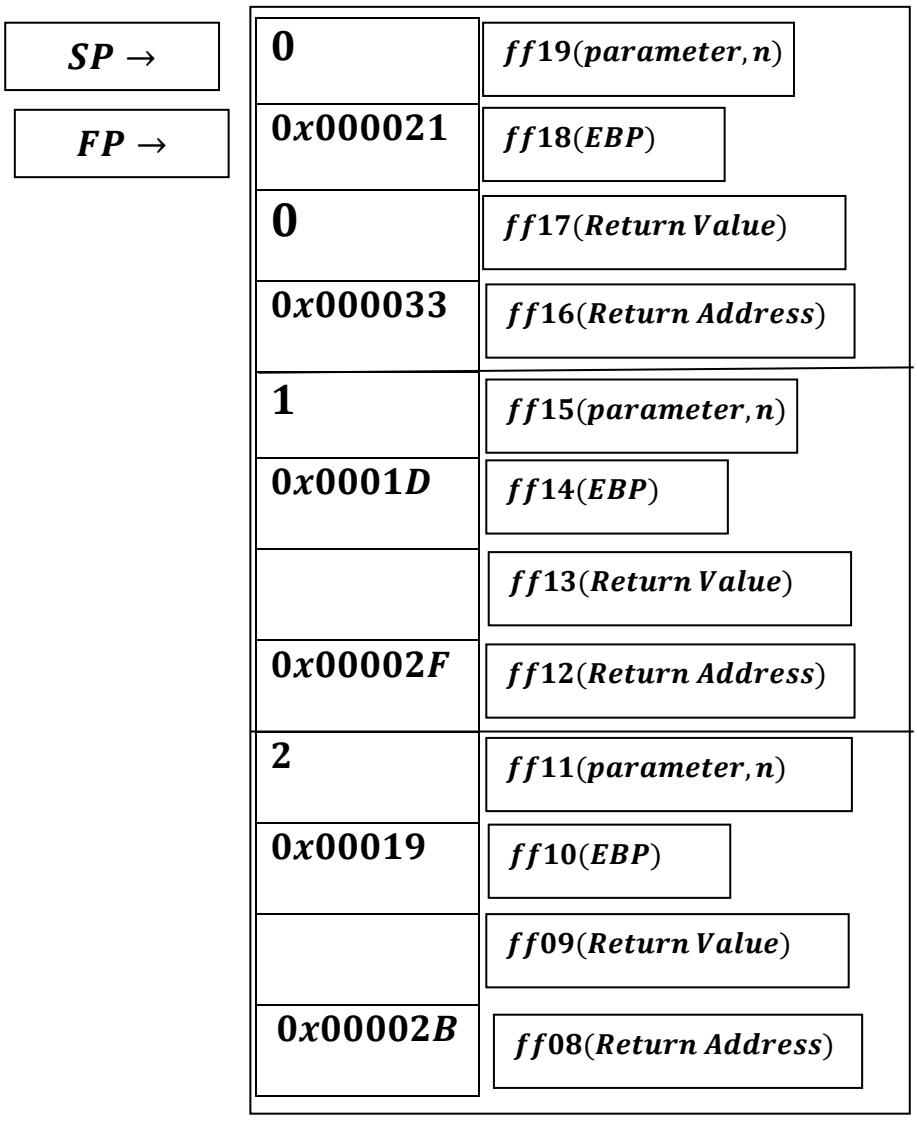
$Recursion - 2$

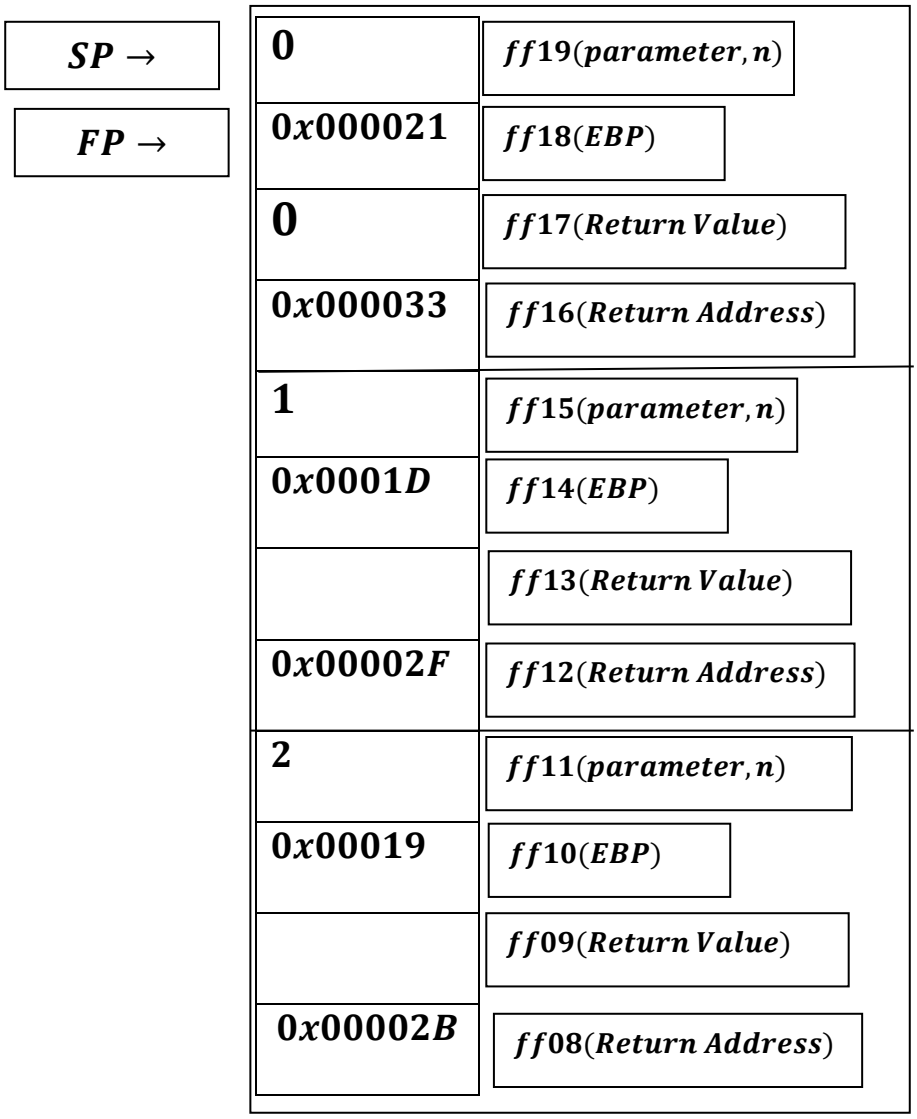
Now lets consider a similar type of regular example:

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

int print(int n){

