**LAB 08**

**STACK, IT’S OPERATION AND NESTED**

**PROCEDURES**





EL

2003

COMP ORG & ASSEMBLY

LANGUAGE

LAB

LABORATORY MANUAL



Fall 2021



Instructor:

Aamir Ali

, Aashir Mehboob, Amin Sadiq, Qurat

-

ul

-

Ain, M Kariz, Rabia

Syed Muhammad Faheem 20K-1054 3E

STUDENT NAME ROLL NO SEC

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

SIGNATURE & DATE

# MARKS AWARDED: \_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES**

**(NUCES), KARACHI**

Version: 1.0

Prepared by: Aashir Mahboob th

Date: 27 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

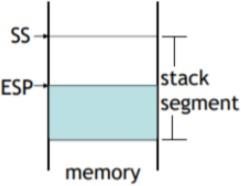
* To learn about user defined procedures and to use related Instructions
* Undersatnding the Nested Procedures and the way those are implemented in assembly

**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.

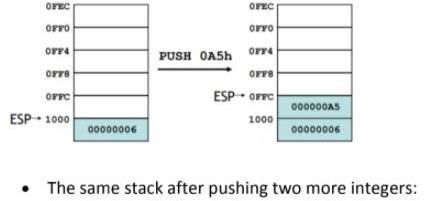
**Runtime Stack:**

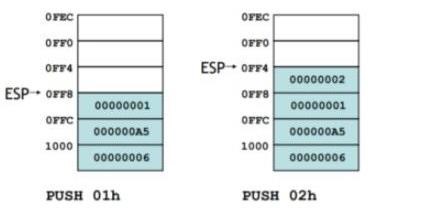
* Managed by the CPU, using two registers
* SS (stack segment)
* ESP (stack pointer): point the last value to be added to, or pushed on, the top of stack usually modified by instructions: ***CALL, RET, PUSH and POP***



**Push Operation**

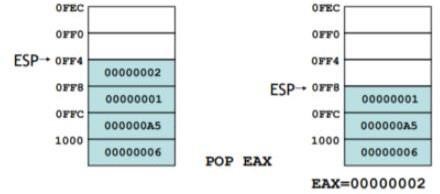
A 32-bit push operation decrements the stack pointer by 4 and copies a value into the location in the stack pointed to by the stack pointer.





