

# Assembly Language LAB

*Islamic University – Gaza*  
*Engineering Faculty*  
*Department of Computer Engineering*  
*2013*

*ECOM 2125: Assembly Language LAB*

*Created by: Eng. Ahmed M. Ayash*

*Modified and Presented By: Eihab S. El-Radie*



## *Lab # 9*

## *Advanced Procedures*

## Objective:

To learn more about procedures.

### Part1: Stack Frame (activation record):

Area of the stack set aside for passed arguments, subroutine return address, local variables, and saved registers.

#### Created by the following steps:

1. Passed arguments, if any, are pushed on the stack.
2. The subroutine is called, causing the subroutine return address to be pushed on the stack.
3. As the subroutine begins to execute, EBP is pushed on the stack.
4. EBP is set equal to ESP. From this point on, EBP acts as a base reference for all of the subroutine parameters.
5. If there are local variables, ESP is decremented to reserve space for the variables on the stack.
6. If any registers need to be saved, they are pushed on the stack.

#### ➤ Explicit Access to Stack Parameters:

A procedure can explicitly access stack parameters using constant offsets from EBP.

- EBP is often called the base pointer or frame pointer because it holds the base address of the stack frame.
- EBP must be restored to its original value when a procedure returns.

#### ➤ RET Instruction:

- Return from subroutine.
- Pops stack into the instruction pointer (EIP or IP). Control transfers to the target address.

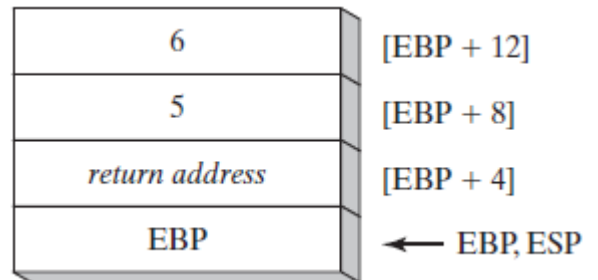
#### *Syntax:*

- RET
  - RET  $n$
- 
- Optional operand  $n$  causes  $n$  bytes to be added to the stack pointer after EIP (or IP) is assigned a value.

### Example:

```
.data
sum DWORD ?
.code
push 6      ; second argument
push 5      ; first argument
call AddTwo ; EAX = sum
mov  sum,eax ; save the sum
```

```
AddTwo PROC
push ebp
mov ebp,esp ; base of stack frame
mov eax,[ebp + 12] ; second parameter
add eax,[ebp + 8] ; first parameter
pop ebp
ret
AddTwo ENDP
```



### Part2: Local Variables:

- A **local variable** is created, used, and destroyed within a single procedure.
- Local variables are created on the runtime stack, usually below the base pointer (EBP).
- The LOCAL directive declares a list of local variables and immediately follows the PROC directive, each variable is assigned a type.
- Syntax: LOCAL varlist

