Computer Organization & Assembly Language

Lab 07
Stack Operations & Procedures

Instructors:

NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES

LAB 07

Learning Objectives

a. Runtime Stack
b. Push instruction
c. Pop instruction
d. PROC Directive
e. Call & Ret Instructions
f. Nested Procedures

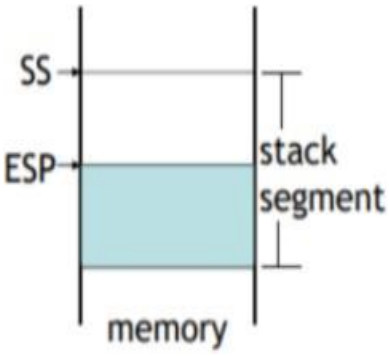
Stack:

- 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:

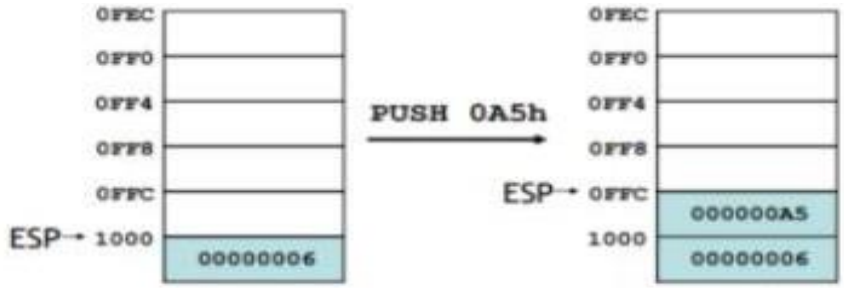
Managed by the CPU, using two registers

- 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

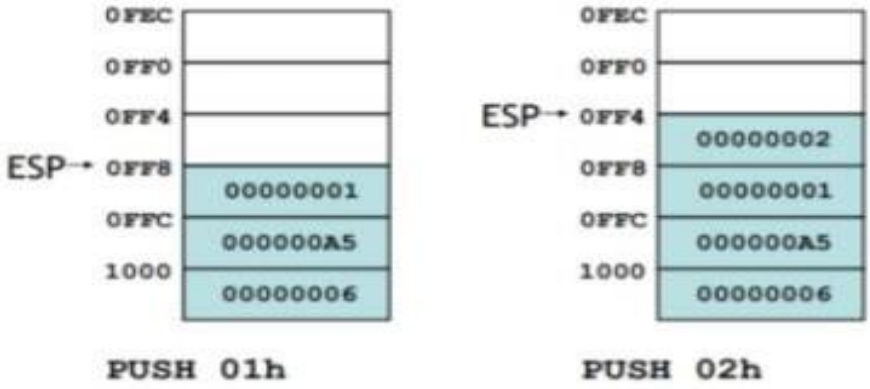


Push Operation

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.

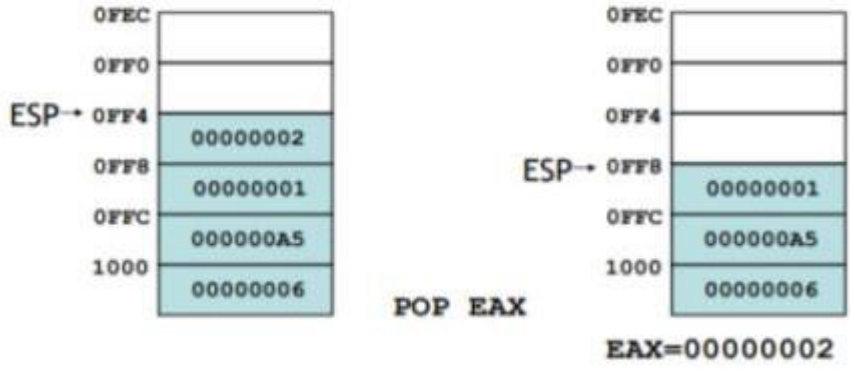


- The same stack after pushing two more integers:



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: Stack and nested loops.

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

Example: displays the product of three integers through a stack

```
TITLE Reversing a String (Prod.asm)
INCLUDE Irvine32.inc
.data
multp DWORD 2
.code
     main PROC
          mov eax, 1
          mov ecx, 3
          L1:
               PUSH multp
               ADD multp, 2
          LOOP L1
          mov ecx, 3
          L2:
```

```
                POP ebx
                MUL ebx    ;eax value multiply
            LOOP L2

            CALL DumpRegs
            EXIT
      main ENDP
 END main
```

Example: To find the largest number through a stack

```
.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.

PROC Directive

We can define a *procedure* as a named block of statements that ends in a return statement. A procedure is declared using the PROC and ENDP directives. It must be assigned a name (a valid identifier). When we create a procedure other than your program's startup procedure, end it with a RET instruction. RET forces the CPU to return to the location from where the procedure was called:
Let's say, ***sample*** is the name of procedure.

