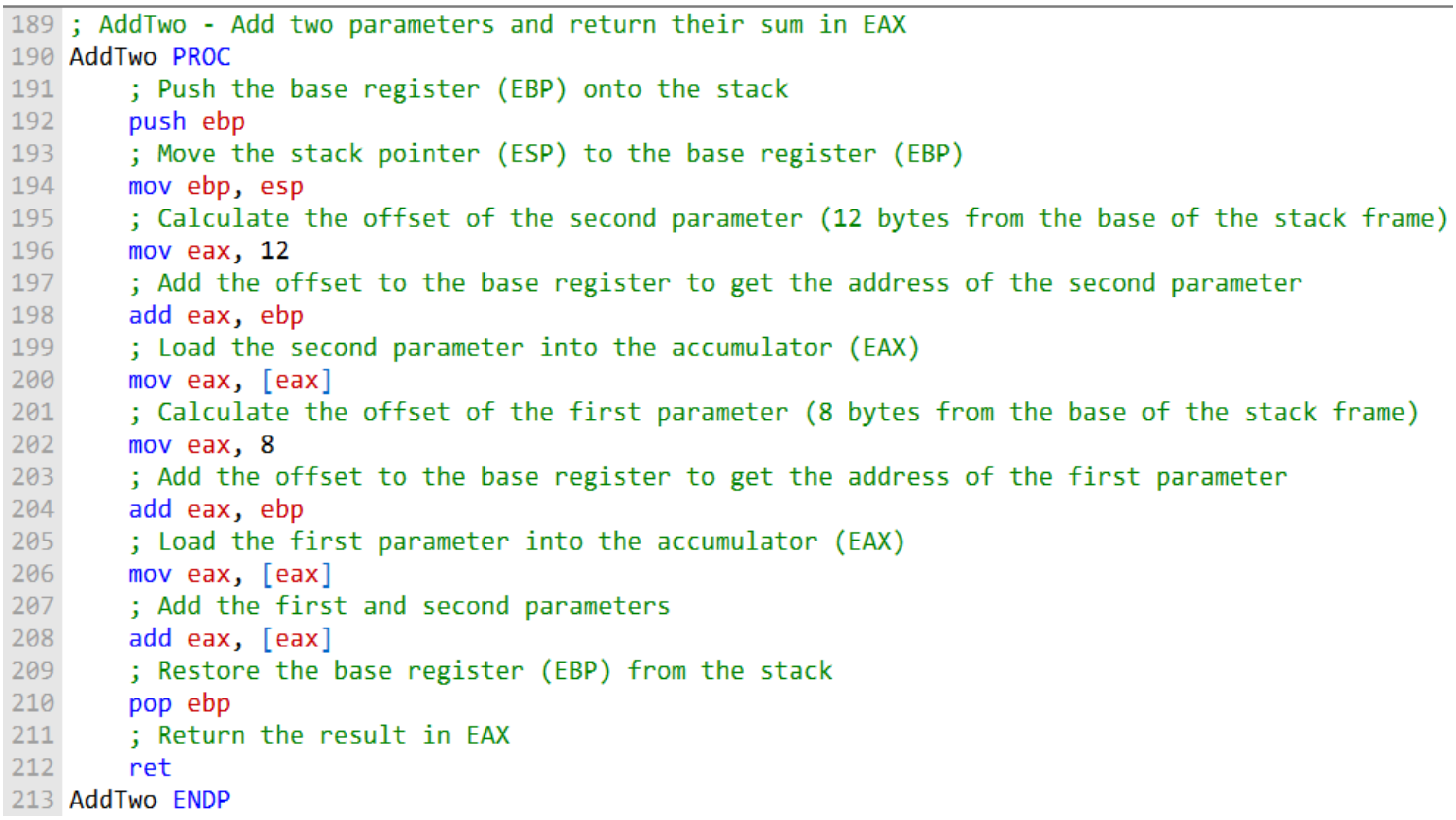
# *Base Offset Addressing*

The following code is a rewritten and explained implementation of AddTwo using base-offset addressing to access stack parameters:



```
; AddTwo - Add two parameters and return their sum in EAX
AddTwo PROC
    ; Push the base register (EBP) onto the stack
    push ebp
    ; Move the stack pointer (ESP) to the base register (EBP)
    mov ebp, esp
    ; Calculate the offset of the second parameter (12 bytes from the base of the stack frame)
    mov eax, 12
    ; Add the offset to the base register to get the address of the second parameter
    add eax, ebp
    ; Load the second parameter into the accumulator (EAX)
    mov eax, [eax]
    ; Calculate the offset of the first parameter (8 bytes from the base of the stack frame)
    mov eax, 8
    ; Add the offset to the base register to get the address of the first parameter
    add eax, ebp
    ; Load the first parameter into the accumulator (EAX)
    mov eax, [eax]
    ; Add the first and second parameters
    add eax, [eax]
    ; Restore the base register (EBP) from the stack
    pop ebp
    ; Return the result in EAX
    ret
AddTwo ENDP
```

## *Explanation:*

The first instruction, **push ebp**, saves the base register (EBP) onto the stack. This is important because EBP will be used as the base register for accessing stack parameters.

