Recursion – Part 1

1. *A function that calls itself is called recursive.*

2. *The function creates a copy of itself and push it to the stack.*

3. *When the function creates a copy of itself and pushes it to the stack and pops out to solves the given problem for which recursive function is assigned , such steps are known as recursive steps.*

4. *When function calls itself such calls are known as recursive calls of the function.*

5. *Recursive function terminates when it reaches its `base` condition.*

Memory Visualization of Recursive Function.

Lets take an example : –

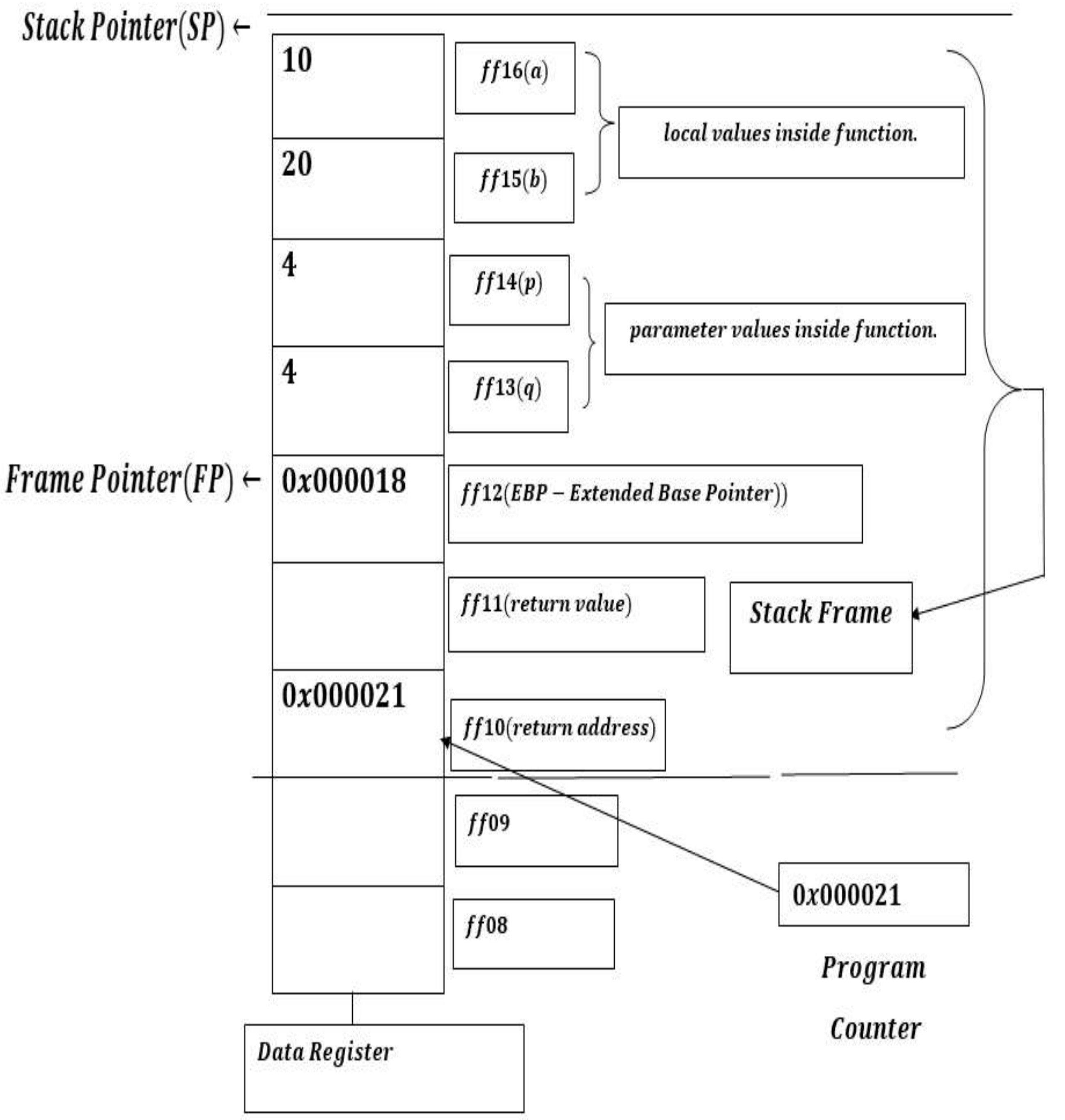
```cpp
#include<iostream>
using namespace std;

int print(int p,int q){
    int a= 10;
    int b =20;

    if(p==2 || q==2){
        return 2;
    }
    else{
        //recursion
        cout<<a<<" "<<b<<endl;
        return print(p-1,q-1);
    }

}

int main(){
    int n;
    cin>>n;
    print(n,n);
    return 0;
}
```