```
sample PROC
     .
     .
     ret
sample ENDP
```

The procedure is called from another function by using the CALL instruction. The CALL instruction should have the name of the called procedure as an argument as shown below

CALL Sample

The called procedure returns the control to the calling procedure by using the RET instruction.

Call & RET Instructions:

CALL instruction is used whenever we need to make a call to some procedure or a subprogram. Whenever a CALL is made, the following process takes place inside the microprocessor:

- The address of the next instruction that exists in the caller program (after the program CALL instruction) is stored in the stack.
- The instruction queue is emptied for accommodating the instructions of the procedure. Then, the contents of the instruction pointer (IP) is changed with the address of the first instruction of the procedure.
- The subsequent instructions of the procedure are stored in the instruction queue for execution.
- The Syntax for the CALL instruction is mentioned above.

RET instruction stands for return. This instruction is used at the end of the procedures or the subprograms. This instruction transfers the execution to the caller program.
Whenever the RET instruction is called, the following process takes place inside the microprocessor:

- The address of the next instruction in the mainline program which was previously stored inside the stack is now again fetched and is placed inside the instruction pointer (IP).
- The instruction queue will now again be filled with the subsequent instructions of the mainline program.

Example:

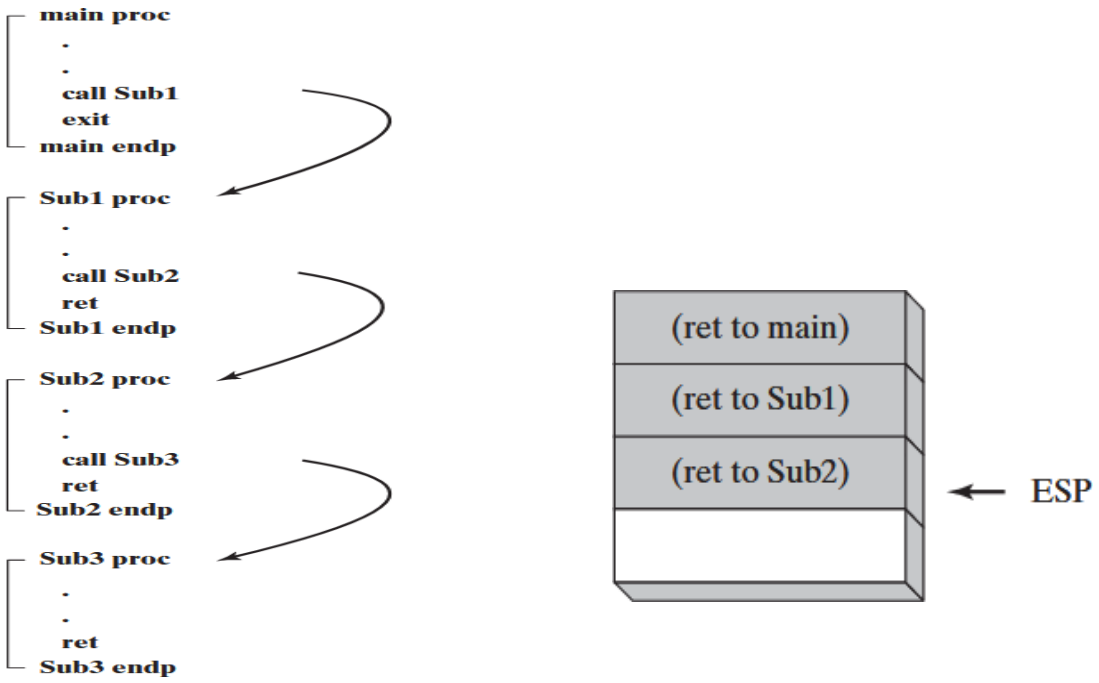
```
INCLUDE Irvine32.inc
.data
var1 DWORD 5
var2 DWORD 6
.code
main PROC

call AddTwo
call writeint
call crlf
exit
main ENDP

AddTwo PROC
mov eax,var1
mov ebx,var2
add eax,var2
ret
AddTwo ENDP
END main
```

Example:

```
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 PROC USES ecx edx esi
mov edx,OFFSET str1        ; "Enter a signed integer"
L1: call 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 PROC USES esi ecx
mov eax,0             ; set the 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 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:

```
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

END main
```

Exercise: Run on IDE

Task#1:
Take an array of 10 numbers move word-type of data into another empty array using stack push and pop technique.

Task#2
Write a program which displays the addition of three integers through a stack.

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

Task#4
Print the following pattern using a function call in which number of columns is pass through a variable.

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

Task#5
Print the following pattern using a function call in which number of columns is pass through a variable.

```
   A
  BC
 DEF
GHIJ
KLMN
```

Task#6
Write a function that asks the user for a number n and prints the sum of the numbers 1 to n