Function and its Memory Representation in Stack.

Consider a function:

```
int add(int p, int q){
  int a = 10;
  int b = 20;
return (p − a) + (q − b);
}
add(40, 50)
```

Explanation:

Now, add(40, 50) is function call (callee) that carries an address.

And this address is the section where we will be returning the return value.

Such addresses are known as reurn addresses.

Q) How this address is made?

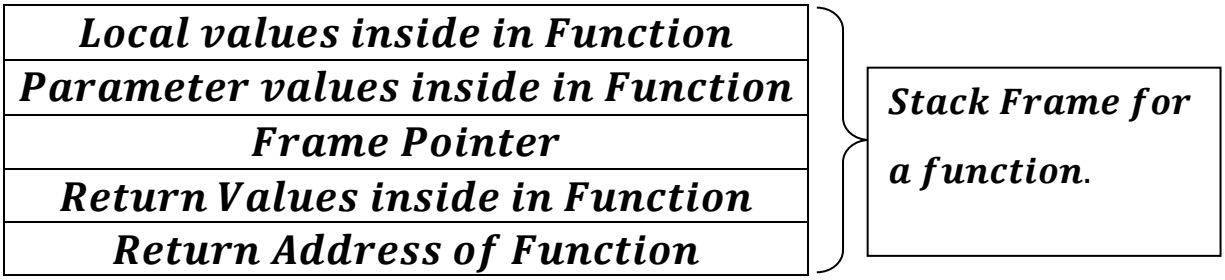
Explanation:

Before to start , Program Counter is register which holds address to which next instruction to be executed. And this register is associated with the CPU. CPU will push the current value of the Program Counter (PC) to the stack frame.
And the value is known as `return address`.

STACK FRAME

Stack Frame is mainly composed of : Frame Pointer, Local variables of the function, Parameter of the function, Return value of the function, and Return address of the callee.

Note , a function also may have contain: Exception Handling , hence stack frame also contain a section of Exception Handling , Buffer and Registers.
But let exclude this things focus on main functionary.



Memory of Stack Frame of a Function.

***Say, Frame Pointer is* 0*x*000018 *and Program Counter has value i.e. considered address of function call is* 0*x*000021.**

0*x*000021

Program Counter

And with the help of CPU , Program Counter pushes the address*: 0*x*000021 (*considered to be return address*)*to the Stack Frame. This operation will occur as soon as the function call will take place.

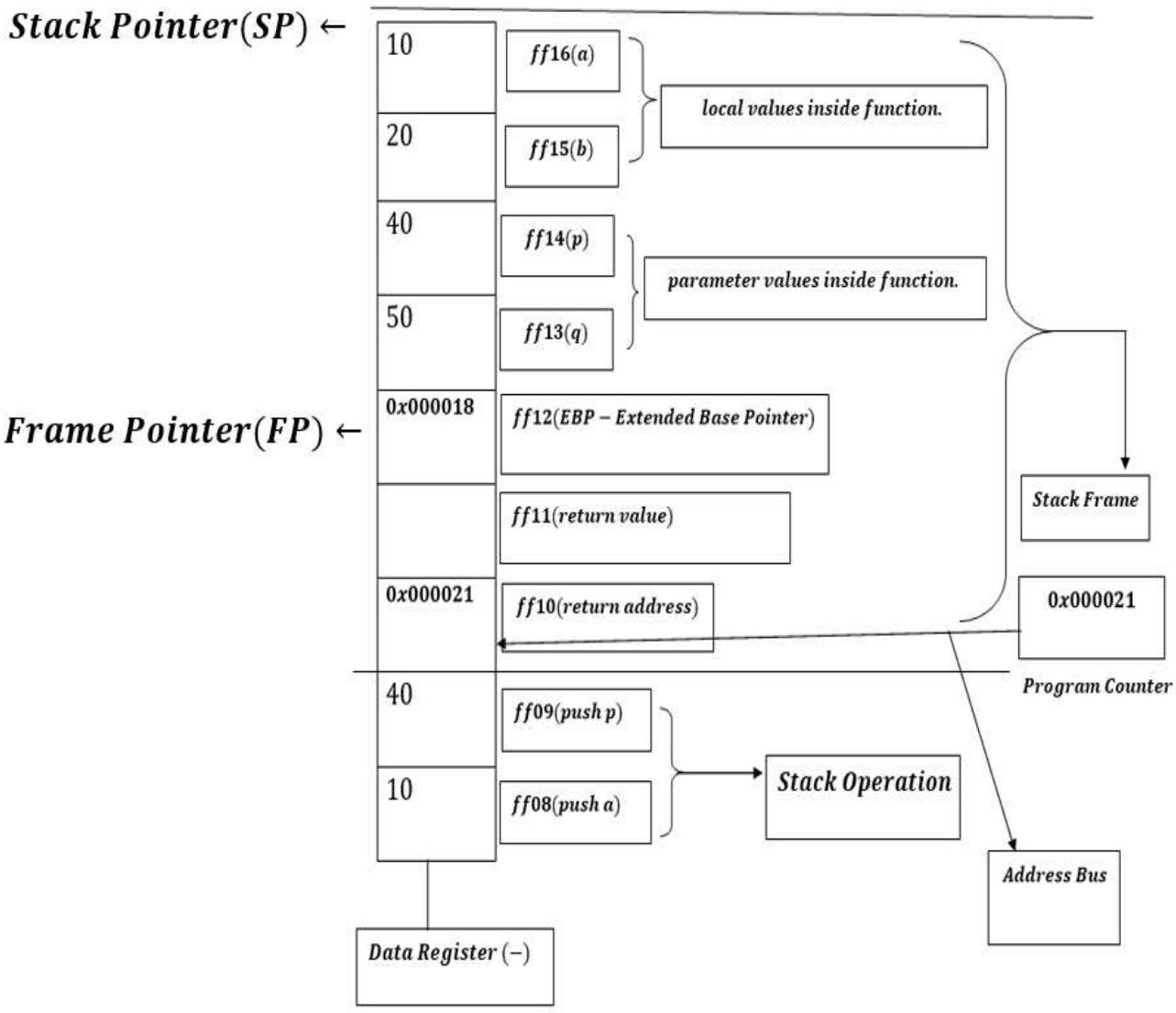
***Now, to we have to note of two new terms*:**

1) *ESP* (*Extended Stack Pointer*)*is another name of stack pointer*.

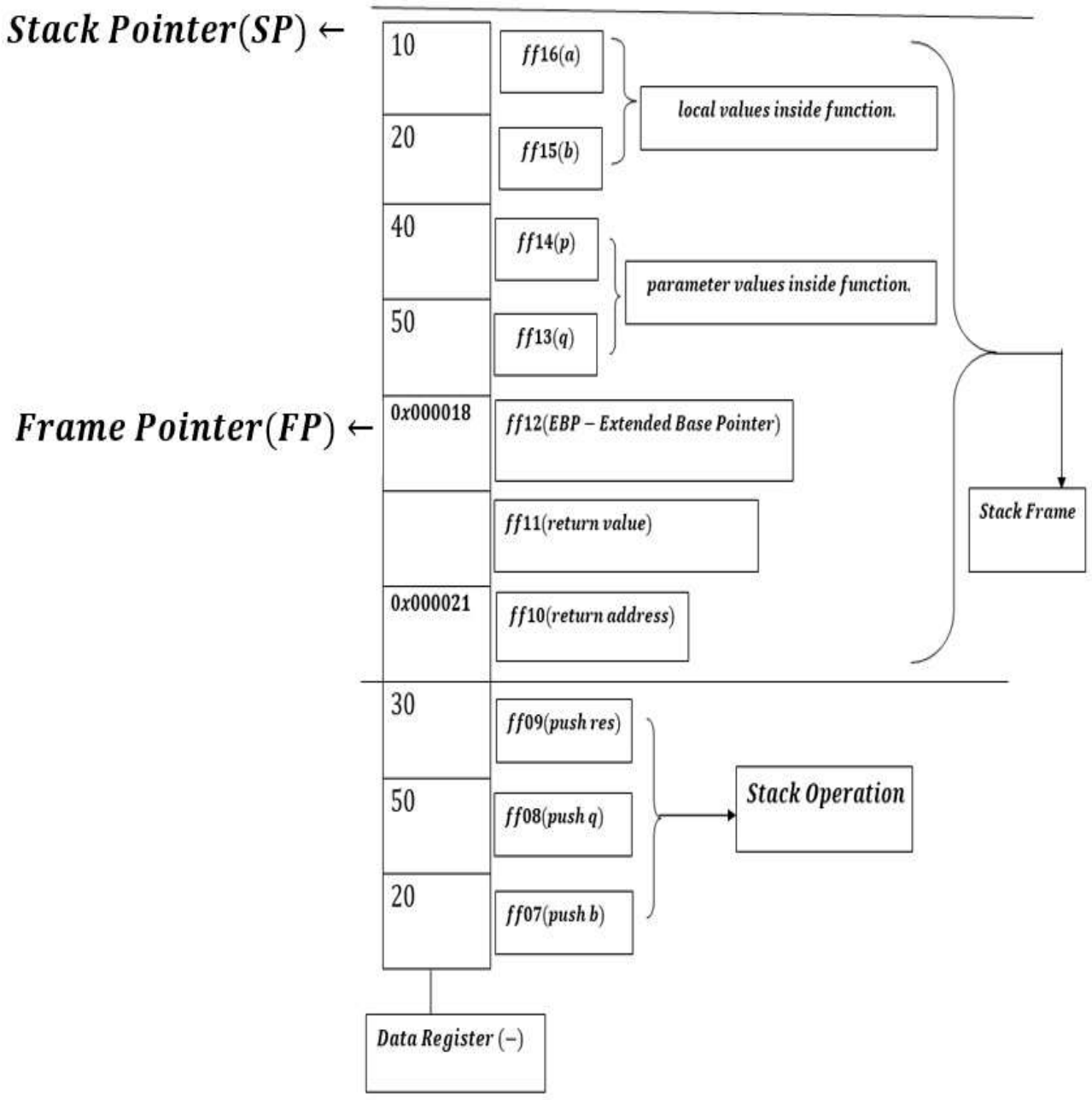
2)EBP i.e. Extended Base Pointer another name of Frame Pointer, which act as a fixed reference point and this reference point allows local variables, formal parameter variables etc. to be stored within the stack frame.

Now coming back to the above example we have :

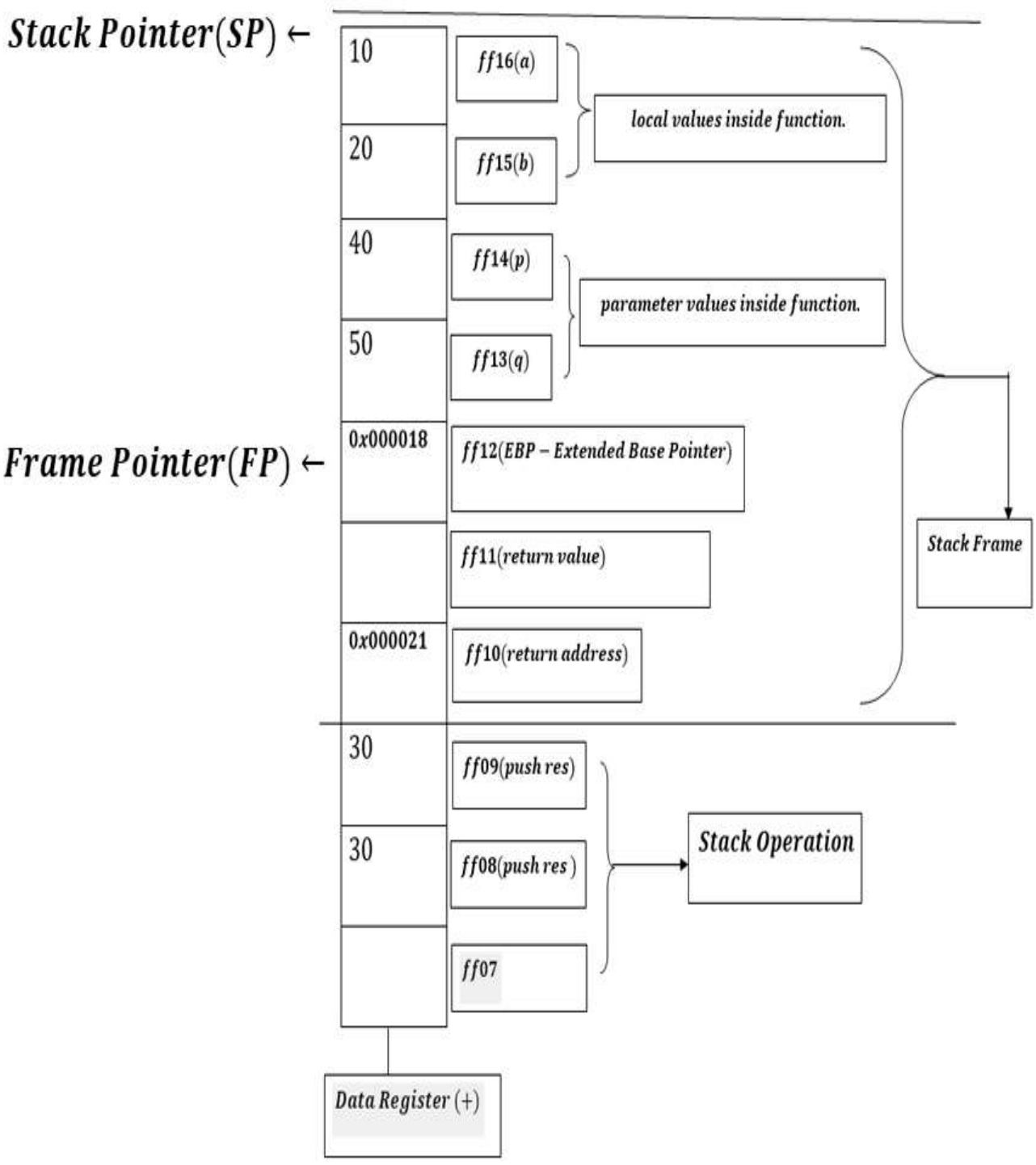
1st Step



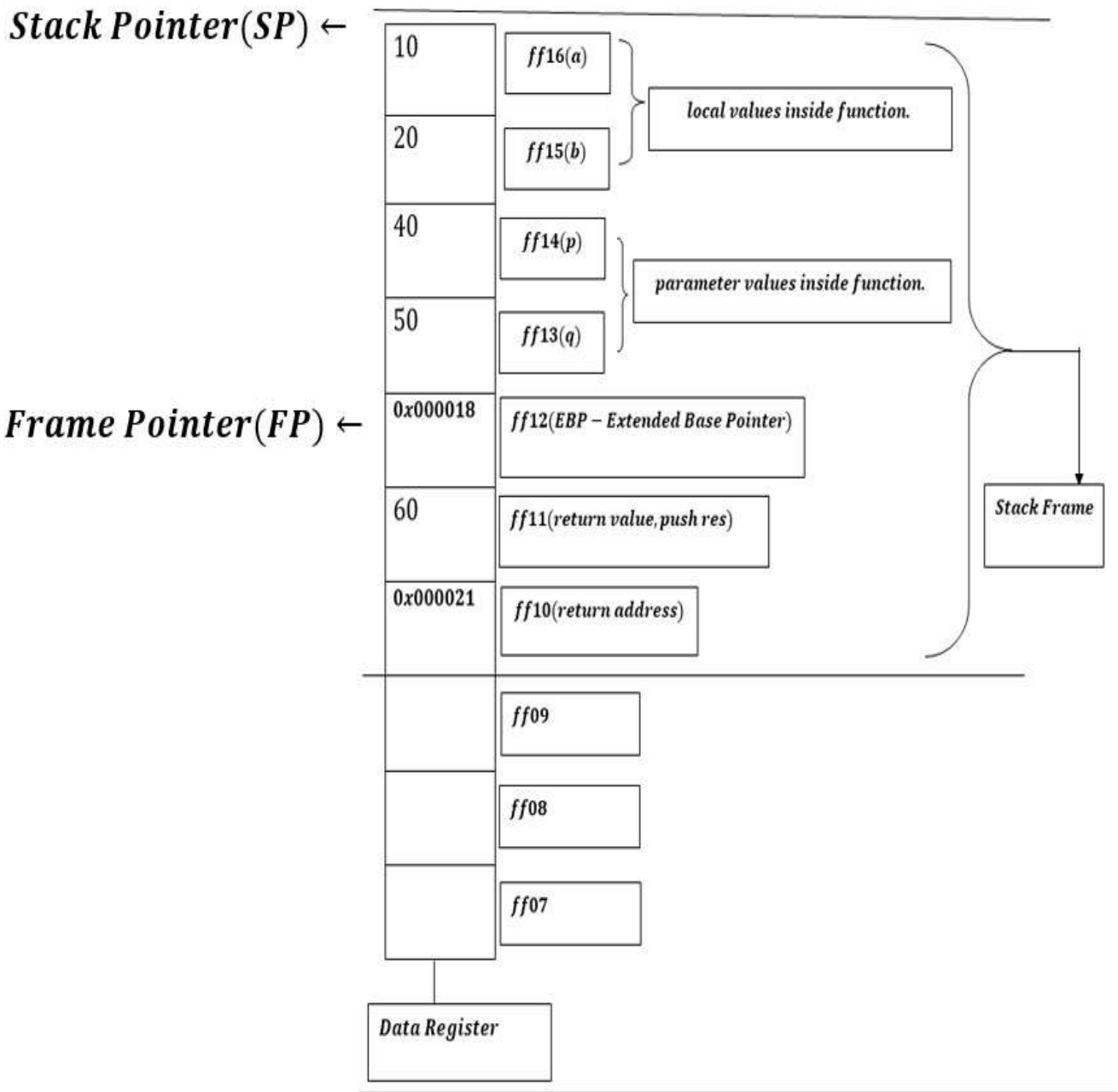
2nd Step


Stack Pointer(SP) ←
10
ff16(a)
20
ff15(b)
local values inside function.
40
ff14(p)
50
ff13(q)
parameter values inside function.
Frame Pointer(FP) ←
0x000018
ff12(EBP – Extended Base Pointer)
ff11(return value)
0x000021
ff10(return address)
Stack Frame
30
ff09(push res)
50
ff08(push q)
20
ff07(push b)
Stack Operation
Data Register (–)

3rd Step


Stack Pointer(SP) ←
10
ff16(a)
local values inside function.
20
ff15(b)
40
ff14(p)
parameter values inside function.
50
ff13(q)
Frame Pointer(FP) ←
0x000018
ff12(EBP – Extended Base Pointer)
ff11(return value)
Stack Frame
0x000021
ff10(return address)
30
ff09(push res)
Stack Operation
30
ff08(push res)
ff07
Data Register (+)

Final Step


Stack Pointer(SP) ←
10
ff16(a)
local values inside function.
20
ff15(b)
40
ff14(p)
parameter values inside function.
50
ff13(q)
Frame Pointer(FP) ←
0x000018
ff12(EBP – Extended Base Pointer)
60
ff11(return value, push res)
Stack Frame
0x000021
ff10(return address)
ff09
ff08
ff07
Data Register

After the execution of the Stack Operation.

1. *Return Value and Return Address will get popped out from from the stack and PC (Program Counter)will hold the return address, which will prompt CPU that no further execution is needed.*

2. *No sooner after the instruction , function gets deallocated and the formal parameter and local variable of the function become invalid, hence gets destroyed.*

3. *The deallocation and destruction occurs automatically.*

Stack Pointer in Addtion/Subtraction VS Stack Pointer in Stack Frame.

Stack Pointer involved in standard addition and subtraction and Stack Pointer in Stack Frame. In standard addition and subtraction , stack pointer , which is a register itself points to the top of the stack.

Where as,

in Stack Frames, the stack pointer is used to keep track of the current position in the stack for a specific stack frame. It points to the top of the stack within that frame.

****