$if(p == 2\ ||\ q == 2)$ ***is the Base Case ,***
where as $print(p-1, q-1)$ ***is Recursive Function Call.***
Let $n = 4$.

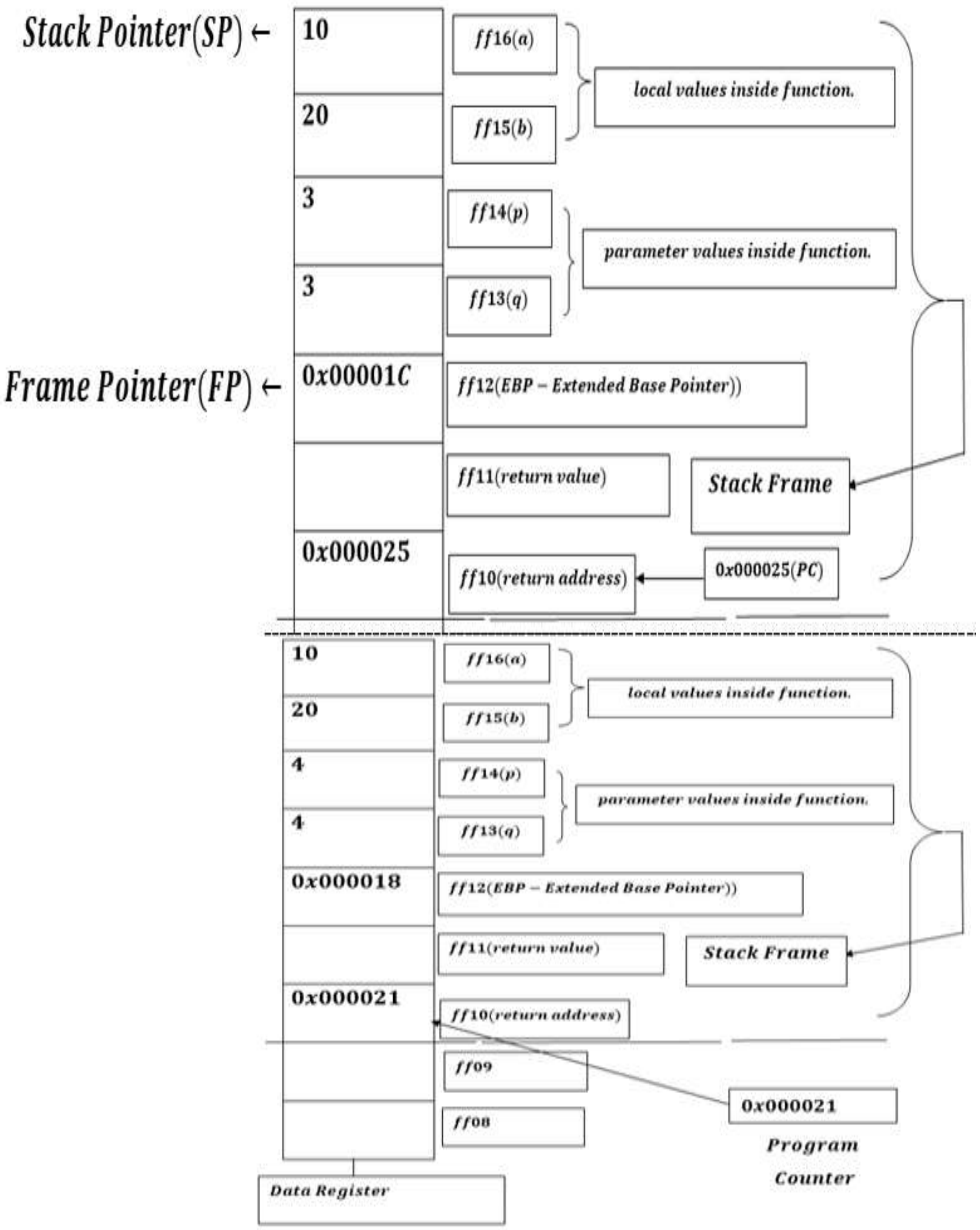
<u>Push Operation of the Recursive Function</u>

As we have experienced in Function , stack frame is created during function call , during recursive call the function creates a copy of itself and a new stack frame is pushed above like:

1st Part


Stack Pointer(SP) ←
10
20
4
4
Frame Pointer(FP) ←
0x000018
0x000021
ff16(a)
ff15(b)
local values inside function.
ff14(p)
ff13(q)
parameter values inside function.
ff12(EBP – Extended Base Pointer))
ff11(return value)
Stack Frame
ff10(return address)
ff09
ff08
0x000021
Program Counter
Data Register

$2nd\ Part \rightarrow Print(p-1, q-1)$



As usual value of PC(Program Counter) is pushed inside and each return address will be unique for each stack frame. Also if we see Frame pointer was 0x000018 and now 0x00001C
frame pointer or EBP is 4 bytes larger than the previous stack frame . And Each stack frame will have unique frame pointers.

$(\mathbf{0x0000018})_{16} \rightarrow (\mathbf{24})_{10}$
$(\mathbf{0x000001C})_{16} \rightarrow (\mathbf{28})_{10}$

Q) Why the next Frame Pointer is 4 bytes?

<u>Explanation:</u>
Local Variables , Parameter Values etc. takes 4 bytes of space in the stack. Hence we add 4 bytes to the stack frame.

Stack Frame	*OFFSET*	*SIZE*	*Contents*
EBP	4	4 *bytes*	0*x*000018
Aguments	8	4 *bytes*	*p, q*
Local Variables	12	4 *bytes*	*a, b*
Return Address	16	4 *bytes*	0*x*000021

$\boldsymbol{Next\ stack\ frame\ will\ be}$:

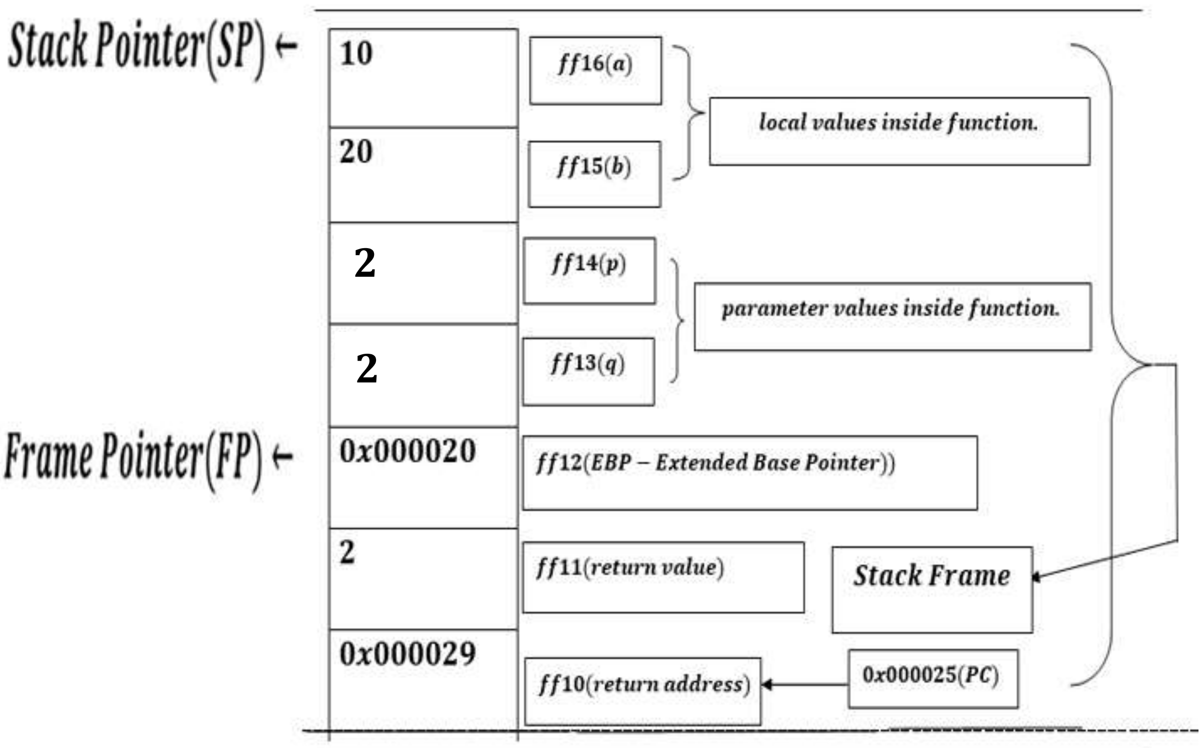
$Stack\ Frame$	$OFFSET$	$SIZE$	$Contents$
EBP	4	$4\ bytes$	$0x00001C$
$Aguments$	8	$4\ bytes$	p, q
$Local\ Variables$	12	$4\ bytes$	a, b
$Return\ Address$	16	$4\ bytes$	$0x000025$

$\boldsymbol{Hence\ return\ address\ \ will\ also\ be\ increased\ 4\ bytes\ long}$.

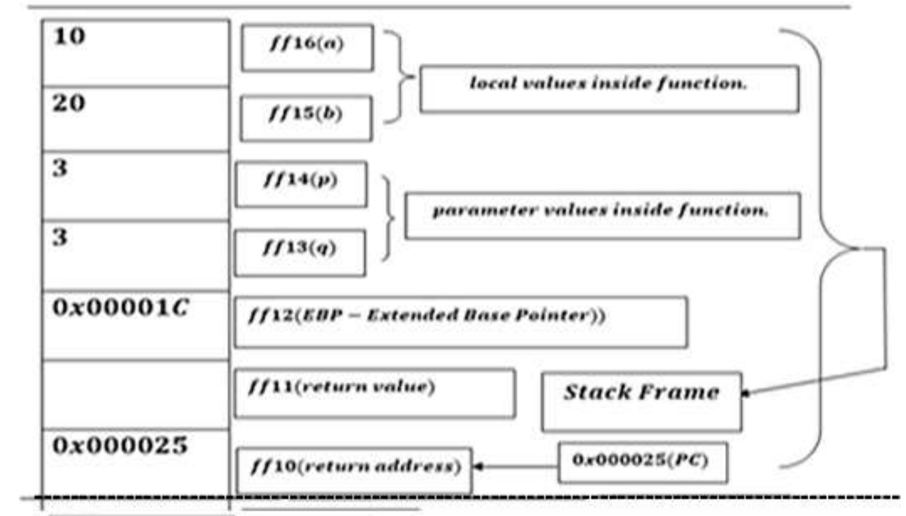
$\boldsymbol{Q})\ \boldsymbol{What\ is\ Offset\ then}$?

<u>$\boldsymbol{Explanation}$:</u>
$\boldsymbol{An\ offset\ in\ a\ stack\ frame\ is\ the\ distance\ between\ a\ variable\ and\ frame\ pointer. If\ frame\ pointer\ is}$:
$(\boldsymbol{0x12345678})_{16} = (\boldsymbol{305419896})_{10}$ $\boldsymbol{and\ a\ offset\ of\ a\ variable\ is\ 4\ then\ the\ address\ of\ the\ variable\ is}$:
$(\boldsymbol{0x1234567C})_{16} = (\boldsymbol{305419900})_{10}$ $\boldsymbol{i.e.}$
$\boldsymbol{0x12345678 + 4\ or\ 305419896 + 4}$.

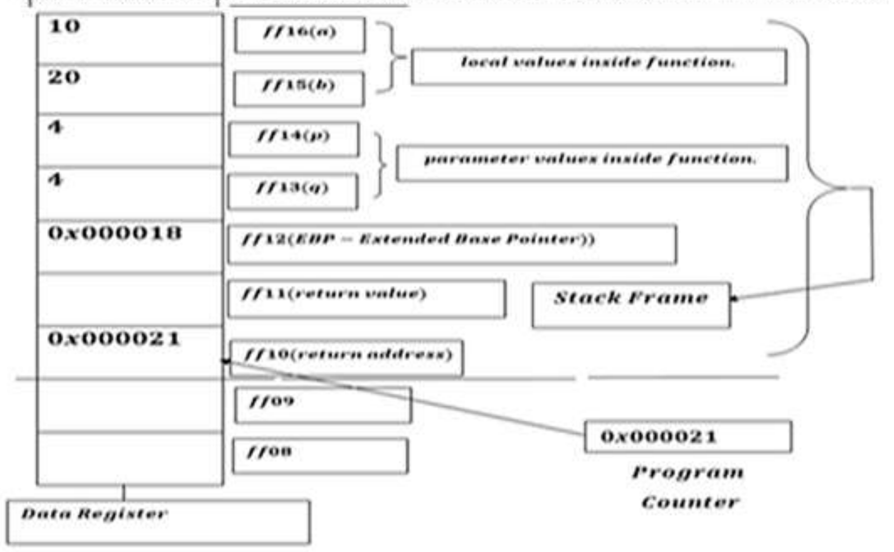
<u>$\boldsymbol{Last\ Part \rightarrow Print(p-1, q-1)}$</u>



Stack Pointer(SP) ←
Frame Pointer(FP) ←
10
20
2
2
0x000020
2
0x000029
ff16(a)
ff15(b)
local values inside function.
ff14(p)
ff13(q)
parameter values inside function.
ff12(EBP – Extended Base Pointer))
ff11(return value)
Stack Frame
ff10(return address)
0x000025(PC)



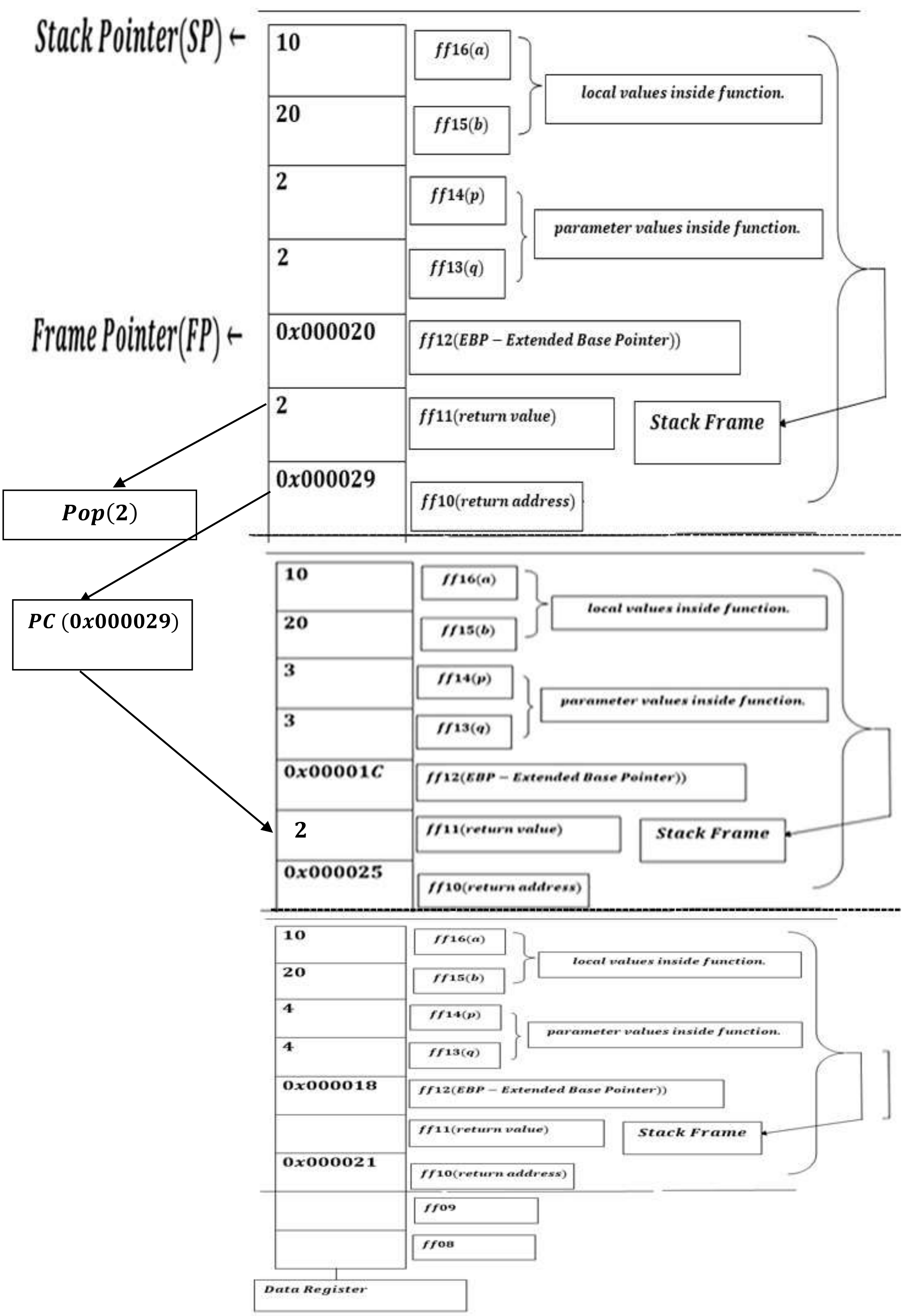
10
20
3
3
0x00001C
0x000025
ff16(a)
ff15(b)
local values inside function.
ff14(p)
ff13(q)
parameter values inside function.
ff12(EBP – Extended Base Pointer))
ff11(return value)
Stack Frame
ff10(return address)
0x000025(PC)



10
20
4
4
0x000018
0x000021
ff16(a)
ff15(b)
local values inside function.
ff14(p)
ff13(q)
parameter values inside function.
ff12(EBP – Extended Base Pointer))
ff11(return value)
Stack Frame
ff10(return address)
ff09
ff08
0x000021
Program Counter
Data Register

Now as the base condition now we remian with the popping operation.

Pop Operation

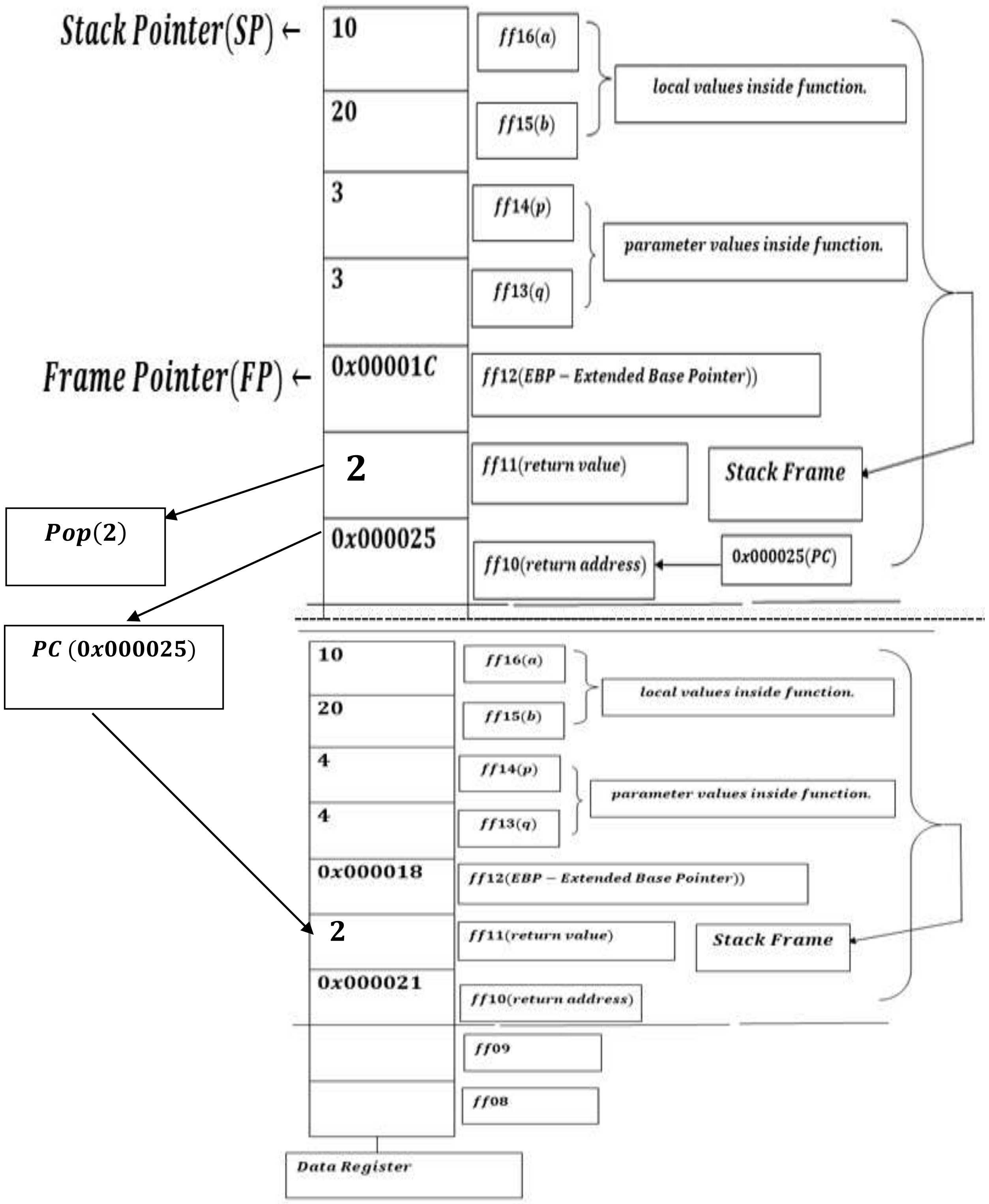

Stack Pointer(SP) ←
Frame Pointer(FP) ←
10
20
2
2
0x000020
2
0x000029
ff16(a)
ff15(b)
local values inside function.
ff14(p)
ff13(q)
parameter values inside function.
ff12(EBP – Extended Base Pointer))
ff11(return value)
Stack Frame
ff10(return address)
Pop(2)
PC (0x000029)
10
20
3
3
0x00001C
2
0x000025
ff16(a)
ff15(b)
local values inside function.
ff14(p)
ff13(q)
parameter values inside function.
ff12(EBP – Extended Base Pointer))
ff11(return value)
Stack Frame
ff10(return address)
10
20
4
4
0x000018
0x000021
ff16(a)
ff15(b)
local values inside function.
ff14(p)
ff13(q)
parameter values inside function.
ff12(EBP – Extended Base Pointer))
ff11(return value)
Stack Frame
ff10(return address)
ff09
ff08
Data Register

1. *Return Value and Return Address gets popped out.*
2. *Program Counter retrieves the Return Address, and push the return value related to the return Address to the stack frame.*

3. *Now Frame Pointer will gets subtracted by* 4 *bytes.*
i.e. $0x00020 - 4\ bytes = 0x00001C$. *Hence frame pointer now points to* $\rightarrow 0x00001C$.

4. *The above stack frame gets deallocated and the local variables and parameter variables gets destroyed automatically.*

Similarly,


Stack Pointer(SP) ←
Frame Pointer(FP) ←
10
20
3
3
0x00001C
2
0x000025
ff16(a)
ff15(b)
local values inside function.
ff14(p)
ff13(q)
parameter values inside function.
ff12(EBP – Extended Base Pointer))
ff11(return value)
Stack Frame
ff10(return address)
0x000025(PC)
Pop(2)
PC (0x000025)
10
20
4
4
0x000018
2
0x000021
ff16(a)
ff15(b)
local values inside function.
ff14(p)
ff13(q)
parameter values inside function.
ff12(EBP – Extended Base Pointer))
ff11(return value)
Stack Frame
ff10(return address)
ff09
ff08
Data Register

