Stack and Local Variables

CSC03B3

Outline

Outline

Working with the Stack

Working with the Stack

The stack is a globally allocated block of memory where you can store items. The stack on an Intel machine operates the same way as a stack data-structure: new items are placed on the top of the stack, items are removed from the top of the stack. The stack grows **DOWNWARDS** (from high memory to low memory).

Each item on the stack is a 1 byte memory location. If a **DWORD** is placed on the stack then 4 bytes are used.

There are two registers involved when using the stack:

ESP **The stack pointer** – Points to the top of the stack

EBP **The base pointer** – Points to the bottom of the stack

Critical Thinking!

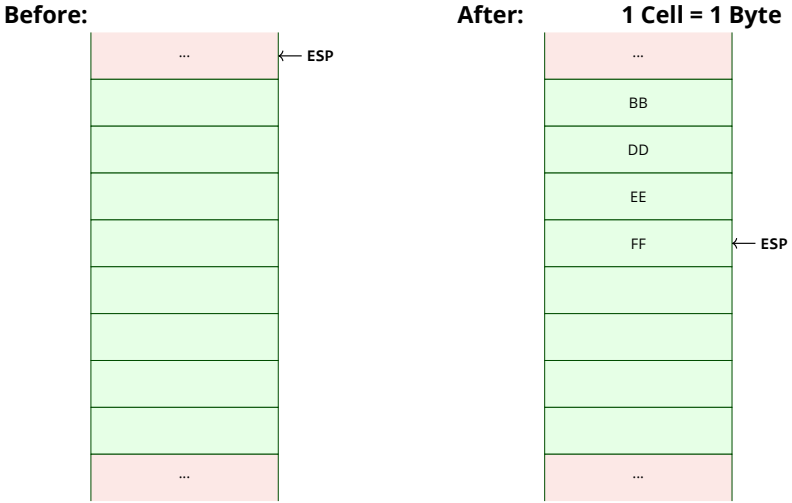
Always keep track of what the stack looks like! If you do not keep track of what the stack looks like, you **WILL** get confused. You *MUST* draw a picture of the stack for each function in your design!

Working with the Stack - Instructions

PUSH	**Push 32bit int onto stack** Places a 32bit int onto the top of the stack and decreases the value of **ESP** by 4.
Formats	
PUSH *reg*	**PUSH** *mem*
PUSH *imm*	
Flags Modified	
None	

POP	**Pop 32bit int from stack** Retrieves a 32bit int from the stack and stores it in the specified destination. **ESP** is increased by 4.
Formats	
POP *reg*	**POP** *mem*
Flags Modified	
None	

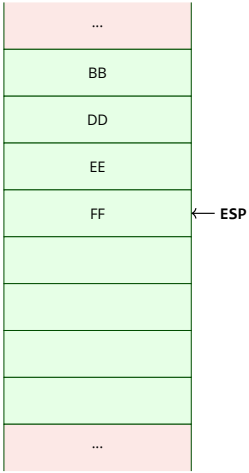
Stack Example



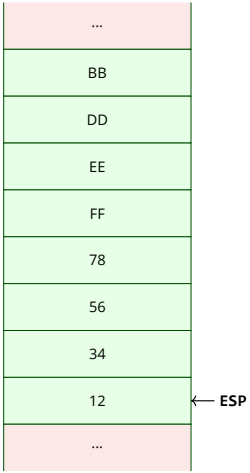

```
1 MOV   eax, FFEEDDBBh
2 PUSH  eax
```


Stack Example

Before:



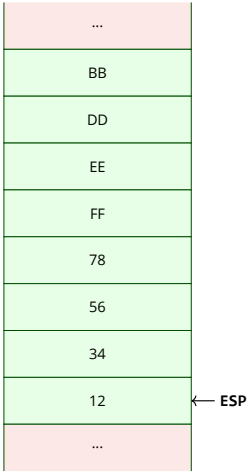
After:



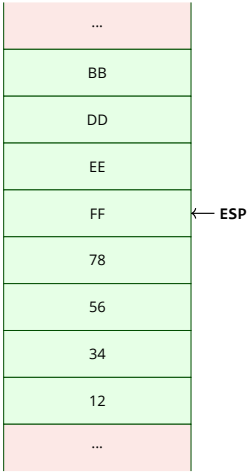

```
1 | PUSH  12345678h
```


Stack Example

Before:



After:

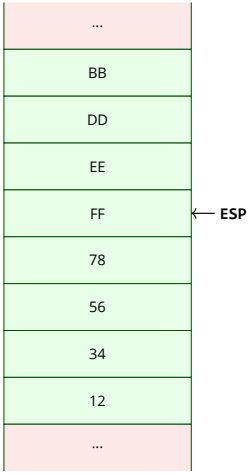



```
1 | POP  ebx
```

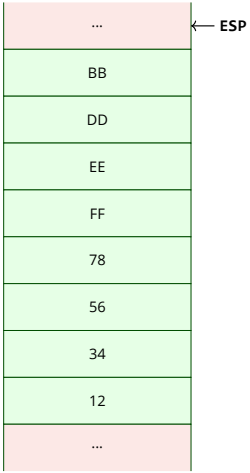
EBX: 12345678h

Stack Example

Before:



After:




```
1 | POP ecx
```

ECX: FFEEDDBBh

Local Variables

Local Variables

So far the programs which have been written have used *global variables*. Local variables avoids having to allocate memory locations ahead of time.

- Local variables are saved on the stack.
- Local variables do not have names.
- Local variables are referenced using indirect addressing relative to either the **ESP** or **EBP** register.

A ***stack frame*** is a region on the stack where **EBP** points to a known location. The location of **EBP** is not known when the application starts and therefore needs to be set up manually. Local variables are created in this stack frame.

- Set up the local stack frame.
- Reserve space on the stack frame for local variables.
- Reference local variables relative to **EBP**.
- Destroy the local stack frame.

Local Variables - Code

```
PUSH  ebp        ; Save Base Pointer
MOV    ebp,  esp  ; Create Stack Frame
```

```
; Allocate 1 DWORD on the stack
SUB ESP, 4  ; reserve 1 4-byte location on the stack
; Allocate 2 DWORDs on the stack
SUB ESP, 8  ; reserve 2 4-byte locations on the stack
```

```
; Reserved 1 DWORD, access it
MOV  [EBP-4], X  ; move X into the memory location

; Reserved 2 DWORDs, access 2nd DWORD
MOV  [EBP-8], Y  ; move Y into the memory location

; Reserved 3 DWORDs, access 3rd DWORD
MOV  [EBP-12], Z  ; move Z into the memory location
```

```
MOV  esp,  ebp  ; Destroy your stack frame
POP ebp        ; Restore the Base Pointer
```


Local Variables Example I

Calculate the sum of two user provided numbers using only local variables.

```
1  ; BLOCK Create Stack Frame
2  PUSH  ebp          ; Save EBP
3  MOV   ebp, esp  ;  Create stack frame
4  SUB   esp, 8    ; reserve 2 DWORDS on stack
5
6  ; BLOCK Input
7  INVOKE  InputInt
8  MOV   [ebp-4], eax  ; Put user value into local1
9  INVOKE  InputInt
10 MOV   [ebp-8], eax  ; Put user value into local2
11
12 ; BLOCK Calculations
13 MOV   ebx, [ebp-4]    ; ebx = local1
14 ADD   ebx, [ebp-8]    ; ebx += local2
15 INVOKE  OutputInt, ebx
16
17 ; BLOCK Cleanup Stack Frame
18 MOV   esp, ebp  ; destroy stack frame
19 POP   ebp          ; restore ebp
```


Local Variables Example II