

# x86 Function Call Conventions



1

## Register use in the stack frame

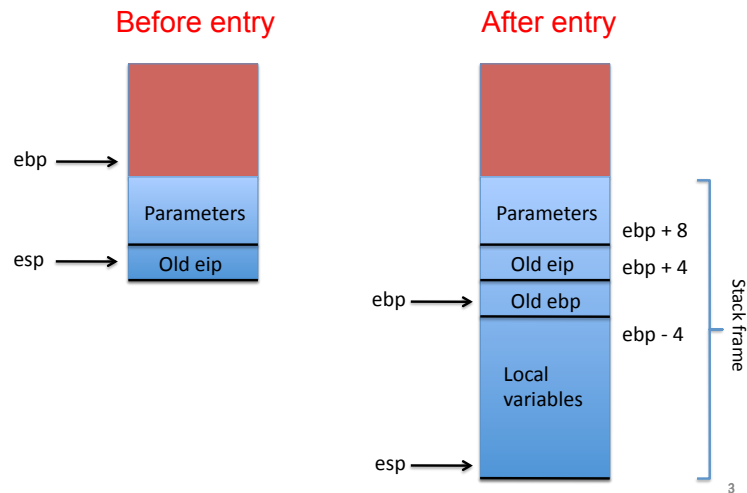
**%ESP** - Stack Pointer  
This 32-bit register always points to the last element used on the stack.

**%EBP** - Base Pointer  
This 32-bit register is used to reference all the function parameters and local variables in the current stack frame.

**%EIP** - Instruction Pointer  
This holds the address of the next CPU instruction to be executed. It is saved onto the stack as part of the CALL instruction.

2

## Stack frame



## Calling a function

1. Push parameters onto the stack, from right to left.
2. Call the function. The contents of the %EIP (instruction pointer) is pushed onto the stack. It points to the first byte *after* the CALL instruction.

4

## Executing a function

3. Save and update the %ebp.

```
pushl %ebp
movl %esp, %ebp
```

4. Save CPU registers used for temporaries.

5. Allocate local variables.

6. Perform the function's purpose.  
Store return value, if any, in %eax.

7. Restore saved CPU registers.

8. Release local storage.  
`movl %ebp, %esp`

9. Restore the old base pointer.  
`popl %ebp`

} leave

10. Return from the function.  
`ret`

11. Clean up pushed parameters.  
The *caller* must clean up the parameters pushed onto the stack.

5

6

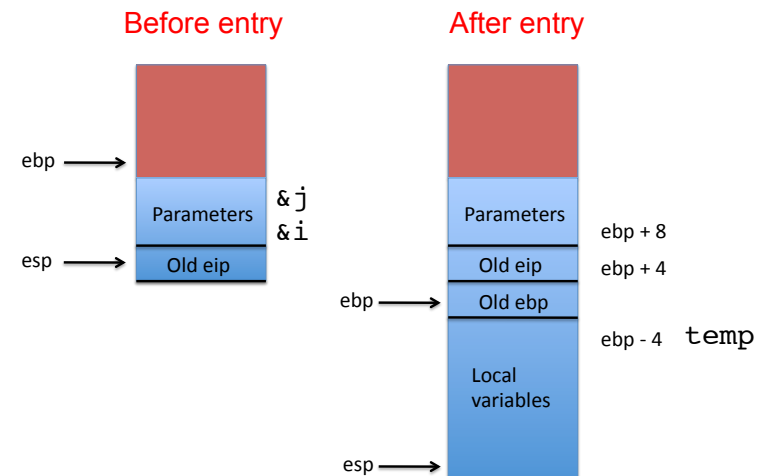
## An example

```
int main() {
    int i = 7;
    int j = 13;
    swap(&i, &j);
}

void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}
```

7

## Stack frame



8