Local Variables

The C++ function MySub() declares two local variables, X and Y. When this function is compiled into machine language, the following assembly code is generated:

```
void MySub()
{
    int X = 10;
    int Y = 20;
}
```

```
MySub PROC
    push ebp
    mov ebp, esp
    sub esp, 8 ; create locals
    mov DWORD PTR [ebp-4], 10 ; X
    mov DWORD PTR [ebp-8], 20 ; Y
    mov esp, ebp ; remove locals from stack
    pop ebp
    ret
MySub ENDP
```

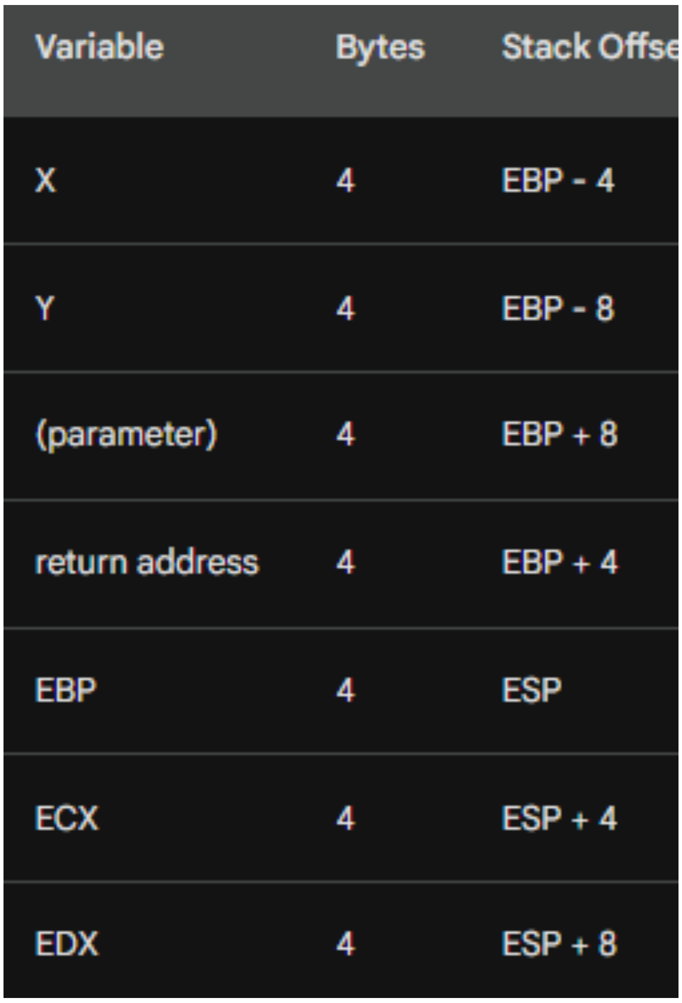
This assembly code shows how local variables are allocated on the stack. The push ebp and mov ebp, esp instructions save the current value of the EBP register onto the stack. The EBP register is used to reference the stack frame for the current function.

The sub esp, 8 instruction allocates 8 bytes on the stack for the two local variables. This is because each stack entry defaults to 32 bits, and each variable's storage size is rounded upward to a multiple of 4.

The mov DWORD PTR [ebp-4], 10 and mov DWORD PTR [ebp-8], 20 instructions initialize the local variables X and Y to the values 10 and 20, respectively.

When MySub() is finished executing, the mov esp, ebp and pop ebp instructions remove the local variables from the stack and restore the previous value of the EBP register.

The image you sent shows the stack frame for the MySub() function. The following table shows the contents of the stack frame:



Variable	Bytes	Stack Offse
X	4	EBP - 4
Y	4	EBP - 8
(parameter)	4	EBP + 8
return address	4	EBP + 4
EBP	4	ESP
ECX	4	ESP + 4
EDX	4	ESP + 8

The first column in the table shows the name of the variable. The second column shows the number of bytes that the variable occupies in memory. The third column shows the stack offset of the variable. The stack offset is the distance from the base pointer (EBP) to the variable.