**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

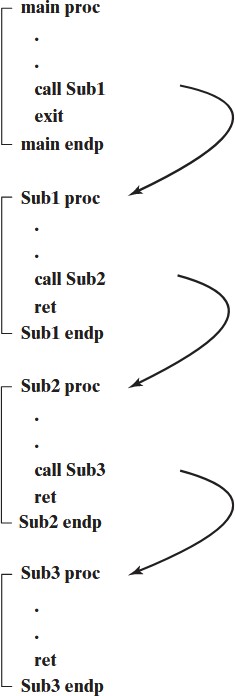
ret

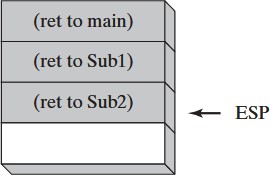
DisplaySum ENDP

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**

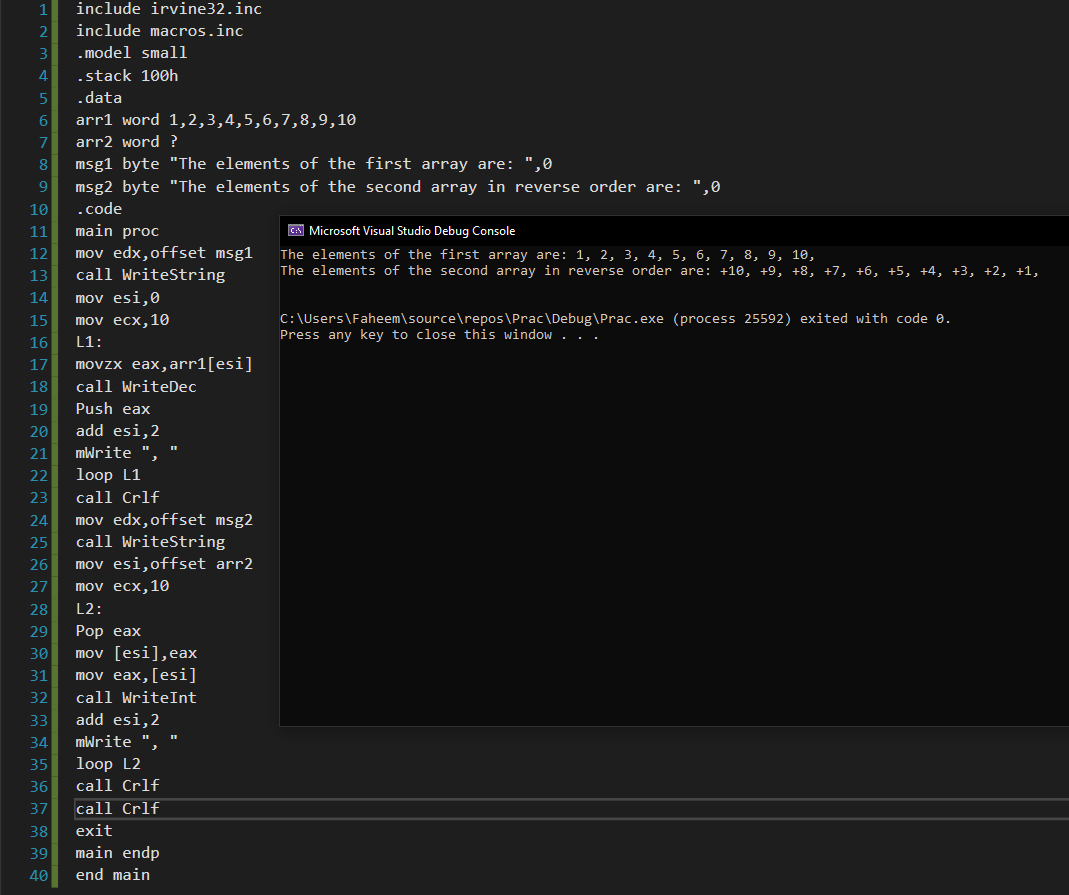
**DEF**

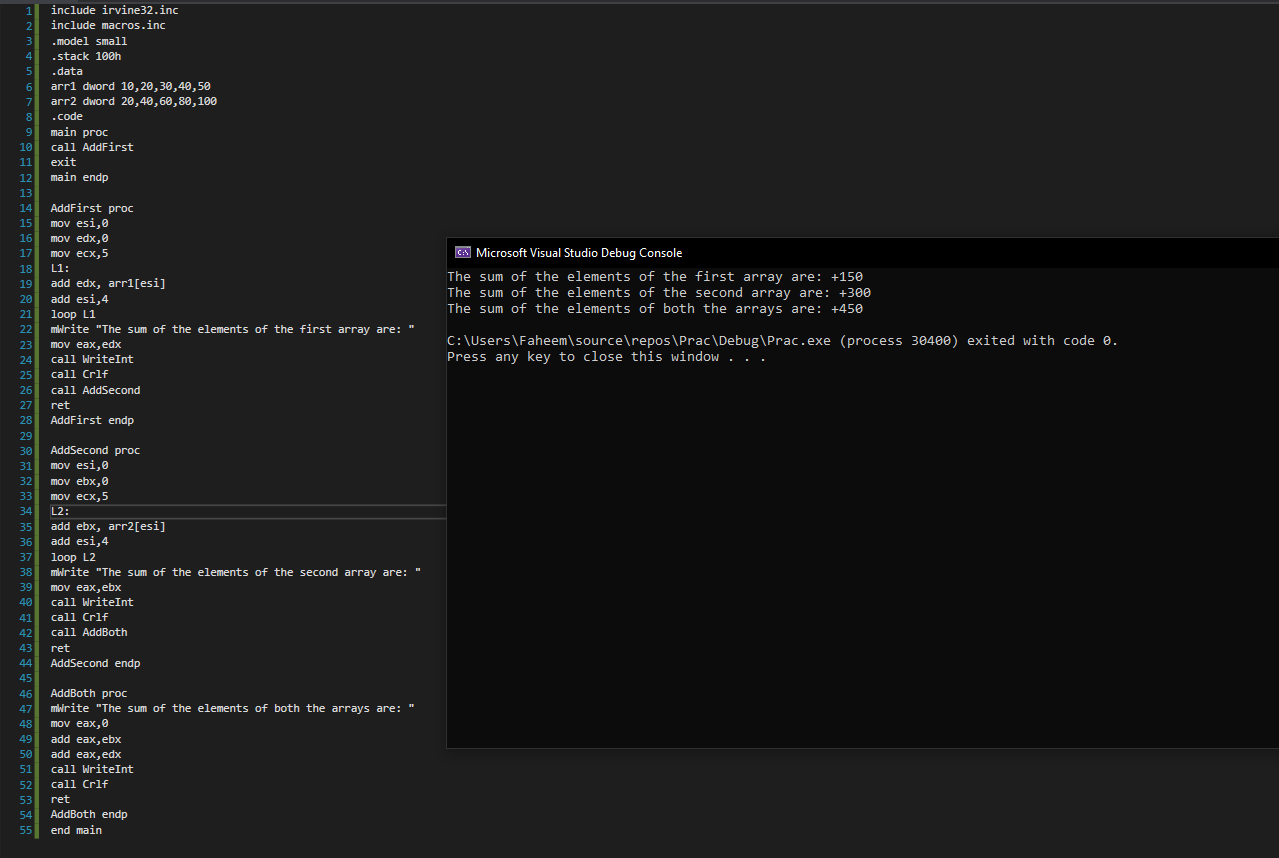
**GHIJ**

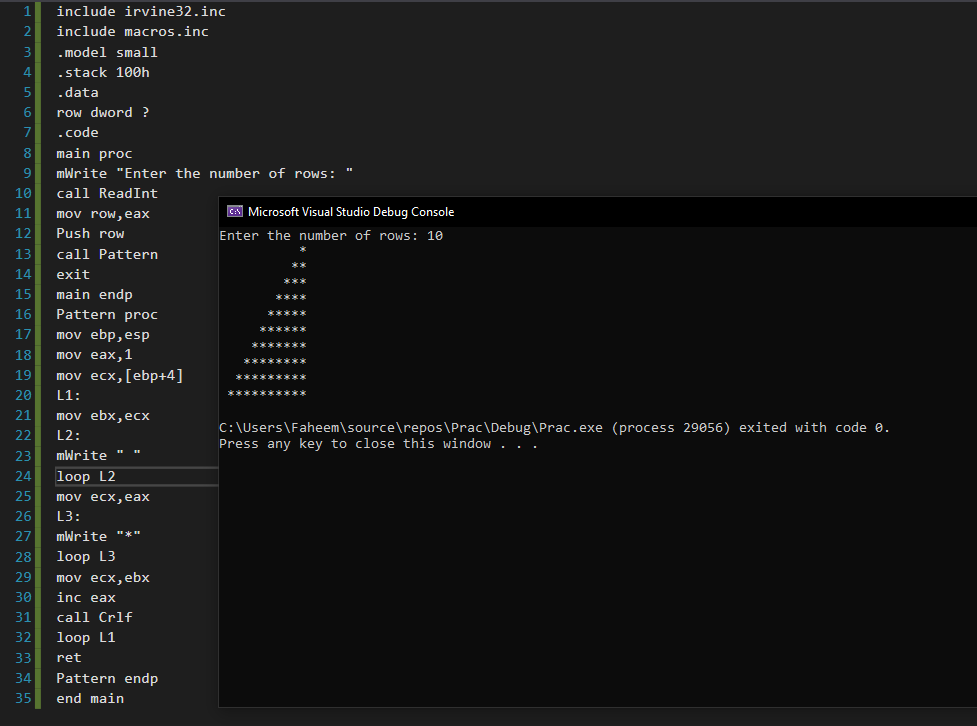
**KLMN**

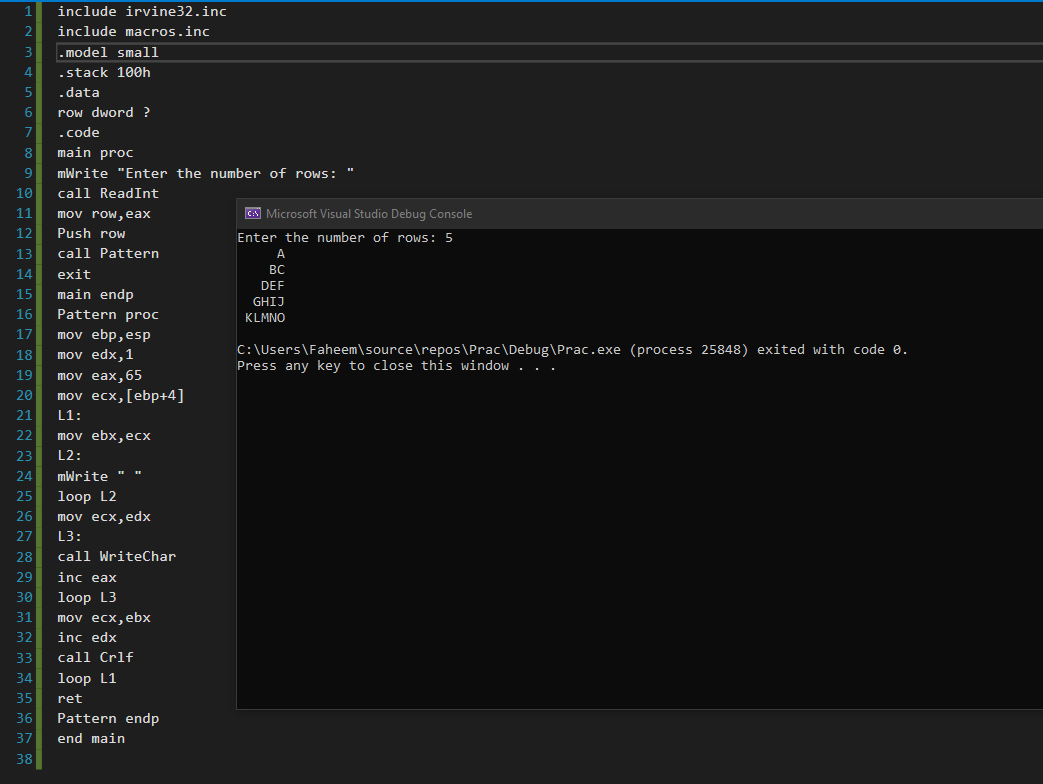
