# EL2003 Computer Organization & Assembly Language

# Lab 10
Advanced Procedures

**Prepared By:**

**Muhammad Nadeem Ghouri**
**muhammad.nadeem@nu.edu.pk**

**NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES**

# LAB 10

## Learning Objectives

- Implementing procedures using stack frame
- Using stack parameters in procedures
- Passing value type and reference type parameters

## Stack Applications

There are several important uses of runtime stacks in programs:

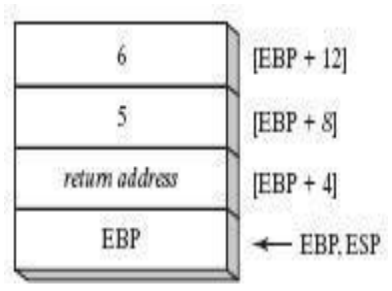
- A stack makes a convenient temporary save area for registers when they are used for more than one purpose. After they are modified, they *can* be restored to their original values.
- When the CALL instruction executes, the CPU saves the current subroutine's return address on the stack.
- When calling a subroutine, you pass input values called arguments by pushing them on the stack.
- The stack provides temporary storage for local variables inside subroutines.

## Stack Parameters

- **Passing by value**
  When an argument is passed by value, a copy of the value is pushed on the stack.

### EXAMPLE # 01:

```
.data
        var1    DWORD        5
        var2    DWORD        6

.code
        push var2
        push var1
        call AddTwo
        exit
AddTwo PROC
        push    ebp
        mov     ebp, esp
        mov     eax, [ebp + 12]
        add     eax, [ebp + 8]
        pop     ebp
```



```
        ret
AddTwo ENDP
```

- **Explicit stack parameters**

When stack parameters are referenced with expressions such as [ebp+8], we call them explicit stack parameters.

**EXAMPLE # 02:**

```
.data
        var1    DWORD       5
        var2    DWORD       6

        y_param     EQU     [ebp + 12]
        x_param     EQU     [ebp+ 8]

.code
        push var2
        push var1
        call AddTwo
        exit

AddTwo PROC
        push        ebp
        mov ebp, esp
        mov   eax,  y_param
        add   eax,  x_param
        pop ebp
        ret
AddTwo ENDP
```

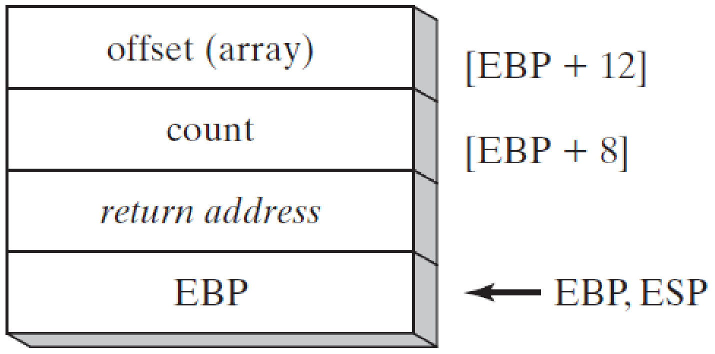
- **Passing by reference**

An argument passed by reference consists of the offset of an object to be passed.

**EXAMPLE # 03:**

```
.data
        count = 10
        arr     WORD        count DUP (?)

.code
        push    OFFSET arr
        push    count
        call    ArrayFill
```

```
 exit

ArrayFill        PROC
        push    ebp
        mov     ebp, esp
        pushad
        mov  esi,  [ebp  +  12]
        mov  ecx,  [ebp  +  8]
        cmp  ecx, 0
        je      L2

L1:
        mov     eax, 100h
        call    RandomRange
        mov     [esi], ax
        add     esi, TYPE WORD
        loop    L1
L2:
        popad
        pop     ebp
        ret     8
ArrayFill        ENDP
```



## LEA Instruction

LEA instruction returns the effective address of an indirect operand. Offsets of indirect operands are calculated at runtime.

**<u>EXAMPLE # 04:</u>**

```
.code
        call    makeArray
        exit

makeArray       PROC
        push    ebp
        mov     ebp, esp
        sub     esp, 32
        lea     esi, [ebp - 30]
        mov ecx,30
L1:
        mov     BYTE PTR [esi], '*'
        inc     esi
```

```
          loop     L1
          add      esp, 32
          pop      ebp
          ret
makeArray     ENDP
```

## ENTER & LEAVE Instructions

Enter instruction automatically creates stack frame for a called Procedure. Leave instruction reverses the effect of enter instruction.

**EXAMPLE # 06:**

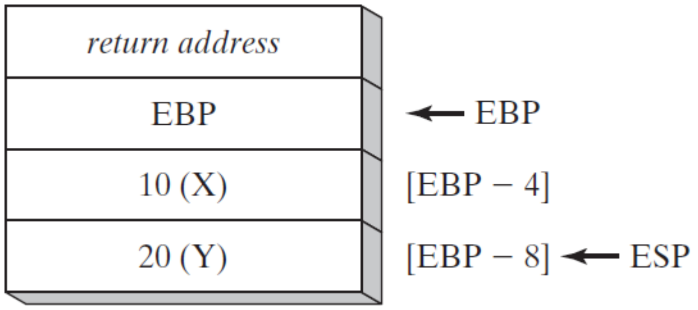
```
.data
          var1     DWORD        5
          var2     DWORD        6

.code
          push var2
          push var1
          call AddTwo
          exit


AddTwo PROC
          enter    0, 0
          mov      eax, [ebp + 12]
          add      eax, [ebp + 8]
          pop      ebp
          leave
          ret
AddTwo ENDP
```

## Local Variables

In MASM Assembly Language, local variables are created at runtime stack, below the base pointer (EBP).

**EXAMPLE # 05:**

```
.code
          call     MySub
          exit
```



```
MySub       PROC
        push    ebp
        mov     ebp, esp
        sub     esp, 8
        mov     DWORD       PTR     [ebp - 4], 10   ; first parameter
        mov     DWORD       PTR [ebp - 8], 20       ; second parameter
        mov     esp, ebp
        pop     ebp
        ret
MySub       ENDP
```

## LOCAL Directive

LOCAL directive declares one or more local variables by name, assigning them size attributes.

**EXAMPLE # 07:**

```
.code
        call LocalProc
        exit


LocalProc   PROC
        LOCAL           temp : DWORD
        mov     temp, 5
        mov     eax, temp
        ret
LocalProc   ENDP
```