#### Syntax:

```
LOCAL var1:type1, var2:type2, . . .
```

#### Example:

```
MySub PROC
LOCAL var1:BYTE, var2:WORD, var3:DWORD
```

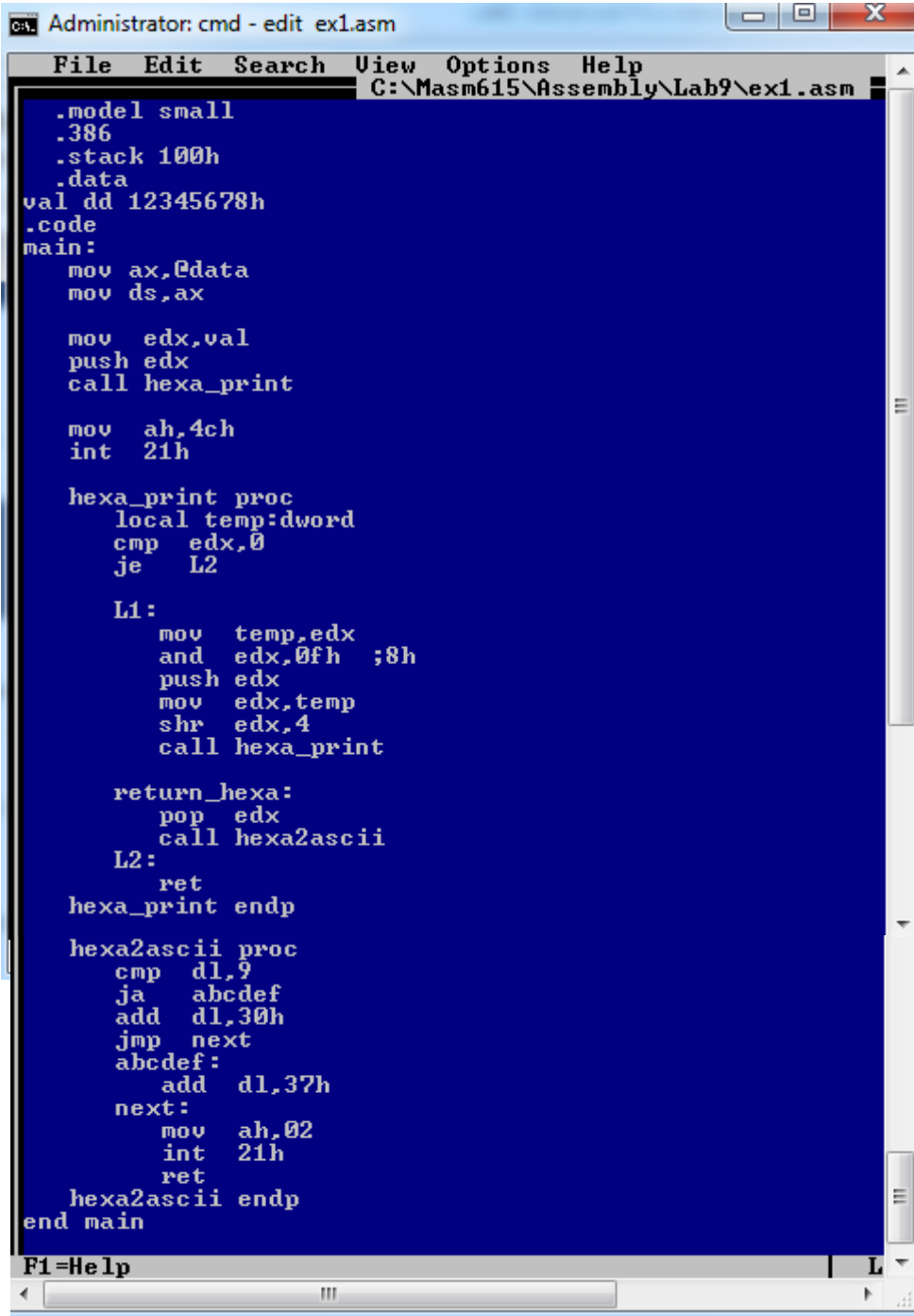
### Part3: Recursion:

- A recursive procedure is one that calls itself, either directly or indirectly.
- Recursion, the practice of calling recursive procedures, can be a powerful tool when working with data structures that have repeating patterns.

## Lab work:

### Excercise1:

Write an assembly recursive procedure that prints a hexadecimal number saved in memory on the screen.



```
Administrator: cmd - edit ex1.asm
File Edit Search View Options Help
C:\Masm615\Assembly\Lab9\ex1.asm
.model small
.386
.stack 100h
.data
val dd 12345678h
.code
main:
    mov ax,@data
    mov ds,ax

    mov edx,val
    push edx
    call hexa_print

    mov ah,4ch
    int 21h

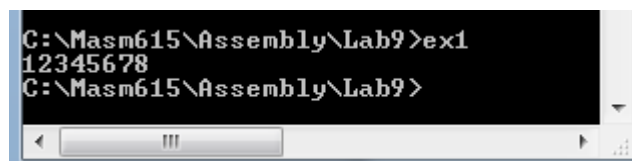
hexa_print proc
    local temp:dword
    cmp edx,0
    je L2

    L1:
        mov temp,edx
        and edx,0fh ;8h
        push edx
        mov edx,temp
        shr edx,4
        call hexa_print

    return_hexa:
        pop edx
        call hexa2ascii
    L2:
        ret
hexa_print endp

hexa2ascii proc
    cmp dl,9
    ja abcdef
    add dl,30h
    jmp next
abcdef:
    add dl,37h
next:
    mov ah,02
    int 21h
    ret
hexa2ascii endp
end main
```