### Task#5

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

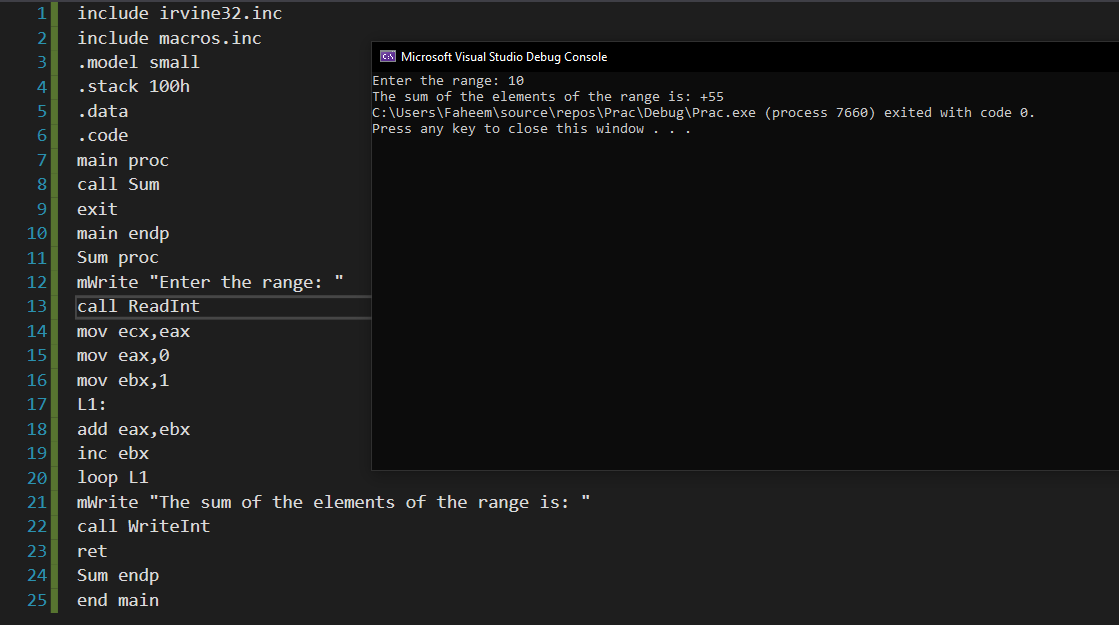
**Task 1: **

**Task 2: **

**Task 3: **

**Task 4: **

**Task 5:**

****