamentos>

      SHL reg,imm8

      SHL mem,imm8

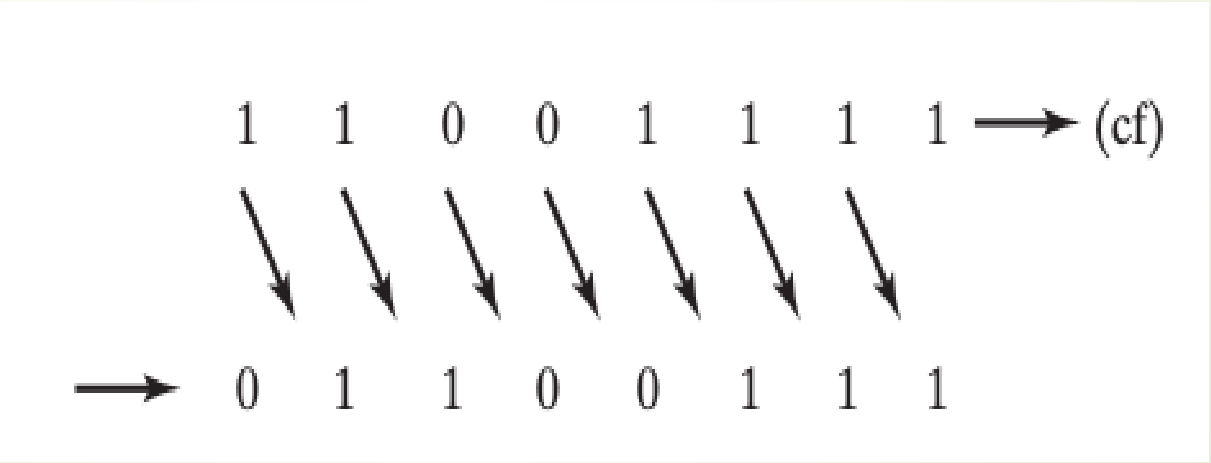
      SHL reg,CL

      SHL mem,CL

  - – O último bit deslocado para fora do registrador é armazenado no Carry Flag (CF)

Programação de Software Básico
Operações com bits

- Deslocamento (shift)
  - SHL e SHR
  - – Exemplo de deslocamento de 1 bit para direita:

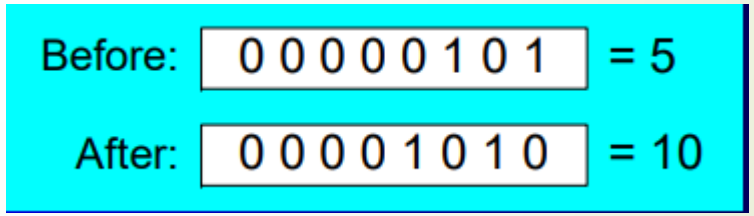


Programação de Software Básico
Operações com bits

- Deslocamento (shift)
  - SHL e SHR
  - – O deslocamento de 1 bit para esquerda multiplica o número por 2; n bits para esquerda multiplica por 2^n

Exemplo: mov dl,5

shl dl,1



- 2 bits

```
mov dl,5
shl dl,2          ;DL = 20
```

Programação de Software Básico
Operações com bits

- Deslocamento (shift)
  - SHL e SHR
  - – O deslocamento de 1 bit para direita divide o número por 2; n bits divide o número por 2^n

Exemplo:

```
mov dl,80
shr   dl,1 ;DL = 40  (divide 80/2)
shr   dl,2 ;DL = 20   (divide 80/4)
```

Programação de Software Básico
Operações com bits

- SAL e SAR instruções

Permite que números inteiros com sinais sejam deslocados conservando o sinal

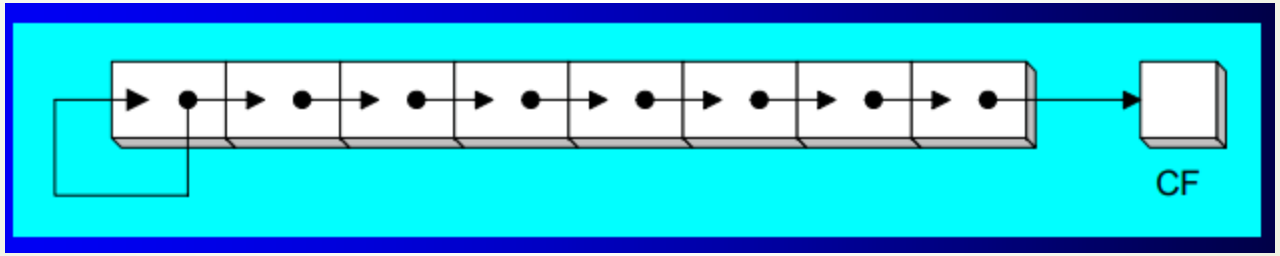
- – SAL/SAR <destino>, <num_deslocamentos>