## Output:

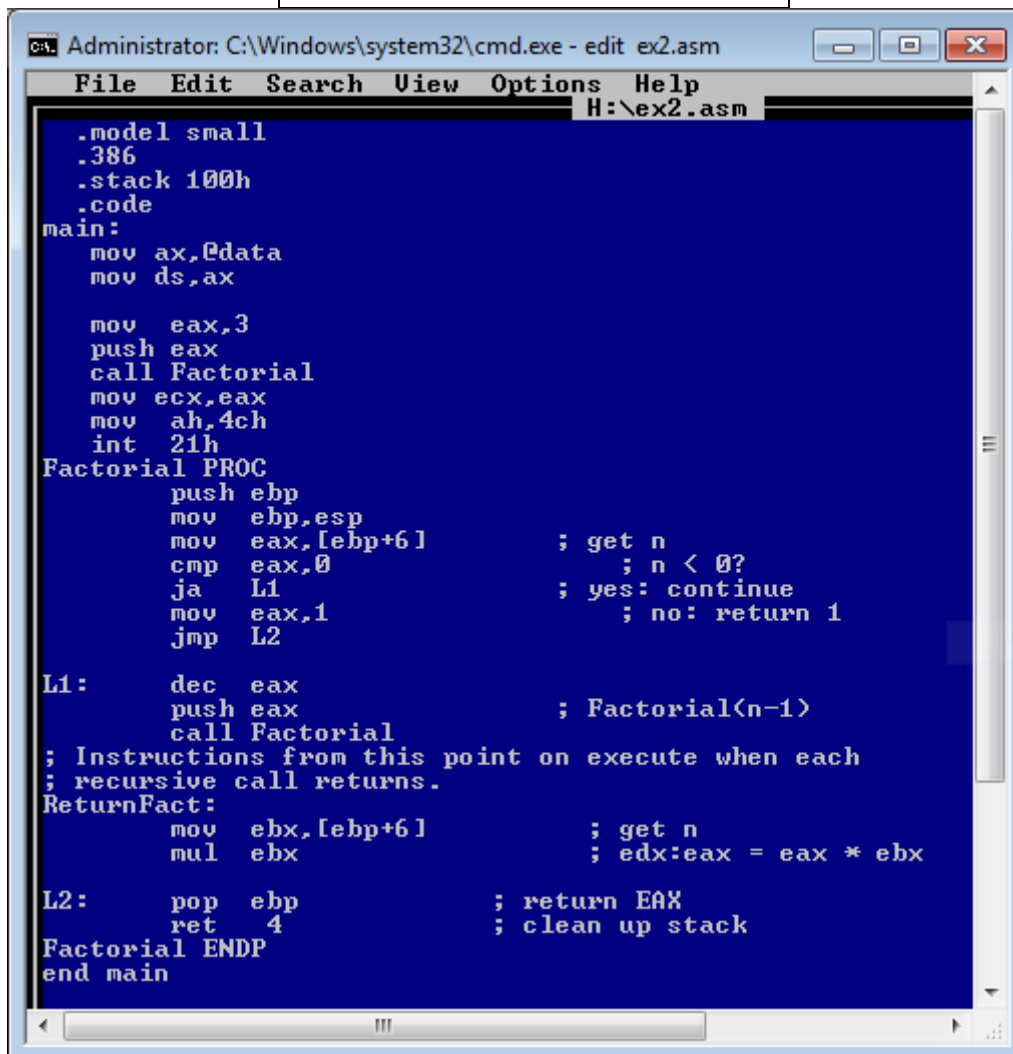


```
C:\Masm615\Assembly\Lab9>ex1
12345678
C:\Masm615\Assembly\Lab9>
```

## Excercise2:

This function calculates the factorial of integer  $n \geq 0$ . A new value of  $n$  is saved in each stack frame: (use  $n=3, 5, 12$ )

```
int function factorial(int n)
{
    if(n == 0)
        return 1;
    else
        return n * factorial(n-1);
}
```



