LAB 08

STACK, IT'S OPERATION AND NESTED PROCEDURES

STUDENT NAME _____ ROLL NO _____ SEC ___

SIGNATURE & DATE _____

MARKS AWARDED: _____

NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES (NUCES), KARACHI

Prepared by: Aashir Mahboob

Version: 1.0
Date: 27th Oct 2021

Lab Session 08: STACK,IT'S OPERATION & NESTED PROCEDURES

Objectives:

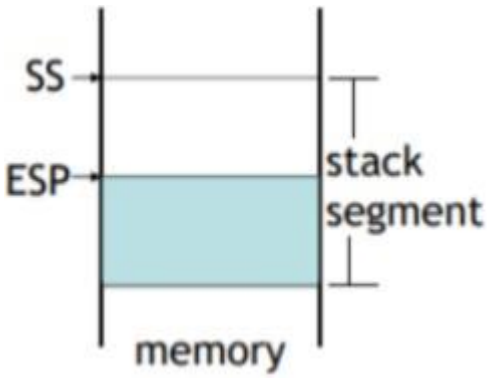
- To learn about Runtime Stack and how to implement using PUSH and POP instructions
- To learn about user defined procedures and to use related Instructions
- Undersatnding the Nested Procedures and the way those are implemented in assembly

Stack:

- LIFO (Last-In, First-Out) data structure.
- push/ pop operations
- You probably have had experiences on implementing it in high-level languages.
- Here, we concentrate on runtime stack, directly supported by hardware in the CPU. It is essential for calling and returning from procedures.

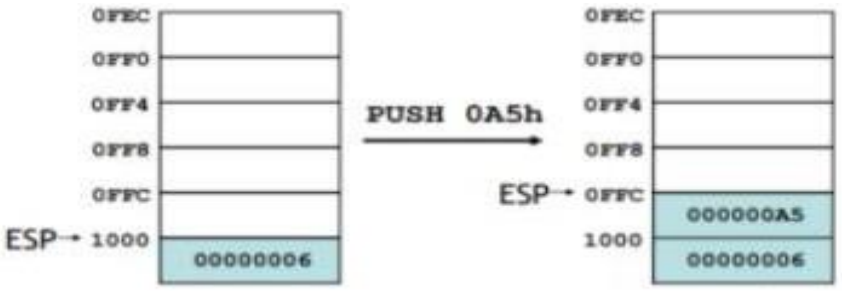
Runtime Stack:

- Managed by the CPU, using two registers
- SS (stack segment)
- ESP (stack pointer): point the last value to be added to, or pushed on, the top of stack usually modified by instructions: ***CALL, RET, PUSH and POP***

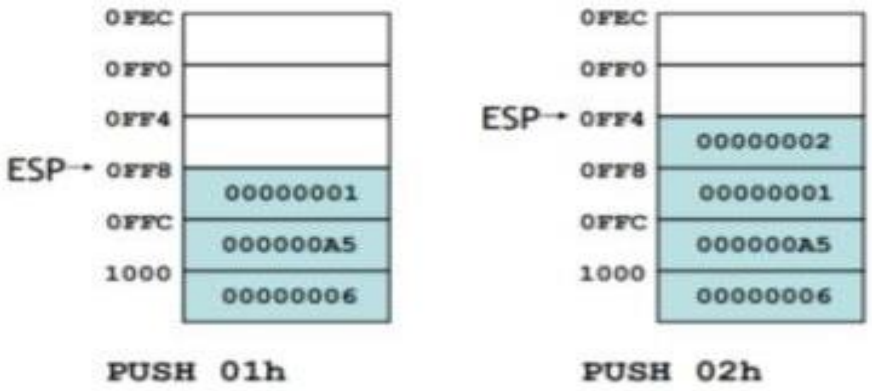


Push Operation

A 32-bit push operation decrements the stack pointer by 4 and copies a value into the location in the stack pointed to by the stack pointer.

