Foundation for Advancement of Science & Technology

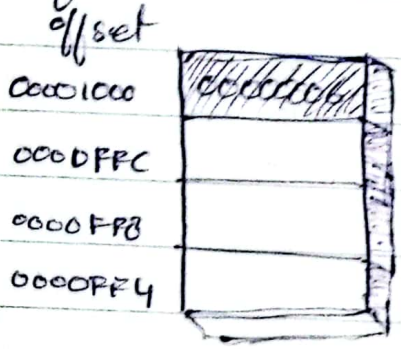
— STACK —

Runtime Stack is managed directly by CPU, using ESP register known as Stack Pointer Register.

→ The ESP hold 32-bit offset into some location on the stack.

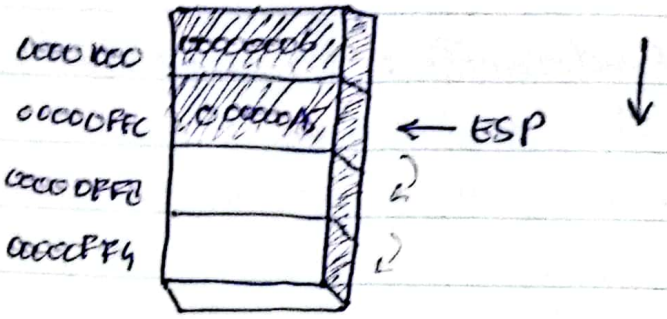
→ ESP is indirectly modified by instr.

→ ESP always points to the last val to be added, or pushed on. top of stack.



▲ Runtime stack grows downward in memory from higher addresses to lower addresses

Push: A [32-bit] push operation "decrements" the stack point by "4" & copies value into the loc in stack pointed to by the stack pointer.

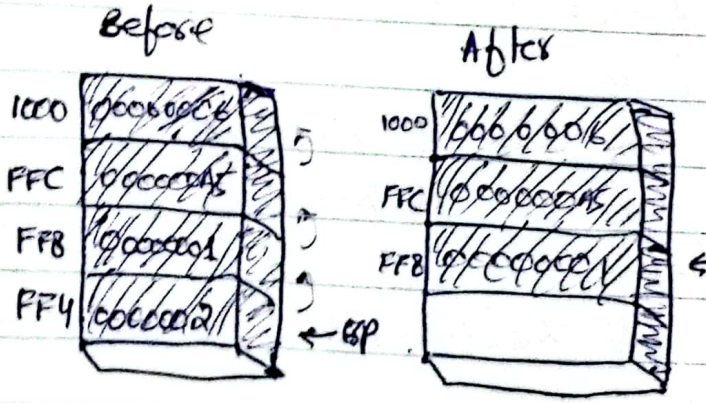


▲ After inserting A5 in stack

push reg/mem16
push reg/mem32
push imm32

→ A [16-bit] opesand causes the stack pointer to be decremented by "2".

Pop: Removes value from stack & increments to point to the next highest location in stack.



▲ after poping 0000 0002

Foundation for Advancement of Science & Technology

→ The area below ESP is logically empty & will be overwritten when push is called next time.

```
pop   reg/mem16
pop   reg/mem32
```

→ If operand 16-bit → inc by 2
→ If operand 32-bit → inc by 4

PUSHFD & POPFD:
Pushfd pushes 32-bit EFLAGS register on the stack
→ We cannot use mov to copy flags to variab so we use Pushfd.
→ Popfd pops the stack into EFLAGS.

```
pushfd        ; save flags
  .
  .
popfd         ; restore the flags
```

```
.data
  saveFlags DWORD ?
.code
  pushfd              ; push flags on stack
  pop   saveFlags     ; copy into variable

  pushfd saveflags    ; push saveflag val.
  popfd               ; copy into the flags
```

PUSHAD:
The pushad instr. pushes all of 32-bit general purpose register on stack. in following order:
EAX, ECX, EDX, EBX, ESP (value before executing PUSHAD), EBP, ESI & EDI.

POPAD:
Popad pops the same register off the stack in reverse order

PUSHA:
Push 16-bit general purpose register in order
AX, CX, DX, BX, SP, BP, SI, D
POPA: Pops out in reverse order

Foundation for Advancement of Science & Technology

PROC DIRECTIVE

Defining a procedure,

```
Name PROC        ; Name -> any valid identifier
  ⋮
  ret            ; -> this is necessary & This forces the CPU to return to the location from where procedure was called.
Name ENDP
```

```
MySub PROC
    pushad       ;save gen purpose reg.
    ⋮
    mov eax --
    mov edx --
    mov ecx --
    ⋮
    popad        ;restore values.
    ret
MySub ENDP
```

→ hum kisi function se phele apni register ki values save kr sakte hain or jab sab kaam hojayega to wapis restore krlenge.

LABELS:

→ the labels in procedure ~~are~~ have a scope within the procedure.

jump destination
↳ this destination label must be inside the function.

→ It is possible to work around this limitation by declaring a ~~global~~ global label, identified by a double colon (::) after its name.

→ Not good idea to jump or loop out of the current procedure

```
→ ReadVal PROC
    pushad
    ⋮
    mov eax, return_value
    ⋮
    popad      → ab jo humne returned eax mein save krii thi wo overwrite hochuki hai popad ki waja se to ab ghalat value recieve hogi.
    ret
ReadVal ENDP
```

→ REVERSE STRING Example in BOOK pg# 162

FAST
Foundation for Advancement of Science & Technology

SUM OF THREE INTEGERS:

Assuming that relevant integers are assigned to eax, ebx, & ecx before procedure is called. The sum is returned in eax.

```
Sum PROC
    add eax, ebx
    add eax, ecx
    return
Sum ENDP
```

CALL & RET Example:

Suppose in 'main' a CALL statement is located at offset 00000020. And after this next instr. is at 00000025:

```
    main PROC
00000020    call MySub
00000025    mov eax, ebx.

    MySub PROC
00000040    mov eax, edx
            ret
    MySub ENDP
```

→ ab jab CALL instr. execute hogi to CALL k baad wali instr. stack mein Push hojayegi. OR ab hamara ESP 00000025 ko point krrha ha.

→ OR instruction pointer EIP 00000040 ko point krega jo function k andar wali statement hai.

→ Jab function ka "return" call hoga to ESP se 00000025 pop out ho kr EIP mein load hoga or execute hojayega.

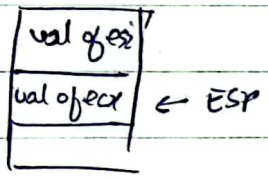
SUMMING AN ARRAY.

```
.data
    arr DWORD 10000h, 20000h, 30000h, 40000h, 50000
    Sum DWORD ?
.code
    main PROC
        mov esi, OFFSET arr ;esi points to arr
        mov ecx, LENGTHOF arr
        call ArraySum
        mov Sum, eax        ;returned in eax
    end main ENDP
```

; anything which we want to recieve as an argument we will pass it in register before function call.

```
ArraySum PROC
; we recive offset of arr in esi
; no. of elements in ecx

push esi        ; Saving esi & ecx
push ecx        ; in stack.
mov eax,0
```




```
L1:
    add eax, [esi]
    add esi, TYPE DWORD
    Loop L1
```

last val in stack was of ecx so ecx was popout 1st.

```
    pop ecx     ; restoring values
    pop esi     ; of ecx & esi

    ret
ArraySum ENDP

END main.
```

USES Operator: USES operator lets you list the names of all registers modified within a procedure

→ It 1st save the value of register which we have name at beging of program.

→ 2nd ly it restores the values of registers at the End.

```
ArraySum PROC USES esi ecx
    :
    :
ArraySum ENP
```

ab push or pop waali instru. khud execute hojayegi