Lab Session 10

Advanced Procedures

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:

```
INCLUDE Irvine32.inc
.data
var1    DWORD       5
var2    DWORD       6

.code
main PROC
push var2
push var1
```

```
call AddTwo
call Writeint
exit
main   ENDP
AddTwo PROC
push    ebp
mov     ebp, esp
mov     eax, [ebp  + 12]
add     eax, [ebp + 8]
pop     ebp
ret
AddTwo ENDP
END main
```

- **Explicit stack parameters**

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

EXAMPLE:

```
INCLUDE Irvine32.inc
.data
var1    DWORD        5
var2    DWORD        6
y_param        EQU    [ebp + 12]
x_param        EQU    [ebp+ 8]


.code
main PROC
push var2
push var1
call AddTwo
call Writeint
exit
main   ENDP
AddTwo PROC
push    ebp
mov     ebp, esp
mov     eax, y_param
add     eax, x_param
pop     ebp
ret
AddTwo ENDP
END main
```

- **Passing by reference**

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

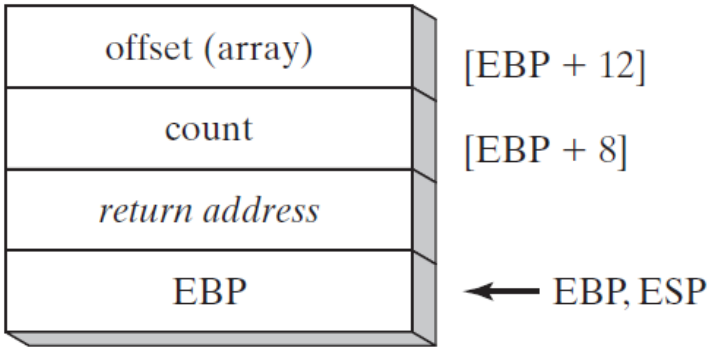
EXAMPLE:

```
.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
```

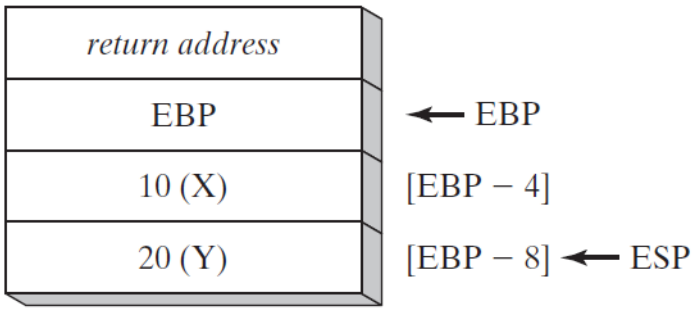


Local Variables

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

EXAMPLE:

```
.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
```

ENTER & LEAVE Instructions

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

EXAMPLE:

```
.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 Directive

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

EXAMPLE:

```
.code
        call LocalProc
        exit


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

Recursive Procedures

Recursive procedures are those that call themselves to perform some task.

EXAMPLE:

```
.code
        mov     ecx, 5
        mov     eax, 0
        call    CalcSum
L1:
        call    WriteDec
        call    crlf
        exit

CalcSum         PROC
        cmp     ecx, 0
        jz      L2
        add     eax, ecx
        dec     ecx
        call    CalcSum
L2:
        ret
CalcSum         ENDP
```

LAB TASK

1. Write a program which contains a procedure named **ThreeProd** that displays the product of three numeric parameters passed through a stack.

2. Write a program which contains a procedure named **MinArray** that displays the minimum value in an array. Pass a size-20 array by reference to this procedure. **(Use WORD type Array).**

3. Write a program which contains a procedure named **MaxArray** that displays the maximum value in an array. Pass a size-20 array by reference to this procedure. **(Use WORD type Array).**

4. Write a program which contains a procedure named **LocalSquare**.The procedure must declare a local variable. Initialize this variable by taking an input value from the user and then display its square. Use **ENTER*& LEAVE*** instructions to allocate and de-allocate the local variable.

5. Write a program that calculates factorial of a given number ***n***. Make a recursive procedure named **Fact** that takes n as an input parameter.