    if(n==0){
        return 0;
    }

    cout<<n<<endl;
    return print (n-1);
}

int main (){

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

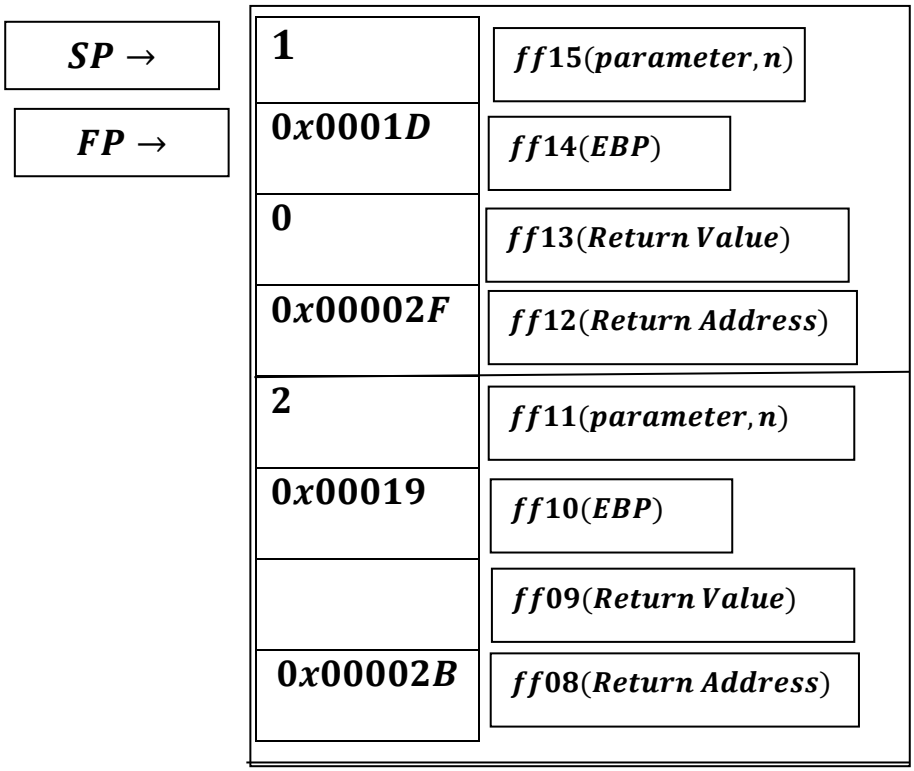
If $n = 2$ ***, then push operation***:



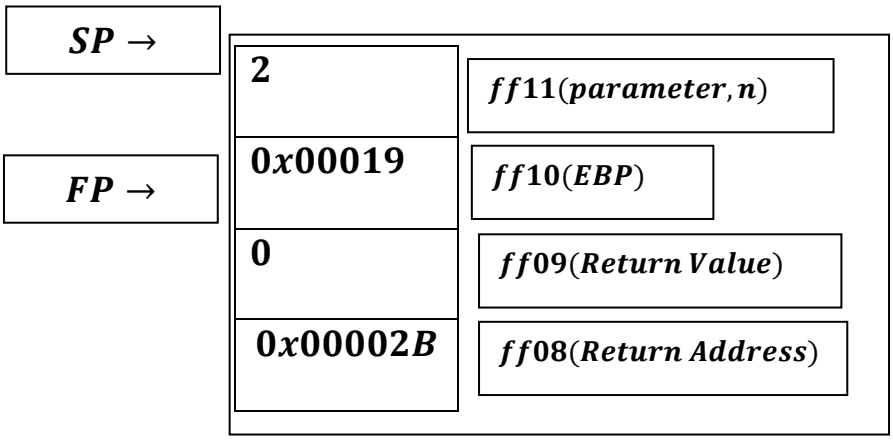
Now, Pop operation :


SP →
FP →
0
ff19(parameter, n)
0x000021
ff18(EBP)
0
ff17(Return Value)
0x000033
ff16(Return Address)
1
ff15(parameter, n)
0x0001D
ff14(EBP)
ff13(Return Value)
0x00002F
ff12(Return Address)
2
ff11(parameter, n)
0x00019
ff10(EBP)
ff09(Return Value)
0x00002B
ff08(Return Address)

Next:


SP →
FP →
1
ff15(parameter, n)
0x0001D
ff14(EBP)
0
ff13(Return Value)
0x00002F
ff12(Return Address)
2
ff11(parameter, n)
0x00019
ff10(EBP)
ff09(Return Value)
0x00002B
ff08(Return Address)

Next:


SP →
FP →
2
ff11(parameter, n)
0x00019
ff10(EBP)
0
ff09(Return Value)
0x00002B
ff08(Return Address)

Q)What cout does?

Explanation:

Now `cout` output data to a stream, it has nothing to do with push and pop function of stack.

The value of `n` in the above program is pushed in the stack frame before it is printed .

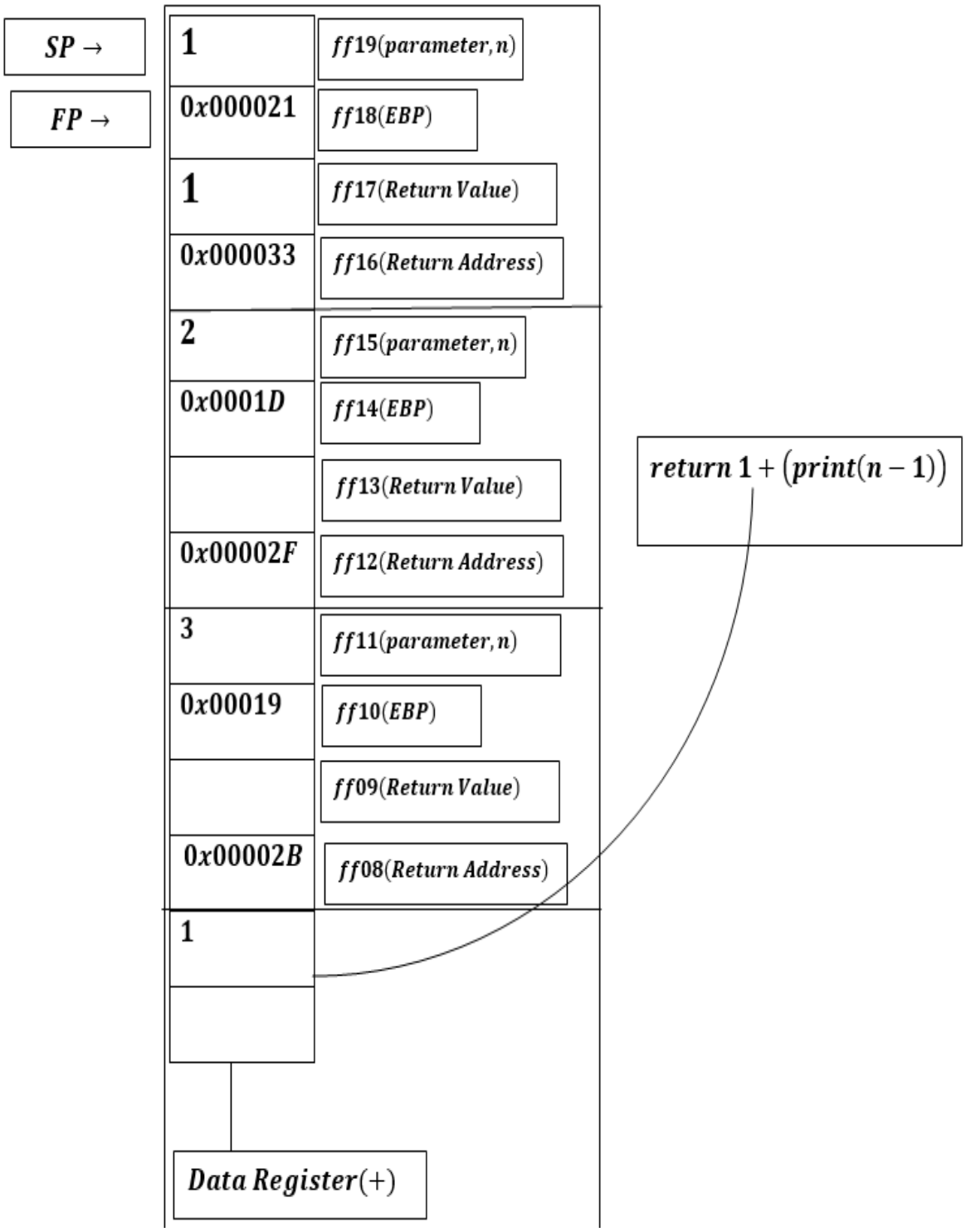
The value of `n` will then be popped off the stack after it has been printed.

