



Week 5

Chapter 5: Stack and Procedures

Class 13

Chapter Overview

- Stack Operations
- Defining and Using Procedures
- Linking to an External Library
- The Irvine32 Library
- 64-Bit Assembly Programming

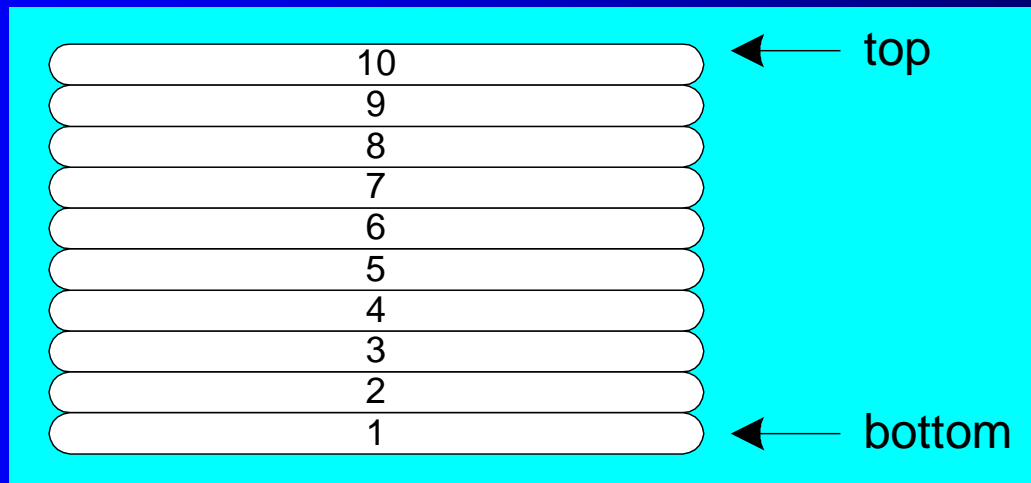
From Book's Page No 140

Stack Operations

- Runtime Stack
- PUSH Operation
- POP Operation
- PUSH and POP Instructions
- Using PUSH and POP
- Example: Reversing a String
- Related Instructions

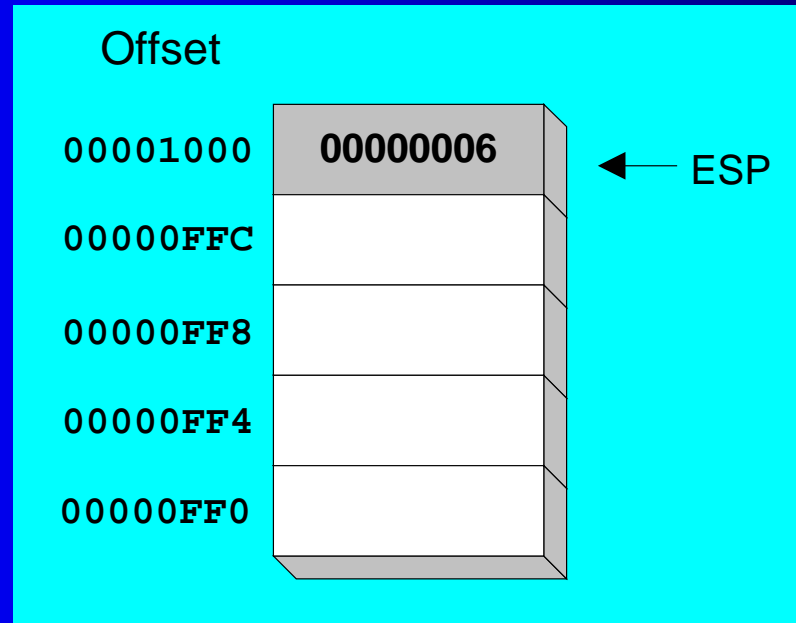
Runtime Stack

- Imagine a stack of plates . . .
 - plates are only added to the top
 - plates are only removed from the top
 - LIFO structure