The X and Y variables are local variables. This means that they are created and destroyed within the current function. Local variables are allocated on the stack. In the image you sent, the X variable is located at stack offset -4, and the Y variable is located at stack offset -8.

The (parameter) variable is a parameter to the current function. Parameters are passed to functions by pushing them onto the stack before the function is called. In the image above, the parameter is located at stack offset 8.

The return address is the address of the instruction that will be executed after the current function returns. The return address is pushed onto the stack by the caller before the caller calls the current function. In the image you sent, the return address is located at stack offset 4.

The EBP register is the base pointer register. The base pointer register is used to reference the stack frame for the current function. In the image above, the EBP register is located at the same address as the ESP register.

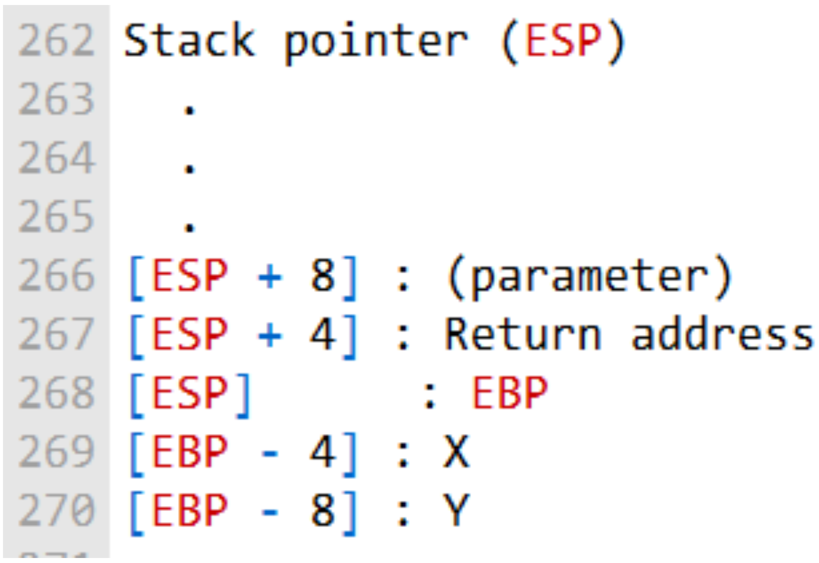
The ECX and EDX registers are **callee-saved registers.** This means that the caller is responsible for saving and restoring the values of these registers before and after calling the current function. In the image you sent, the ECX and EDX registers are located at stack offsets 4 and 8, respectively.

The stack offset of a variable is the distance from the base pointer (EBP) to the variable. In the case of the X and Y variables, their stack offsets are -4 and -8, respectively.

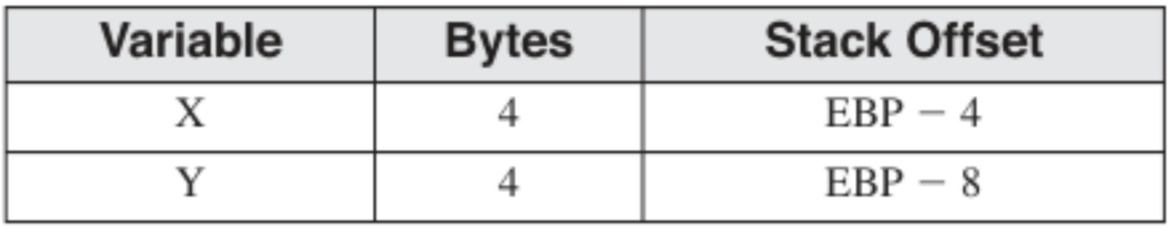
This means that the X variable is located 4 bytes below the EBP register, and the Y variable is located 8 bytes below the EBP register.

The stack grows downward in memory, so the stack offset of a variable is the distance from the base pointer (EBP) to the variable in bytes, going down the stack.