The screenshot shows a Windows command prompt window titled "Administrator: C:\Windows\system32\cmd.exe - edit ex2.asm". The window displays the assembly code for a factorial function. The code is as follows:

```
.model small
.386
.stack 100h
.code
main:
    mov ax,@data
    mov ds,ax

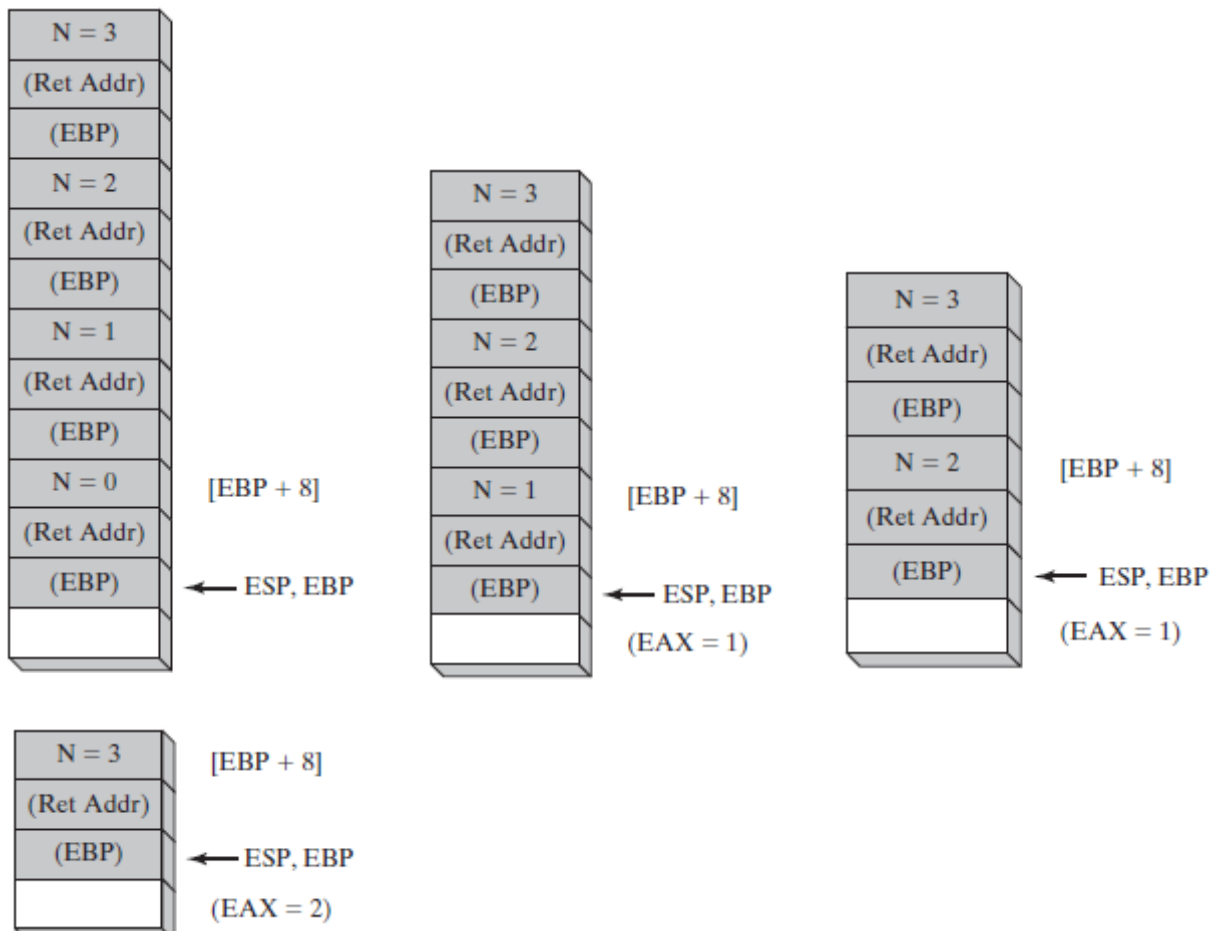
    mov eax,3
    push eax
    call Factorial
    mov ecx,eax
    mov ah,4ch
    int 21h
Factorial PROC
    push ebp
    mov ebp,esp
    mov eax,[ebp+6]      ; get n
    cmp eax,0           ; n < 0?
    ja  L1              ; yes: continue
    mov eax,1           ; no: return 1
    jmp L2
L1:    dec  eax
    push eax            ; Factorial(n-1)
    call Factorial
; Instructions from this point on execute when each
; recursive call returns.
ReturnFact:
    mov  ebx,[ebp+6]    ; get n
    mul  ebx            ; edx:eax = eax * ebx
L2:    pop  ebp          ; return EAX
    ret  4              ; clean up stack
Factorial ENDP
end main
```

```

[ ]-CPU 80486-                               1-[↑][↓]
cs:0010 668BC8      mov     ecx,eax      ▲eax 00000006  c=0
cs:0013 B44C       mov     ah,4C          ■ebx 00000003  z=0
cs:0015 CD21       int     21             ecx 00000006  s=0
cs:0017 6655       push   ebp            edx 00000000  o=0
cs:0019 668BEC     mov     ebp,esp       esi 000026F1  p=1
cs:001C 67668B4506 mov     eax,[ebp+06]  edi 000001A2  a=0
cs:0021 6683F800   cmp     eax,0000      ebp 00000100  i=1
cs:0025 7708       ja     002F           esp 00000100  d=0
cs:0027 66B801000000 mov    eax,00000001  ds 48B1
cs:002D EBOF       jmp     003E           es 489D
cs:002F 6648       dec     eax            fs 0000
cs:0031 6650       push   eax            gs 0000
cs:0033 EBE1FF     call  0017           ss 48B2
                                cs 48AD
                                ip 0013
es:0000 CD 20 FF 9F 00 EA FF FF = f Ω
es:0008 AD DE E0 01 C5 15 AA 01 i |x|s-|
es:0010 C5 15 89 02 20 10 92 01 |s|e| |ff|
es:0018 01 03 01 00 02 FF FF FF |v| | |
                                ss:0102 0000
                                ss:0100 0040

```

L2: pop ebp  
ret 4



### Homework:

1. Write an assembly recursive procedure that calculates the GCD of any two numbers using the following algorithm:

```
GCD(a,b) {  
  If (a%b==0)  
    return b;  
  else  
    return GCD(b , a%b);  
}
```

Find: GCD(72,24), GCD(8,12).

2. Modify Excercise1 so that the output will be 87654321.