Now consider another recursive program:

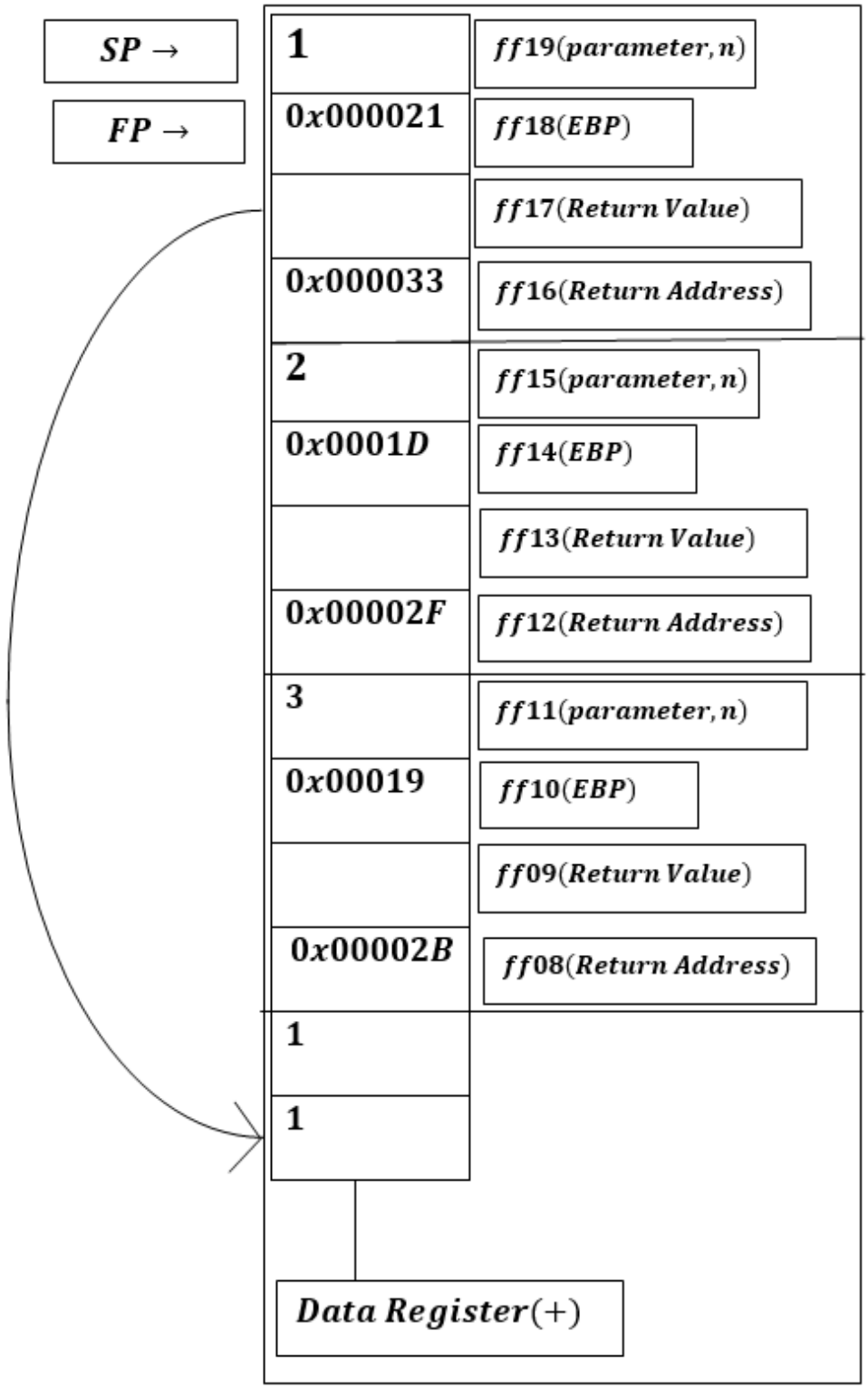
```
int print (int n) {

    if(n==1) {
        return 1;
    }


    return 1 + print(n-1);

}
```

We will start ,when the push operation is completed and the base condition is executed.