The **EBP register** is used to reference the stack frame for the current function. This means that the X and Y variables can be accessed by using the EBP register as a base pointer.


```
Stack pointer (ESP)
   .
   .
   .
[ESP + 8] : (parameter)
[ESP + 4] : Return address
[ESP]       : EBP
[EBP - 4] : X
[EBP - 8] : Y
```

The following diagram shows the stack frame for the MySub() function, with the stack offsets of the X and Y variables indicated: (you can use the image above or this one):



Variable	Bytes	Stack Offset
X	4	EBP – 4
Y	4	EBP – 8

The EBP register is used to reference the **stack frame for the current function.** This means that the X and Y variables can be accessed by using the EBP register as a base pointer. For example, to access the X variable, the following instruction would be used:

```
mov eax, [ebp-4]
```

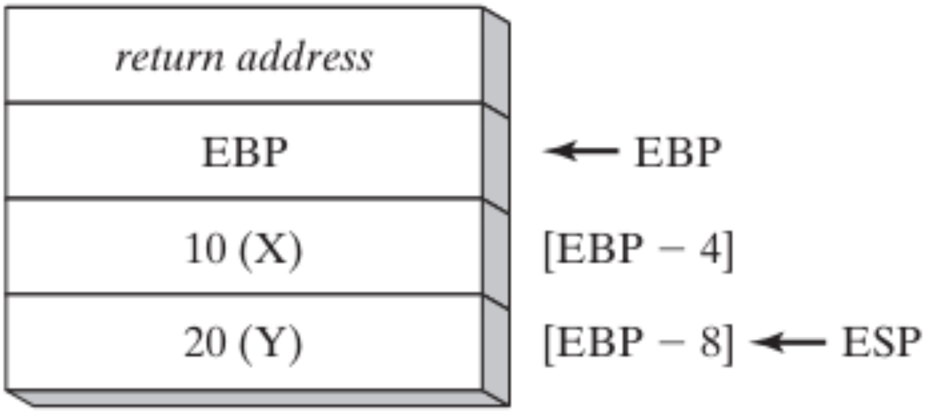
This instruction would copy the contents of the memory location at stack offset -4 into the EAX register.

The stack offset of a variable is also important for function calls. When a function is called, the caller pushes the function's parameters onto the stack.

The called function then allocates space on the stack for its local variables. The stack offset of a parameter is its distance from the EBP register in the caller's stack frame.

The stack offset of a local variable is its distance from the EBP register in the called function's stack frame.

Stack frame after creating local variables.



===================================

Local Variables Symbols

===================================

It is often useful to define a symbol for each local variable's offset. This can make the code easier to read and maintain.

To define a symbol for a local variable's offset, you use the **EQU directive.** For example, the following code defines a symbol for the X local variable:

```
X_local EQU DWORD PTR [ebp-4]
```

This symbol can then be used to access the X local variable, as shown in the following code:

```
MySub PROC
    push ebp
    mov ebp, esp
    sub esp, 8 ; create locals
    mov X_local, 10 ; X
    mov esp, ebp ; remove locals from stack
    pop ebp
    ret
MySub ENDP
```

The line of code **X_local EQU DWORD PTR [ebp-4]** defines a symbol called X_local. This symbol is equivalent to the value of the memory location at stack offset -4.

The stack offset of a variable is the distance from the base pointer (EBP) to the variable in bytes, going down the stack. In this case, the stack offset is -4, which means that the variable is located 4 bytes below the base pointer.

The DWORD PTR keyword specifies that the variable is a 32-bit DWORD (double word).

This symbol can then be used to access the local variable X, as shown above.

Using symbols to access local variables can make the code easier to read and maintain. For example, the following code is easier to read than the previous code:

```
MySub PROC
    push ebp
    mov ebp, esp
    sub esp, 8 ; create locals
    mov [ebp-4], 10 ; X
    mov esp, ebp ; remove locals from stack
    pop ebp
    ret
MySub ENDP
```