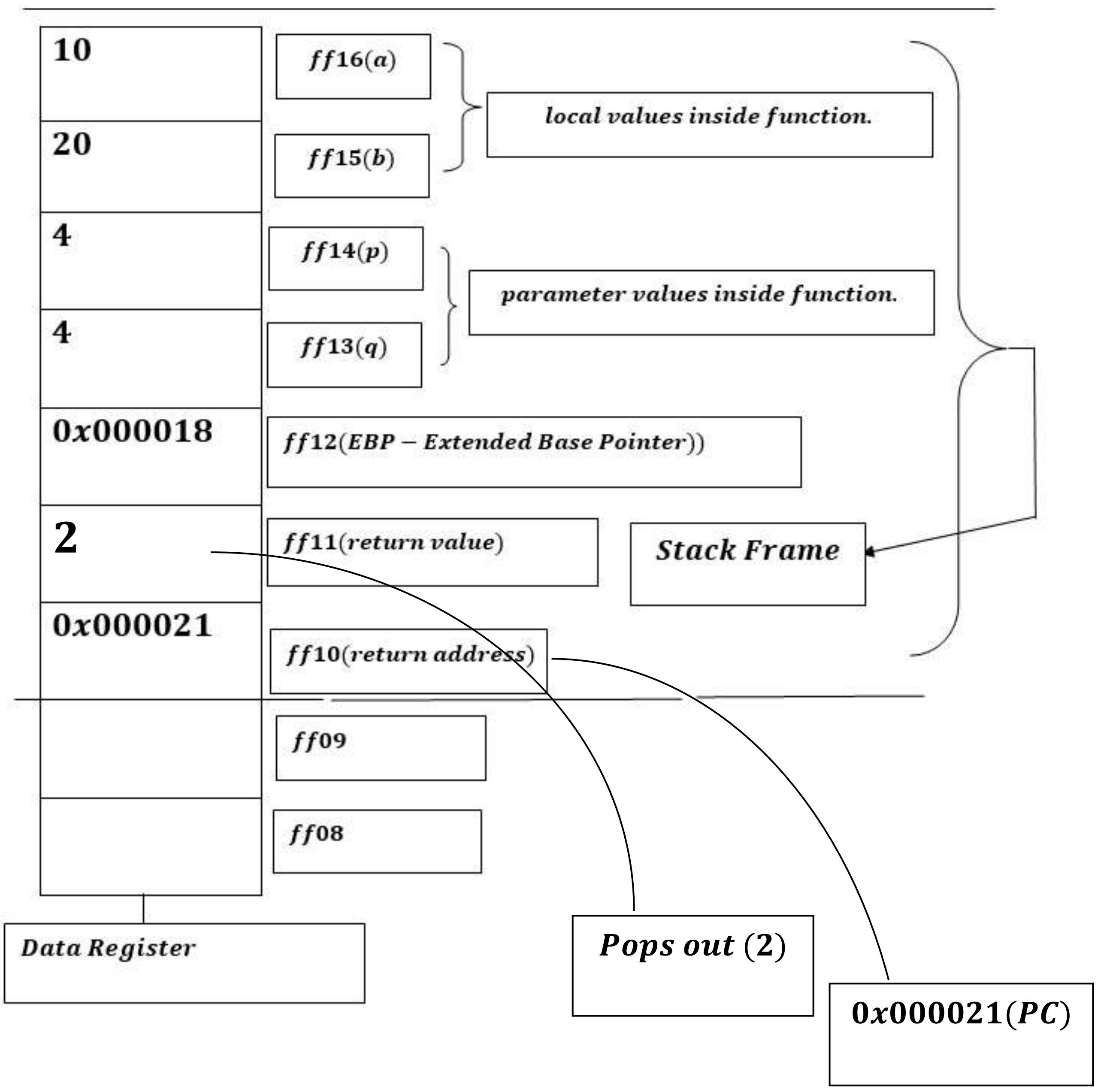
***The above steps are repeated*:**

1. *Return Value and Return Address gets popped out*.
2. *Program Counter retrieves the Return Address, and push the return value related to the return Address to the stack frame*.

3. *Now Frame Pointer will gets subtracted by* 4 *bytes*.
$i.e.\ 0x0001C - 4\ bytes = 0x000021$. *Hence frame pointer now points to* $\rightarrow 0x000021$.

4. *The above stack frame gets deallocated and the local variables and parameter variables gets destroyed automatically*.

***Similarly*,**


10
20
4
4
0x000018
2
0x000021
ff16(a)
ff15(b)
local values inside function.
ff14(p)
ff13(q)
parameter values inside function.
ff12(EBP − Extended Base Pointer))
ff11(return value)
Stack Frame
ff10(return address)
ff09
ff08
Data Register
Pops out (2)
0x000021(PC)

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.***

****