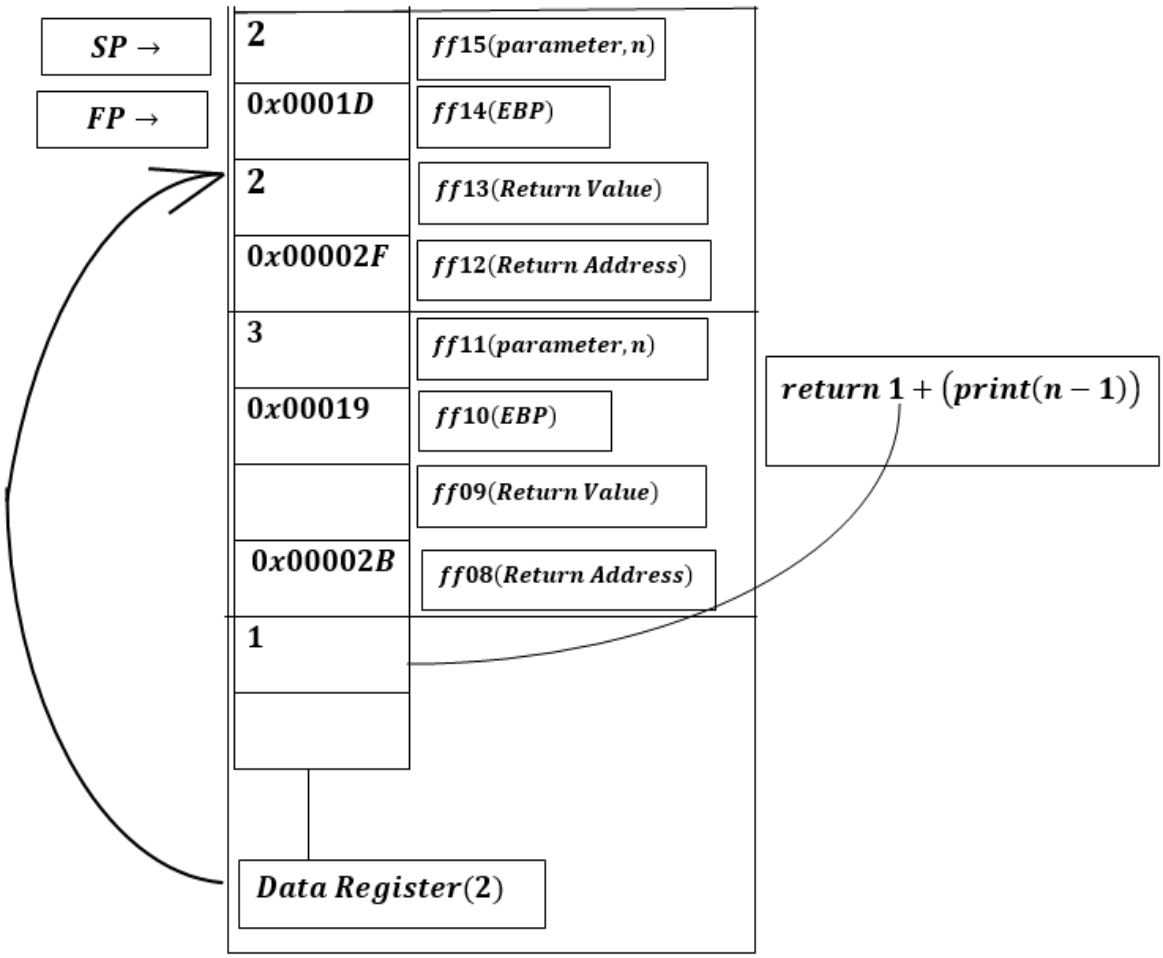
The `1` of the return will be pushed to the stack ,as shown below:

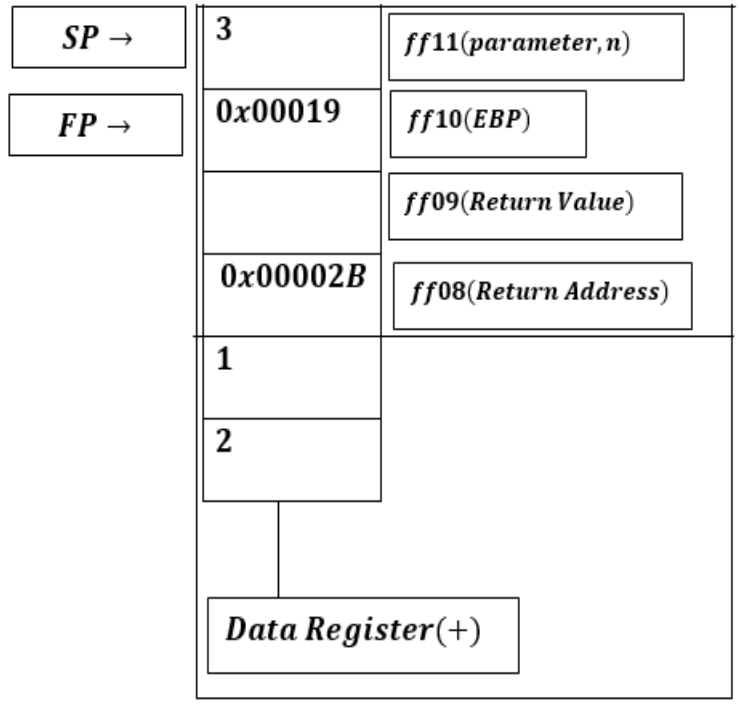

SP →
FP →
1
ff19(parameter, n)
0x000021
ff18(EBP)
1
ff17(Return Value)
0x000033
ff16(Return Address)
2
ff15(parameter, n)
0x0001D
ff14(EBP)
ff13(Return Value)
0x00002F
ff12(Return Address)
3
ff11(parameter, n)
0x00019
ff10(EBP)
ff09(Return Value)
0x00002B
ff08(Return Address)
1
Data Register(+)
return 1 + (print(n − 1))

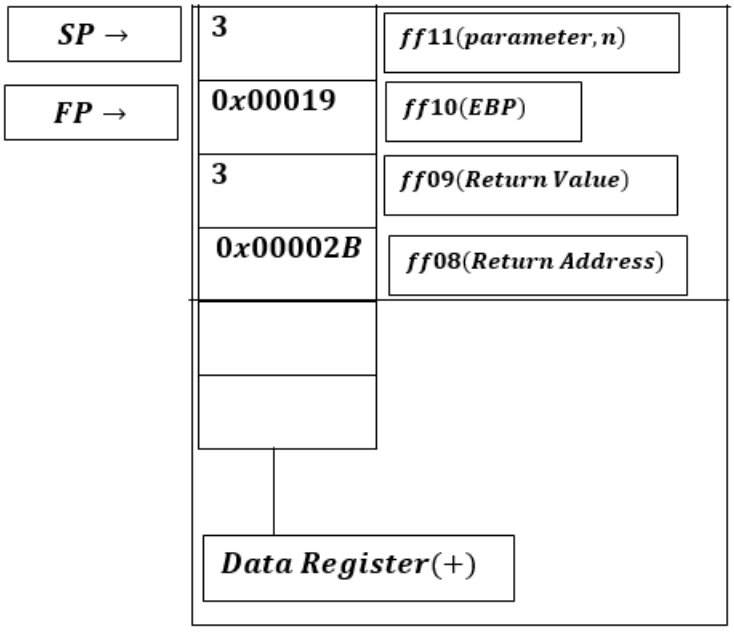
Next the base return `1` will be added with `1` and result will be pushed to the return value section.



***And the process is continued till the stack frame operation is not finished as shown below*:**




SP →
FP →
3
ff11(parameter, n)
0x00019
ff10(EBP)
ff09(Return Value)
0x00002B
ff08(Return Address)
1
2
Data Register(+)


SP →
FP →
3
ff11(parameter, n)
0x00019
ff10(EBP)
3
ff09(Return Value)
0x00002B
ff08(Return Address)
Data Register(+)

Main Function Stack Frame

1. ***There are two kind of scenarios*** **:**

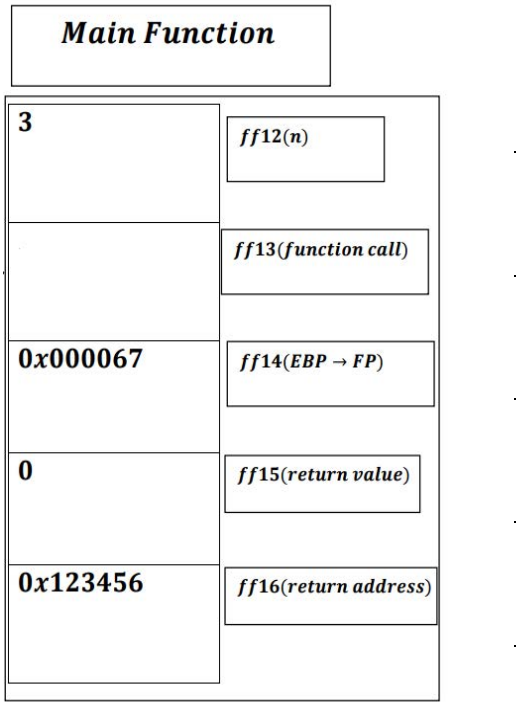
a) either function is called in main function without being assigned to any variable.

b)either function is called in main function assigned to any variable.

