Procedures

Procedures

- Duas instruções são utilizadas para manipular uma procedure: call e ret.
- A instrução call invoca uma procedure proc-name:

```
call    proc-name
```

- A linguagem assembly considera proc-name como o offset da primeira instrução da procedure proc-name.

A instrução call

```
 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
;*************************************************
```

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
- Esta instrução deve ser executada após a execução da procedure **sum**.
- O processador empurra o conteúdo de EIP na pilha.

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 da instrução **push** EBP.

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.

A instrução ret

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

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

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
```

Passagem de parâmetro

- Passagem de parâmetro na linguagem assembly é diferente e mais complicado do que nas linguagens de alto nível.
- Inicialmente, o chamador da procedure coloca todos os parâmetros necessários em uma área comum de armazenamento (memória ou registradores).
- Existem dois métodos para invocar as procedures: o método registrador e o método pilha

O método 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.

O método 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

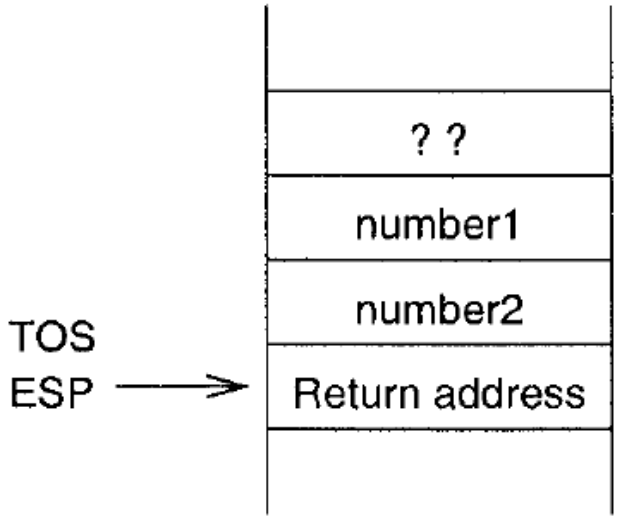
O método pilha

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

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

O método pilha

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



O método pilha

- A leitura dos argumentos number1 e number2 na pilha é complicada, pois teremos que remover o endereço de retorno antes.

```
pop     EAX
pop     EBX
pop     ECX
```

- Desde que o endereço de retorno foi removido temos que empurrá-lo novamente:

```
push    EAX
```

O método pilha

- Se teremos que retirar os parâmetros da pilha e armazená-los nos registradores como retirar 10 parâmetros, por exemplo ?
- É possível utilizar variáveis na memória para receber esses parâmetros, no entanto isto é ineficiente.
- Qual seria o melhor caminho para obter o valor dos parâmetros ?

O método pilha

- O melhor caminho para se obter os valores dos parâmetros é deixá-los na pilha e ler quando necessário.
- Desde que a pilha é uma sequência de locais de memória, **ESP+4** aponta para **number2** e **ESP+6** aponta para **number1**.
- Portanto, faremos, por exemplo:

```
mov     EBX, [ESP+4]
```

O método pilha

- O exemplo anterior causa um problema. O uso das instruções **push** e **pop** atualizam o valor de **ESP**.
- Suponha, por exemplo que a procedura que foi chamada realize operações na pilha. Então **ESP** será atualizado.
- Existe uma alternativa melhor que é a utilização do registrador **EBP.**

O método pilha

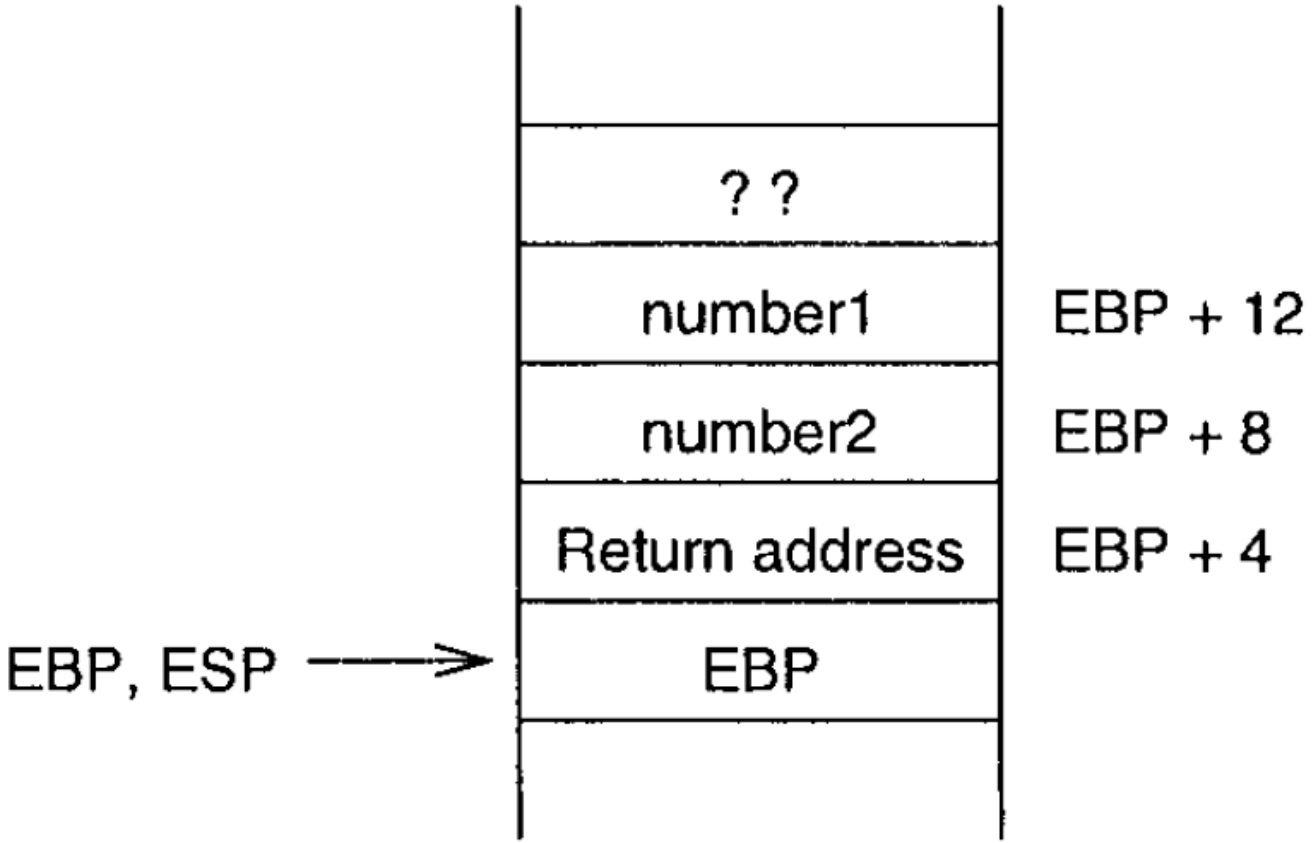
- Para especificar o offset em **EBP** e acessar o parâmetro **number2** e armazená-lo em **EAX**:

```
mov     EBP,ESP
mov     EAX,[EBP+4]
```

- Desde de que todos as procedures utilizam o **EBP** para acessar parâmetros, o valor de **EBP** precisa ser preservado:

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

Pilha após empurrar EBP



(a) Stack after saving EBP

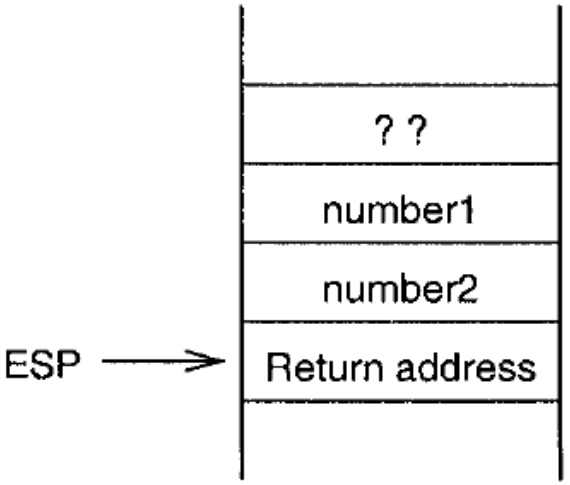
O frame da pilha

- 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.

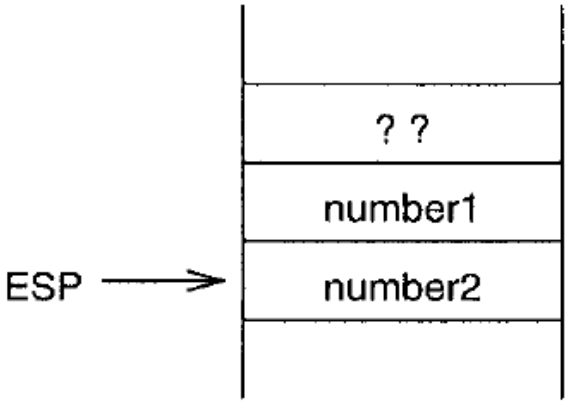
O frame da pilha

- Antes de retornar da procedure, o EBP precisa ser restaurado:

```
pop    EBP
```



(b) Stack after `pop EBP`



(c) Stack after `ret`

Exemplo - passagem de parâmetro por referência com registradores

```
BUF_LEN     EQU  41              ; string buffer length

.DATA
prompt_msg  db  "Please input a string: ",0
length_msg  db  "The string length is ",0

.UDATA
string      resb  BUF_LEN     ;input string < BUF_LEN chars.