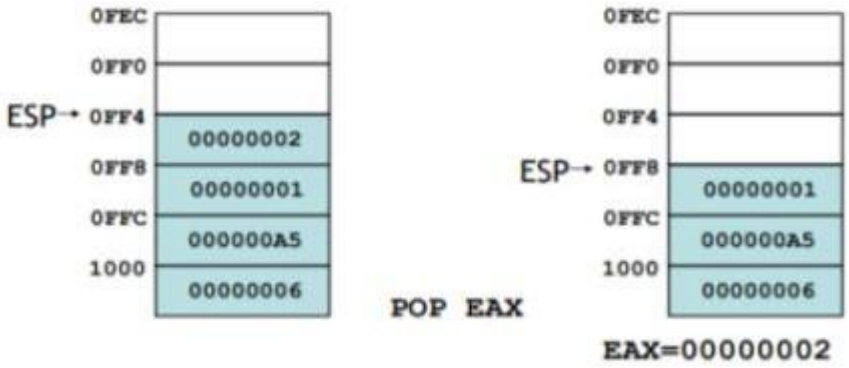


- The same stack after pushing two more integers:



Pop Operation

A pop operation removes a value from the stack. After the value is popped from the stack, the stack pointer is incremented (by the stack element size) to point to the next- highest location in the stack. It copies value at stack [ESP] into a register or variable.



PUSH and POP instructions:

PUSH syntax:

- PUSH r/m16
- PUSH r/m32
- PUSH imm32

POP syntax:

- POP r/m16
- POP r/m32

PUSHFD and POPFD Instructions

The MOV instruction cannot be used to copy the flags to a variable.

The **PUSHFD** instruction pushes the 32-bit EFLAGS register on the stack, and **POPFD** pops the stack into EFLAGS:

- PUSHFD
- POPFD

Example 01: (Stack and nested loops.)

```
Include Irvine32.inc
.code
main proc
mov ecx,5
L1:
      push ecx
      mov ecx, 10
      L2:
            inc ebx
       loop L2
       pop ecx
loop L1

call     DumpRegs
exit
main ENDP
END main
```

Example 02:(displays the Addition of three integers through a stack)

```
Include Irvine32.inc
 .data
      VAR1 DWORD 2
.code
main proc
            mov eax, 0
            mov ecx, 3
            L1:
                  PUSH VAR1
                  ADD VAR1, 2
            LOOP L1
            mov ecx, 3
            L2:
                  POP ebx
                  ADD eax, ebx          ;eax value added
            LOOP L2

call DumpRegs
exit
main ENDP
END main
```

Example 03:(To find the largest number through a stack)

```
Include Irvine32.inc
.code
main proc
PUSH 5
PUSH 7
PUSH 3
PUSH 2
MOV eax, 0                                  ;eax is the largest
MOV ecx, 4
L1:
        POP edx
        CMP edx, eax
        JL SET
        MOV eax, edx
    SET:
LOOP L1
call    DumpRegs
exit
main ENDP
END main
```

Procedures

- Procedures or subroutines are very important in assembly language, as the assembly language programs tend to be large in size.
- Procedures are identified by a name. Following this name, the body of the procedure is described which performs a well-defined job.
- End of the procedure is indicated by a return statement.