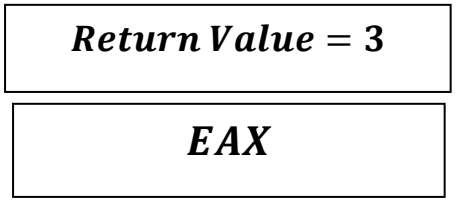
Consider the following example:

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

int print(int n){

    if(n==0){
        return 0;
    }
    return 1+ print(n-1);
}

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

Say $n = 3$.



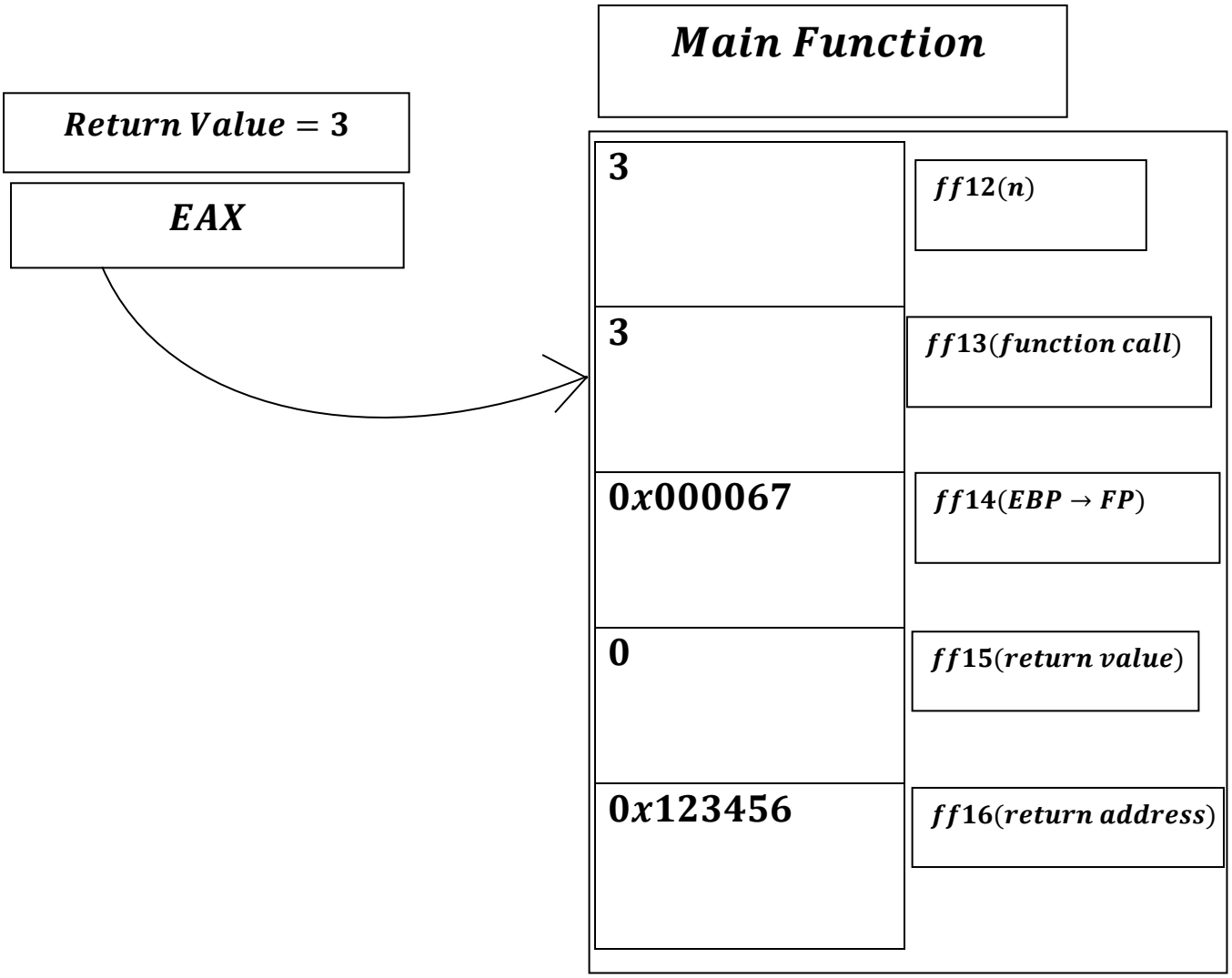
The result gets popped out from print() function stack frame is 3. Return value 3 gets stored in EAX register connected with Microprocessor.



EAX i.e. Extended Accumulator of Microprocessor, which is itself a register.

And taking from the above example PC (Program Counter) has the reurn address : 0x00002B.

Now from EAX register , return value 3 will get pushed to main function stack frame.

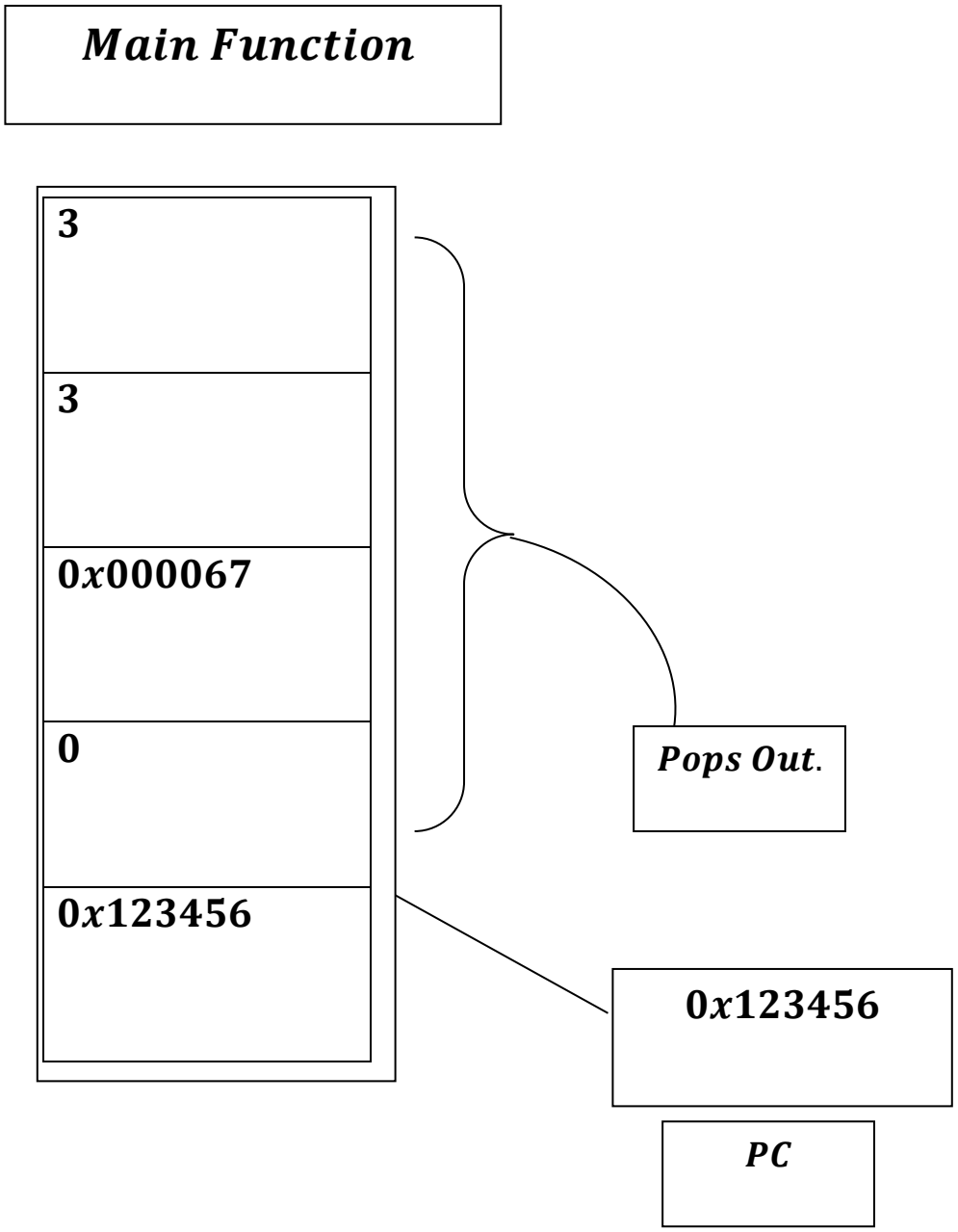


This process remain same for int a = print(n), as shown below:

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

int print(int n){
    if(n==0){
        return 0;
    }
    return 1+ print(n-1);
}

int main(){

    int n;
    cin>>n;
    int a =  print(n);
    return 0;
}
```

In same way cin is standard input from input device and has nothing to do with directly for stack but as it take input for a variable, the input value gets pushed in stack .

Now as return value gets popped out i.e. 3, Program Counter (PC) is updated with return address of main function: 0x123456.



And like other function's stack frame , the main function's stack frame also deallocates and local variables gets destroyed automatically .

****