The next instruction, **mov ebp, esp**, moves the stack pointer (ESP) to the base register (EBP). This effectively sets the base of the stack frame.

The next two instructions, **mov eax, 12** and **add eax, ebp,** calculate the offset of the second parameter. The second parameter is located 12 bytes from the base of the stack frame.

The next instruction, **mov eax, [eax]**, loads the second parameter into the accumulator (EAX).

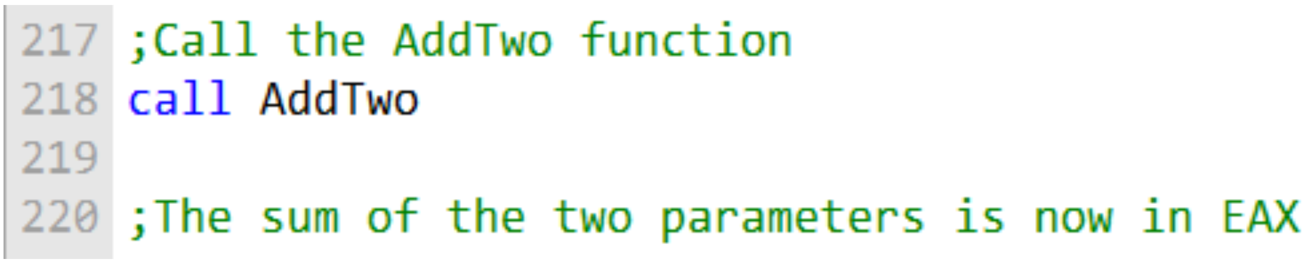
The next two instructions, **mov eax, 8** and **add eax, ebp**, calculate the offset of the first parameter. The first parameter is located 8 bytes from the base of the stack frame.

The next instruction, **mov eax, [eax]**, loads the first parameter into the accumulator (EAX).

The next instruction, **add eax, [eax]**, adds the first and second parameters.

The next instruction, **pop ebp**, restores the base register (EBP) from the stack.

This is important because we need to restore the base register before returning from the function. The final instruction, **ret**, returns from the function. Example usage:



```
;Call the AddTwo function
call AddTwo

;The sum of the two parameters is now in EAX
```

## ***Benefits of using base-offset addressing:***

Base-offset addressing is efficient because it allows us to access stack parameters without having to calculate their absolute addresses.

Base-offset addressing is also **flexible** because it allows us to access stack parameters relative to the base of the stack frame.

This means that we can easily move the stack frame around without having to update the code that accesses stack parameters.

Image 1:

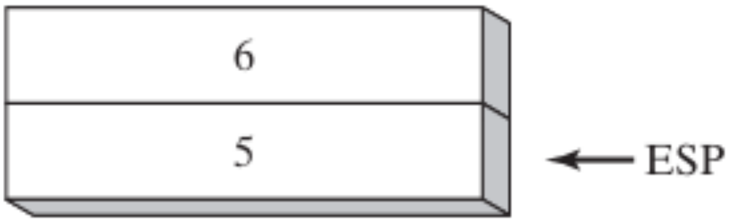
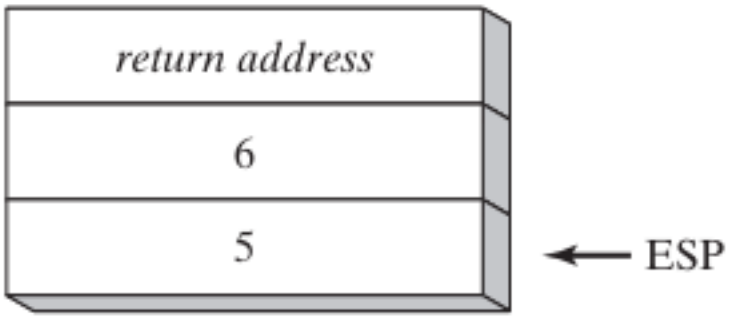


Image 2:


*return address*
6
5
ESP