.CODE
      .STARTUP
      PutStr  prompt_msg      ; request string input
      GetStr  string,BUF_LEN ; read string from keyboard

      mov     EBX,string      ; EBX = string address
      call    str_len         ; returns string length in AX
      PutStr  length_msg      ; display string length
      PutInt  AX
      nwln
```

Exemplo - passagem de parâmetro por referência com registradores

```
30:  ;-------------------------------------------------------
31:  ;Procedure str_len receives a pointer to a string in BX.
32:  ;String length is returned in AX.
33:  ;-------------------------------------------------------
34:  str_len:
35:        push    EBX
36:        sub     AX,AX           ; string length = 0
37:  repeat:
38:        cmp     byte [EBX],0    ; compare with NULL char.
39:        je      str_len_done    ; if NULL we are done
40:        inc     AX              ; else, increment string length
41:        inc     EBX             ; point BX to the next char.


42:        jmp     repeat          ;  and repeat the process
43:  str_len_done:
44:        pop     EBX
45:        ret
```

Exemplo - passagem de parâmetro por referência com pilha

```
 8:  BUF_LEN     EQU  41             ; string buffer length
 9:  %include "io.mac"
10:
11:  .DATA
12:  prompt_msg  db    "Please input a string: ",0
13:  output_msg  db    "The swapped string is: ",0
14:
15:  .UDATA
16:  string      resb  BUF_LEN       ;input string < BUF_LEN chars.
17:
18:  .CODE
19:        .STARTUP
20:        PutStr  prompt_msg      ; request string input
21:        GetStr  string,BUF_LEN ; read string from the user
22:
23:        mov     EAX,string      ; EAX = string[0] pointer
24:        push    EAX
25:        inc     EAX             ; EAX = string[1] pointer
26:        push    EAX
27:        call    swap            ; swaps the first two characters
28:        PutStr  output_msg      ; display the swapped string
29:        PutStr  string
30:        nwln
```

Exemplo - passagem de parâmetro por referência com pilha

```
35:  ;Procedure swap receives two pointers (via the stack) to
36:  ;characters of a string. It exchanges these two characters.
37:  ;-----------------------------------------------------------
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
```

Exemplo - passagem de parâmetro por valor com pilha

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

Exemplo - passagem de parâmetro por valor com pilha

```
30:  ;-------------------------------------------------------
31:  ;Procedure sum receives two integers via the stack.
32:  ;The sum of the two integers is returned in AX.
33:  ;-------------------------------------------------------
34:  sum:
35:        enter   0,0             ; save EBP
36:        mov     AX,[EBP+10]     ; sum = first number
37:        add     AX,[EBP+8]      ; sum = sum + second number
38:        leave                   ; restore EBP
39:        ret     4               ; return and clear parameters
```

Exercícios

1. Escreva uma procedure Min2 para encontrar o mínimo de dois parâmetros inteiros de tamanho 16 bit. Retorne o mínimo no registrador AX.

2. Escreva uma procedure Max3 para encontrar o maximo de três parâmetros inteiros de 32 bits cada. Retorne o máximo no registrador EAX.

Exercícios

3. Escreva uma procedure *Avg para encontrar a média da coleção de inteiros em um array. A* Procedure *Avg terá três parâmetros*:

(1) o endereço do array

(2) o número de inteiros no array

(3) o endereço no qual se armazenará o resultado

Exercício

4 – Escreva uma procedure busca *para consultar um array de inteiros para um valor específico*. A Procedure busca terá três parametros:

(1) o valor para qual buscar

(2) o endereço do array

(3) o número *N inteiros no array*

Retorne in EAX a posição (1,2,...,N) na qual o valor é encontrado ou retorne zero se o valor não aparece no array.

Exercícios

- Torre de hanoi. Mover os discos de A para B sendo que um disco maior nunca pode estar em cima de um menor: