

PUSH and POP instructions

PUSH s32	ESP = ESP - 4 and transfers („pushes”) <s> in the stack (s – doubleword)	-
POP d32	Eliminates („pops”) the current element from the top of the stack and transfers it to d (d – doubleword) ; ESP = ESP + 4	-
Allowed as exceptions for backwards compatibility on 16 bits:		
PUSH s16	ESP = ESP - 2 and transfers („pushes”) <s> in the stack (s – word)	-
POP d16	Eliminates („pops”) the current element from the top of the stack and transfers it to d (d – word) ; ESP = ESP + 2	-
PUSHA / PUSHAD	Pushes in the stack EAX, ECX, EDX, EBX, ESP, EBP, ESI and EDI	
POPA / POPAD	Pops EDI, ESI, EBP, ESP, EBX, EDX, ECX and EAX from stack	
PUSHF	Pushes EFlags in the stack	
POPF	Pops the top of the stack and transfers it to Eflags	

In 32 bits programming the stack is organized on doublewords, so **d** and **s** MUST be doublewords. The stack grows from big addresses to small addresses, 4 bytes at a time, ESP pointing always to the doubleword from the top of the stack.

We can illustrate the way in which these instructions works, by using an equivalent sequence of MOV and ADD or SUB instructions:

push eax \Leftrightarrow sub esp, 4 ; prepare (allocate) space in order to store the value
mov [esp], eax ; store the value in the allocated space

pop eax \Leftrightarrow mov eax, [esp] ; load in eax the value from the top of the stack
add esp, 4 ; clear the location

However, because of the required backwards compatibility with the 16 bits programming, **PUSH** and **POP** with 16 bits operands are also allowed AS AN EXCEPTION !... but NOT recommended as a programming practice for the regular user/programmer !!

PUSH and **POP** only allow you to deposit and extract values represented by word and doubleword.

Thus, **PUSH AL** is not a valid instruction (syntax error !), because the operand is not allowed to be a byte value. On the other hand, the sequence of instructions

```
PUSH    ax ; push ax in the stack  
PUSH    ebx ; push ebx in the stack  
POP ecx ; ecx <- the doubleword from the top of the stack (the value of ebx)  
POP dx  ; dx <- the word from the stack (the value of ax)
```

is a valid sequence of instructions and is equivalent as an effect with:

```
MOV    ecx, ebx  
MOV    dx,  ax
```

In addition to this constraint (which is inherent in all x86 processors), the operating system requires that stack operations be made only through doublewords or multiple of doublewords accesses, for reasons of compatibility between user programs and the kernel and system libraries. The implication of this constraint is that the **PUSH operand16** or **POP operand16** instructions (for example, **PUSH word 10**), although supported by the processor and assembled successfully by the assembler, is not allowed by the operating system, might causing what is named the incorrectly aligned stack error: the stack is correctly aligned if and only if the value in the ESP register is permanently divisible by 4!

(to study at home – a very instructive example !!)

In the perspective of evaluating the effect of instructions such as **PUSH ESP** or **POP dword [ESP]**, the order in which the component (sub)operations of the PUSH and POP instructions are performed should be specified even more clearly:

- a). The **source operand of the instruction is evaluated** (ESP for PUSH and dword [ESP] respectively for POP)
- b). **ESP is updated accordingly** (ESP := ESP-4 for PUSH and ESP := ESP+4 for POP respectively)
- c). **The assignment involved in the effect of the instruction is performed on the destination operand** (the new top of the stack for PUSH and dword [ESP] respectively (this being now after the subtraction of ESP from b) the element immediately below the initial top of the stack - for POP)

Assuming that the initial situation is ESP = 0019FF74, after **PUSH ESP** we will have ESP = 0019FF70 and the contents of the top of the stack will now be 0019FF74.

Assuming that the initial situation is ESP = 0019FF74 and that in this location is the value 7741FA29 (the top of the stack), after **POP dword [ESP]** we will have ESP = 0019FF78 and the content of this location (the content of the location at the top of the stack) will be 7741FA29 (so we could say that "the top of the stack moves one position lower"!!).