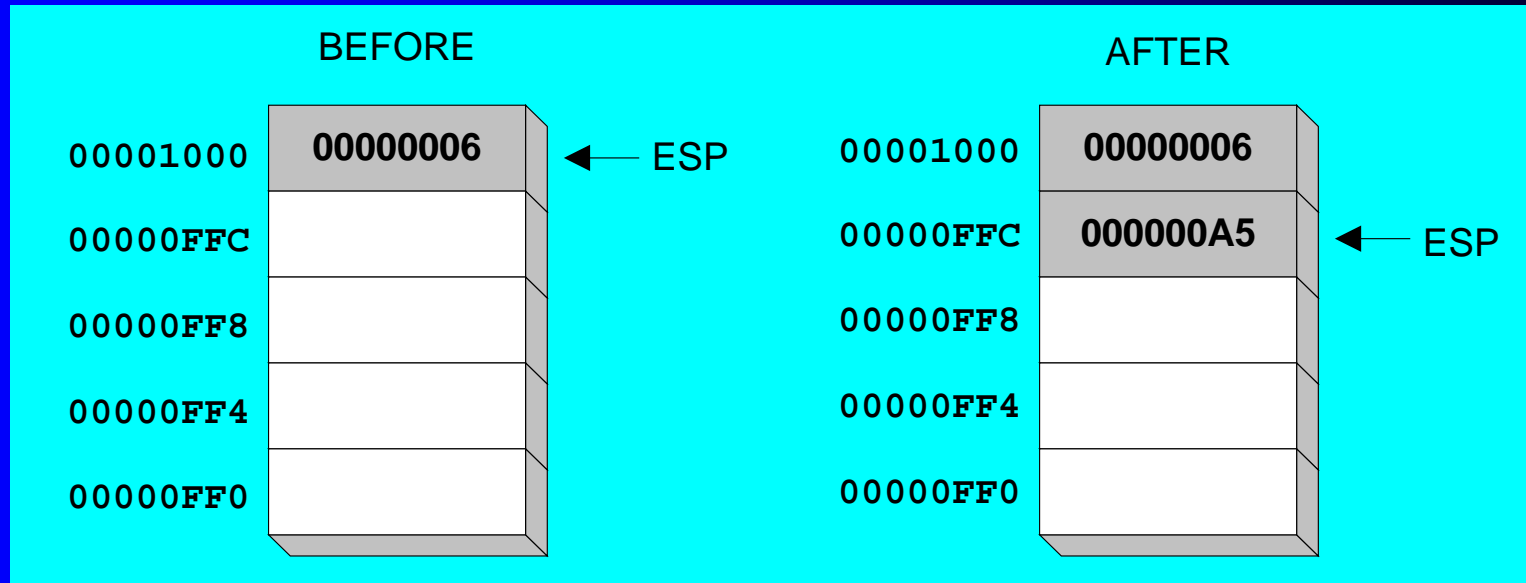
Runtime Stack

- Managed by the CPU, using two registers
 - SS (stack segment)
 - ESP (stack pointer)



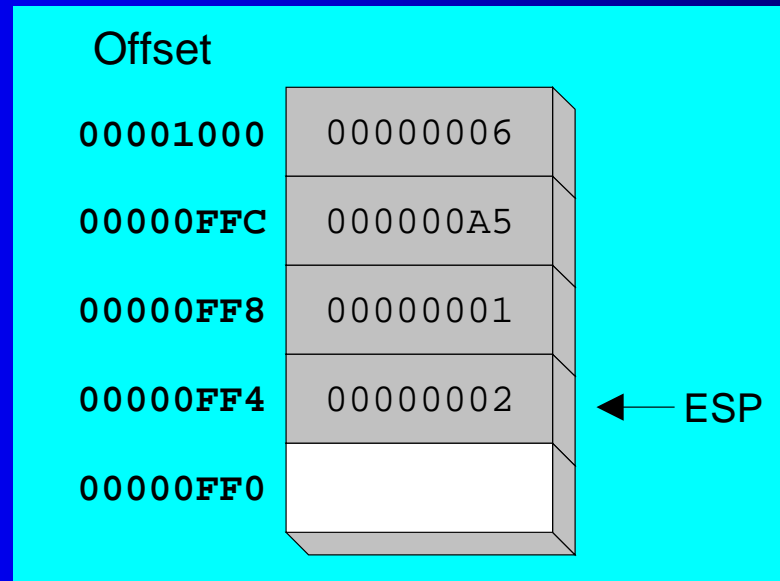
PUSH Operation (1 of 2)

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



PUSH Operation (2 of 2)