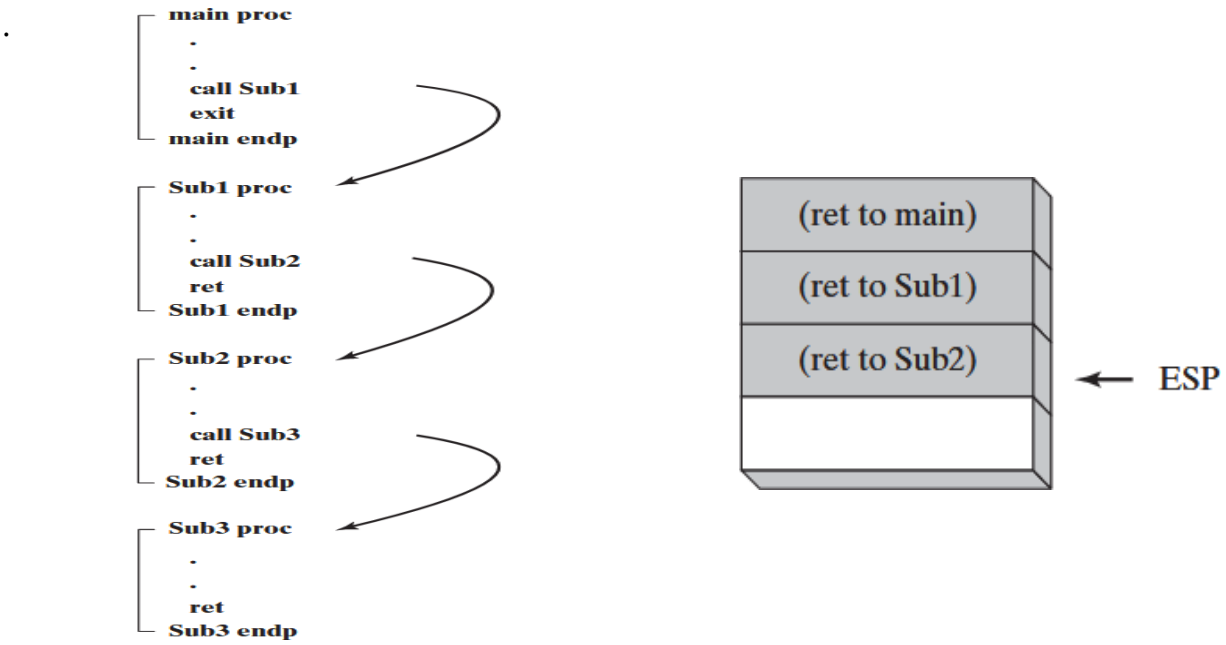
Example 04:

```
INCLUDE Irvine32.inc
INTEGER_COUNT = 3
.data
  str1 BYTE "Enter a signed integer: ",0
  str2 BYTE "The sum of the integers is: ",0
  array DWORD INTEGER_COUNT DUP(?)

.code
main PROC
call Clrscr
mov esi, OFFSET array
mov ecx, INTEGER_COUNT
```

```
call PromptForIntegers
call ArraySum
call DisplaySum

exit
main ENDP

;----------- PromptForIntegers -------------
PromptForIntegers PROC USES ecx edx esi
mov edx, OFFSET str1            ; "Enter a signed integer"
L1:
      WriteString               ; display string
      call ReadInt              ; read integer into EAX
      call Crlf                 ; go to next output line
      mov [esi], eax            ; store in array
      add esi, TYPE DWORD       ; next integer
loop L1
ret
PromptForIntegers ENDP

;----------- ArraySum -------------
ArraySum PROC USES esi ecx
mov eax,0                       ; initialize the value of sum to ZERO
 L1:
      add eax, [esi]            ; add each integer to sum
      add esi, TYPE DWORD       ; point to next integer
loop L1                         ; repeat for array size
ret                             ; sum is in EAX
ArraySum ENDP

;----------- DisplaySum -------------
DisplaySum PROC USES edx
mov edx, OFFSET str2
call WriteString
call WriteInt                   ; display EAX
call Crlf
ret
DisplaySum ENDP
 END main
```

Nested Procedure Calls

A nested procedure call occurs when a called procedure calls another procedure before the first procedure returns.



Example 05:

```
Include Irvine32.inc
 .data
      var1    DWORD 5
      var2    DWORD 6
.code
main  proc
call AddTwo
call dumpregs
call writeint
call crlf
exit
main ENDP

AddTwo PROC
Mov eax,var1
Mov ebx,var2
Add eax,var2

Call AddTwo1
Ret
Addtwo ENDP

AddTwo1 PROC
Mov ecx,var1
Mov edx,var2
Add ecx, var2
Call writeint
Ret
AddTwo1 ENDP
```

Lab Task(s):

Task#1:

Take an array atleast of 10 numbers, move word-type of data in reverse order into another empty array using stack push and pop technique.