A instrução SAL funciona de modo idêntico a SHL, pois o deslocamento a esquerda não implica em nenhuma alteração de sinal do inteiro

Programação de Software Básico
Operações com bits

- SAL e SAR instruções

```
mov dl,-80
sar dl,1 ; DL = -40
sar dl,2 ; DL = -20
```



Programação de Software Básico
Operações com bits

- SAL e SAR instruções
- Execicios:

```
mov al,6Bh
shr al,1
shl al,3
mov al,8Ch
sar al,1
sar al,3
```

a. 35h

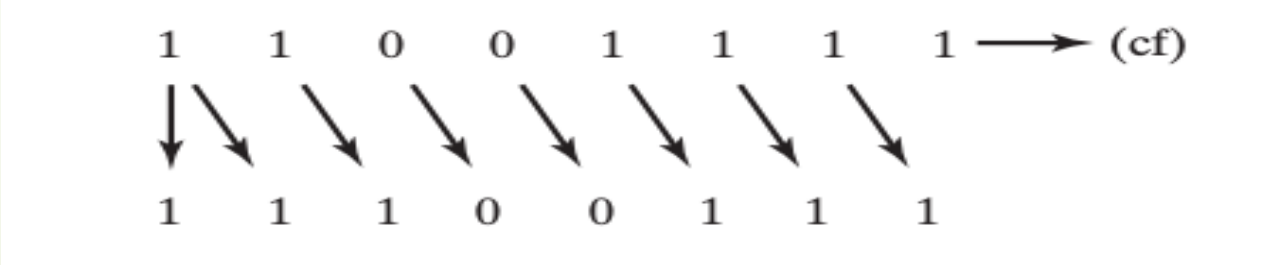
b. A8h

c. C6h

d. F8h

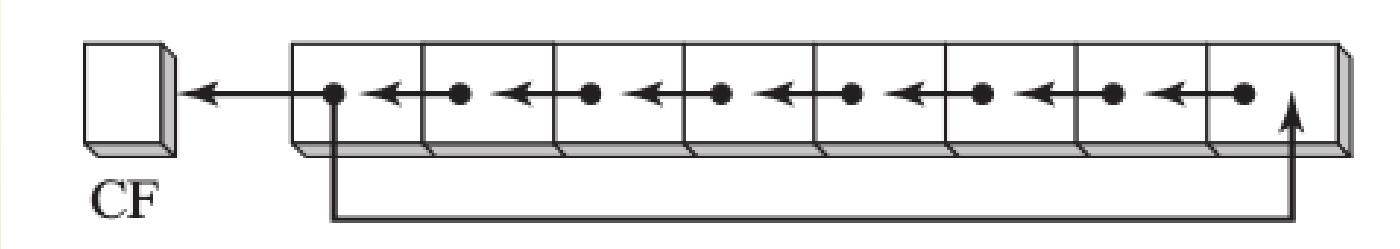
Programação de Software Básico
Operações com bits

- SAL e SAR instruções
- Exemplo de deslocamento de 1 bit:
- – Note que o bit acrescentado é 1, para conservar o sinal do número inteiro (negativo)



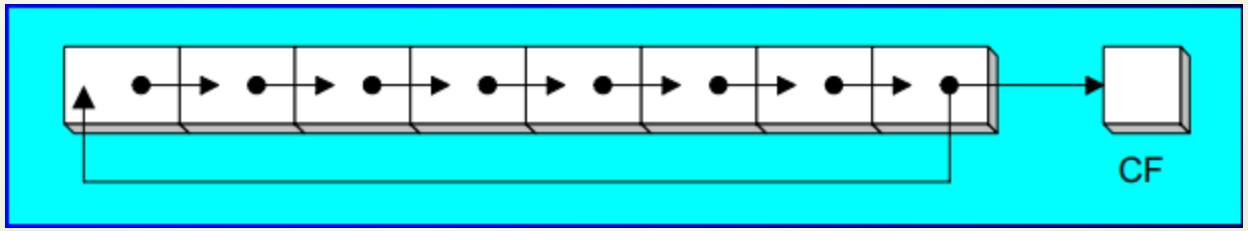
Programação de Software Básico
Operações com bits

- ROL / ROR instrução
- Faz um deslocamento circular, ou seja os bits deslocados irão para fim/início da cadeia
- – Sintaxe:
  - ROL/ROR <destino>, <num_deslocam>



Programação de Software Básico
Operações com bits

- ROL / ROR instrução
- Faz um deslocamento circular, ou seja os bits deslocados irão para fim/início da cadeia
- – Sintaxe:
  - ROL/ROR <destino>, <num_deslocam>