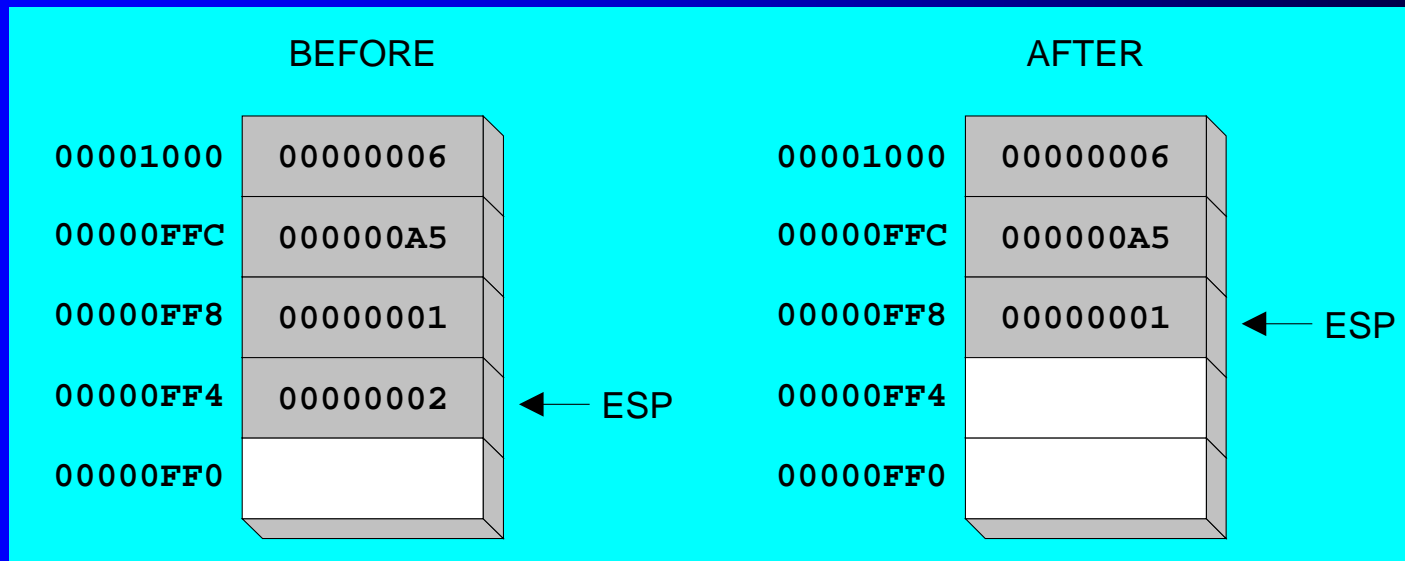
- Same stack after pushing two more integers:



The stack grows downward. The area below ESP is always available (unless the stack has overflowed).

POP Operation

- Copies value at stack[ESP] into a register or variable.
- Adds n to ESP, where n is either 2 or 4.
 - value of n depends on the attribute of the operand receiving the data



PUSH and POP Instructions

- PUSH syntax:
 - PUSH *r/m16*
 - PUSH *r/m32*
 - PUSH *imm32*
- POP syntax:
 - POP *r/m16*
 - POP *r/m32*

Using PUSH and POP

Save and restore registers when they contain important values. PUSH and POP instructions occur in the opposite order.

```
push esi                ; push registers
push ecx
push ebx

mov esi,OFFSET dwordVal ; display some memory
mov ecx,LENGTHOF dwordVal
mov ebx,TYPE dwordVal
call DumpMem

pop ebx                ; restore registers
pop ecx
pop esi
```

Example: Nested Loop

When creating a nested loop, push the outer loop counter before entering the inner loop:

```
    mov ecx,100          ; set outer loop count
L1:          ; begin the outer loop
    push ecx             ; save outer loop count

    mov ecx,20           ; set inner loop count
L2:          ; begin the inner loop
    ;
    ;
    loop L2              ; repeat the inner loop

    pop ecx              ; restore outer loop count
    loop L1              ; repeat the outer loop
```

Example: Reversing a String

- Use a loop with indexed addressing
- Push each character on the stack
- Start at the beginning of the string, pop the stack in reverse order, insert each character back into the string
- Q: Why must each character be put in EAX before it is pushed?

Because only word (16-bit) or doubleword (32-bit) values can be pushed on the stack.

- Using the String Reverse program as a starting point,
- #1: Modify the program so the user can input a string containing between 1 and 50 characters.
- #2: Modify the program so it inputs a list of 32-bit integers from the user, and then displays the integers in reverse order.

Related Instructions

- PUSHFD and POPFD
 - push and pop the EFLAGS register
- PUSHAD pushes the 32-bit general-purpose registers on the stack
 - order: EAX, ECX, EDX, EBX, ESP, EBP, ESI, EDI
- POPAD pops the same registers off the stack in reverse order
 - PUSHA and POPA do the same for 16-bit registers

- Write a program that does the following:
 - Assigns integer values to EAX, EBX, ECX, EDX, ESI, and EDI
 - Uses PUSHAD to push the general-purpose registers on the stack
 - Using a loop, your program should pop each integer from the stack and display it on the screen