Task#2

Write a program having nested procedures used to calculate the total sum of 2 arrays (each array having atleast 5-elements). The sum of 1-array in 1st procedure and in 2nd procedure have sum of 2-array. And the 3rd procedure adds the results of both.

Task#3

Print the following pattern using a function call in which number of columns is passed through a variable.

```
*
**
***
****
*****
```

Task#4

Print the following pattern using a function call in which number of columns is passed through a variable.

```
A
BC
DEF
GHIJ
KLMN
```

Task#5

Write a function that asks the user for a number n and prints the sum of the numbers 1 to n.

LAB 8

Task 1: Code + Output

```
14  mov ecx,lengthof arr
15  mov ebx,type arr
16
17  call dumpmem
18
19  mov esi,0
20  mov ecx,10
21  l1:
22  push arr[esi*type arr]
23  inc esi
24  loop l1
25
26  mov esi,0
27  mov ecx,10
28  l2:
29  pop arr2[esi*type arr]
30  inc esi
31  loop l2
32
33  ;For Calling Dump mem
```

```
Microsoft Visual Studio Debug Console

Dump of offset 00FF6000
-------------------------------
0001 0002 0003 0004 0005 0006 0007 0008 0009 000A

Dump of offset 00FF6014
-------------------------------
000A 0009 0008 0007 0006 0005 0004 0003 0002 0001

C:\Users\acer\source\repos\Project2\Debug\Project2.exe (process 7556) exited with code 0.
Press any key to close this window . . .
```

Task 2: Code + Output

```
8   .code
9   main PROC
10  call sum3
11
12  exit
13  main ENDP
14
15
16  sum1 PROC
17
18  mov esi,0
19  mov ecx,lengthof arr1
20  mov eax, 0
21  l1:
22  add eax, arr1[esi*type arr1]
23  inc esi
24  loop l1
25
26  call writedec
27  call crlf
```

```
Microsoft Visual Studio Debug Console
15
40
55

C:\Users\acer\source\repos\Project2\Debug\Project2.exe (process 10744) exited with code 0.
Press any key to close this window . . .
```

Task 3: Code + Output

```
loop l3

continue:
mov ecx,count2
    l2:
    mov var2,ecx
    mov eax,'*'
    call writechar
    mov ecx,var2
    loop l2
    inc count2

call crlf
mov ecx,var1
dec count3
loop l1
```

```
Enter the number of columns:5
    *
   **
  ***
 ****
*****

C:\Users\acer\source\repos\Project2\Debug\Project2.exe (process 6616) exited with code 0.
Press any key to close this window . . .
```

LAB 8

Task 4: Code + Output

```
53
54  continue:
55  mov ecx,count2
56      l2:
57      mov var2,ecx
58      movzx eax,str2[esi*type str2]
59      call writechar
60      mov ecx,var2
61      inc esi
62      loop l2
63      inc count2
64
65  call crlf
66  mov ecx,var1
67  dec count3
68  loop l1
69
70  ret
71  PatternBuild ENDP
72
```

```
Enter the number of columns:5
    A
   BC
  DEF
 GHIJ
KLMNO

C:\Users\acer\source\repos\Project2\Debug\Project2.exe (process 17844) exited with code 0.
Press any key to close this window . . .
```

Task 5: Code + Output

```
22  call readdec
23  mov count1,eax
24
25  ret
26  GettingInput ENDP
27
28
29  funPrint PROC
30  mov ecx, count1
31  mov eax,1
32  l1:
33  call writedec
34  call crlf
35  inc eax
36
37  loop l1
38
39  ret
40  funPrint ENDP
41
42
43  end main
```

```
Microsoft Visual Studio Debug Console
Enter the number of elements:3
1
2
3

C:\Users\acer\source\repos\Project2\Debug\Project2.exe (process 5800) exited with code 0.
Press any key to close this window . . .
```