Programação de Software Básico
Operações com bits

- ROL / ROR instrução - Exemplos

```
mov al,40h                  ; AL = 01000000b
rol al,1                    ; AL = 10000000b, CF = 0
rol al,1                    ; AL = 00000001b, CF = 1
rol al,1                    ; AL = 00000010b, CF = 0
```

```
mov al,00000100b
ror al,3                    ; AL = 10000000b, CF = 1
```

Programação de Software Básico
Operações com bits

- Conta o número de bits “on” de EAX - Exemplo

```
1       mov     bl, 0           ; bl will contain the count of ON bits
2       mov     ecx, 32         ; ecx is the loop counter
3 count_loop:
4       shl     eax, 1          ; shift bit into carry flag
5       jnc     skip_inc        ; if CF == 0, goto skip_inc
6       inc     bl
7 skip_inc:
8       loop    count_loop
```

Programação de Software Básico
Operações com bits

- Exercício

Como armazenar o resultado de uma multiplicação localizada nos registradores DX:AX em um registrador de 32-bits com as instruções de descolamento de bits?

Programação de Software Básico
Operações com bits

- Operações Booleanas:

AND:

- Realiza a operação booleana “e” bit-a-bit nos operandos
- Sintaxe: AND <destino>, <fonte>

  destino = destino AND fonte

x	y	x ∧ y
0	0	0
0	1	0
1	0	0
1	1	1

Programação de Software Básico
Operações com bits

- Operações Booleanas:

– AND:

- Exemplos:

```
mov al, 11001111b
and al, 00001100b                ; al = 00001100

mov al,10101110b
and al,11110110b                 ;al = 10100110
```

Programação de Software Básico
Operações com bits

- Operações Booleanas:

– AND:

Exemplo de aplicação conversão de caractere de minúsculo para maiúsculo (vice-versa):

```
0 1 1 0 0 0 0 1 = 61h ('a')
0 1 0 0 0 0 0 1 = 41h ('A')
```

Programação de Software Básico
Operações com bits

- Operações Booleanas:

– OR:

- Aplica a operação booleana “ou” bit-a-bit nos operandos

- Sintaxe:

OR <destino>, <fonte>

destino = destino OR fonte

x	y	x ∨ y
0	0	0
0	1	1
1	0	1
1	1	1

Programação de Software Básico
Operações com bits

- Operações Booleanas:

– OR:

Exemplo:

```
mov al, 11001111b
or al, 00001100b                ; al = 11001111
```

Programação de Software Básico
Operações com bits

- Operações Booleanas:

– XOR:

- Aplica o “ou exclusivo” bit-a-bit nos operandos

- Sintaxe:

XOR <destino>, <fonte>

destino = destino XOR fonte

x	y	x ⊕ y
0	0	0
0	1	1
1	0	1
1	1	0

Programação de Software Básico
Operações com bits

- Operações Booleanas:

– XOR:

Exemplo:

```
mov al, 11001111b
xor al, 00001100b          ; al = 11000011
```

Programação de Software Básico
Operações com bits

- Operações Booleanas:

NOT

● Aplica o complemento de 1, ou seja inverte o valor de cada bit

● Sintaxe: NOT <destino>

ATENÇÃO! A operação NOT é diferente da NEG.

– NEG: aplica complemento de 2

– NOT: aplica complemento de 1

Programação de Software Básico
Operações com bits

- Operações Booleanas:

– NOT

Exemplo:

```
mov al, 11001111b
not al                    ; al = 00110000
```

Programação de Software Básico
Operações com bits

- Operações Booleanas:

TEST

– Aplica uma operação AND porém não armazena o resultado.

– Modifica os bits Flags de acordo com o resultado da operação (Zero Flag)

- Caso o resultado tenha algum bit com valor 1 o ZF=0
- Caso o resultado tenha todos os bits com valor 0 o ZF=1

– Sintaxe: TEST <destino>, <fonte>

Programação de Software Básico
Operações com bits

- Operações Booleanas:

TEST

– Exemplos:

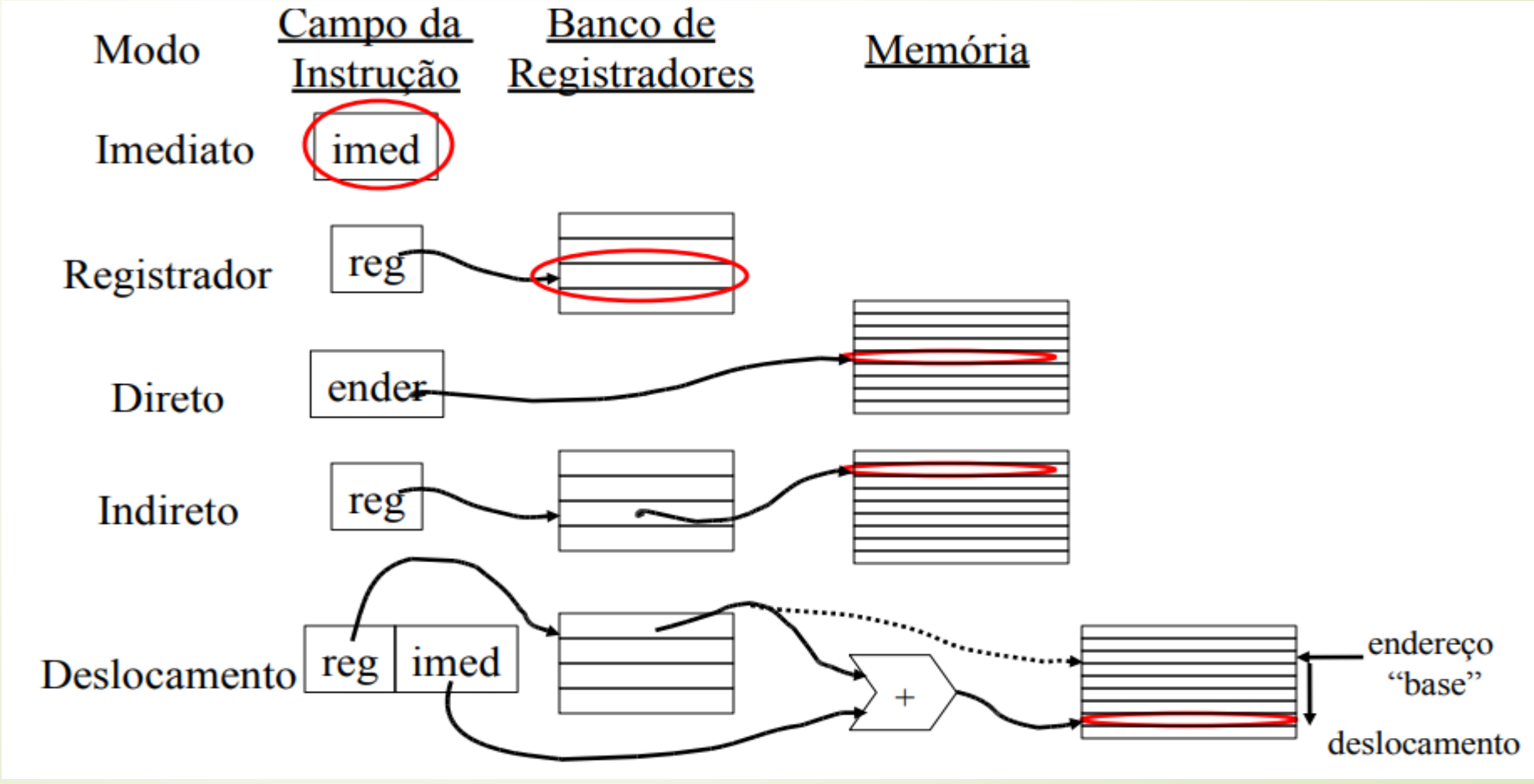
```
0 0 1 0 0 1 0 1   <- input value
0 0 0 0 1 0 0 1   <- test value
0 0 0 0 0 0 0 1   <- result: ZF = 0

0 0 1 0 0 1 0 0   <- input value
0 0 0 0 1 0 0 1   <- test value
0 0 0 0 0 0 0 0   <- result: ZF = 1
```

Programação de Software Básico
Modos de endereçamento

- Modos de endereçamento:
  - Imediato MOV AL,22H
  - Direto MOV [1234H],CX
  - Registro MOV CX,DX
  - Indireto MOV AX,[BX]
  - Base mais índice MOV [BX+DI],CL
  - Relativo a registro MOV AX,[BX+4]

Programação de Software Básico
Modos de endereçamento

- Modos de endereçamento:



Programação de Software Básico
Operações com bits

- Implemente uma calculadora pós-fixada, contendo as operações básicas (+, -, *, /) e com números positivos e negativos.

Programação de Software Básico
Operações com bits

- Torre de Hanói com 3 discos

Desenvolva um programa para resolução de uma Torre de Hanoi com 3 discos:

Regras para resolução da Torre de Hanoi

O objetivo é passar os três discos da coluna A para coluna C

Seguindo as seguintes regras:

– Somente um disco por vez;

– Um disco maior não pode ficar sobre um menor;

Programação de Software Básico
Operações com bits

- Torre de Hanói com 3 discos