The ret instruction:

ret == popl   %eip

the ret instruction accesses the address of the top of the stack from esp register. Call that address as x.

It transfers the content of 4 bytes of the stack top (i.e. M[x : x + 3]) to eip register.

1) eip <- M[esp : esp + 3]

2) esp <- esp + 4

Step 1 + Step 2 == popl %eip

#--------------------

For learning function definition and function call in systematic manner, we will divide C function in 4 progressive stages of the evolution:

Step 1 : A C function without any parameter, without return value,
without local variables, without nested function calls.

Step 2: A C function with parameters and return value. but no local variables
and no nested function calls

Step 3: A C function
1) with parameters
2) with return value
3) with local variables
4) without nested function calls

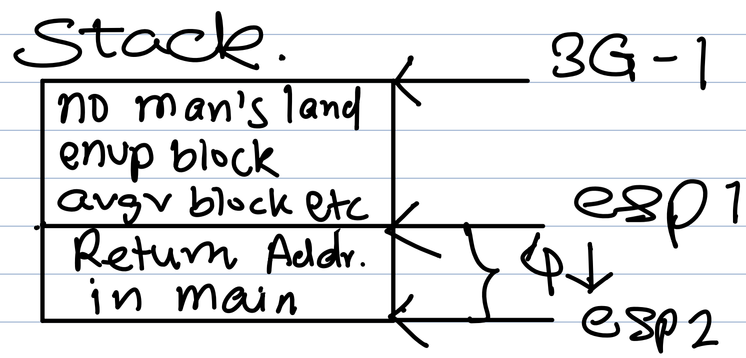
Step 4: <FULL FLEDGED C FUNCTION>

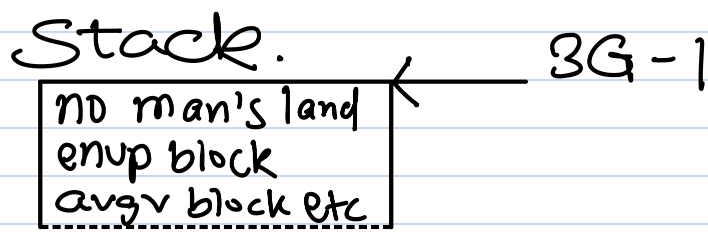
1) function with parameters
2) function with return value
3) function with local variables
4) function with nested calls (having all 4 capacities)

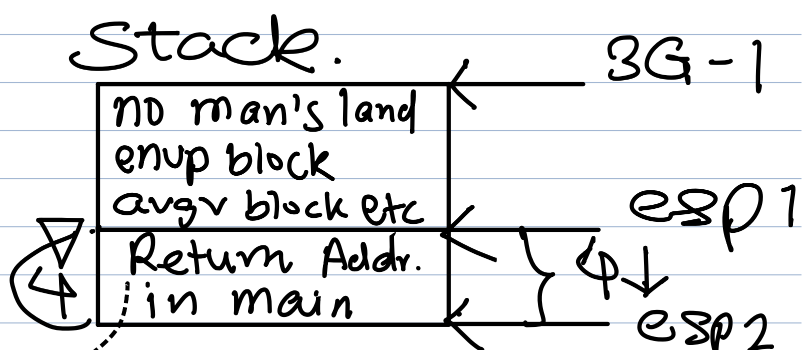
#------------------------------

Step 1:
step1.c

```
int num1, num2, result;
void test_function(void){
    // global variable manipulation
    num1 = 10;
    num2 = 20;
    result = num1 + num2;
}

void main(){
    test_function();
}

.section .bss
    .comm  num1, 4, 4
    .comm  num2, 4, 4
    .comm  result, 4, 4

.section .text
.globl    main
.type     main, @function

main:
    call   test_function --- eip1
    ret

.globl test_function
.type test_function, @function

test_function:
    movl   $10, num1 --- eip2
    movl   $20, num2 --- eip3
    movl   num1, %eax --- eip4
    addl   num2, %eax --- eip5
    movl   %eax, result --- eip6
    ret ---- eip7
```



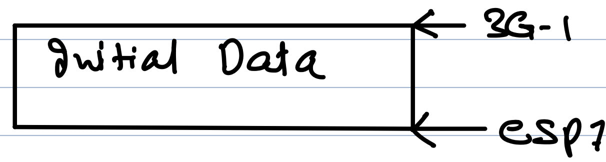


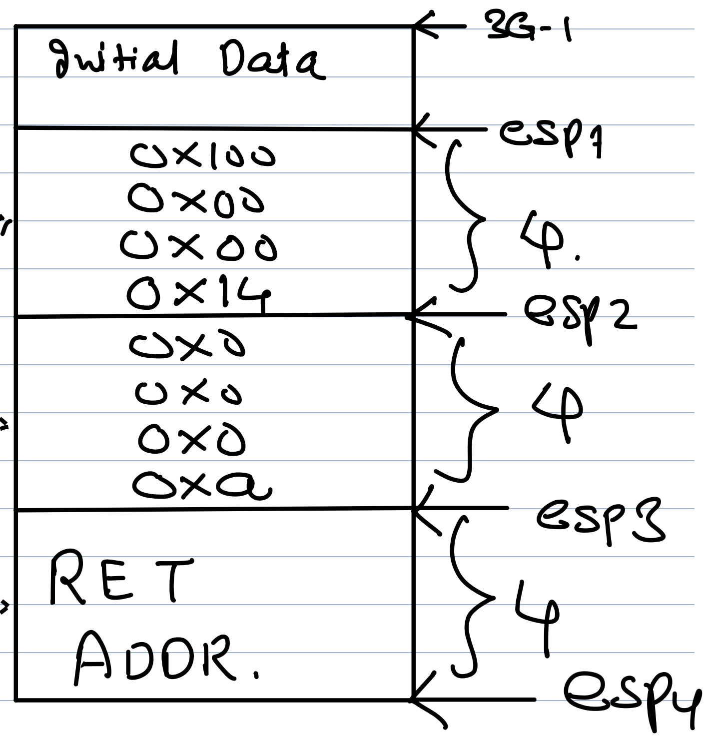


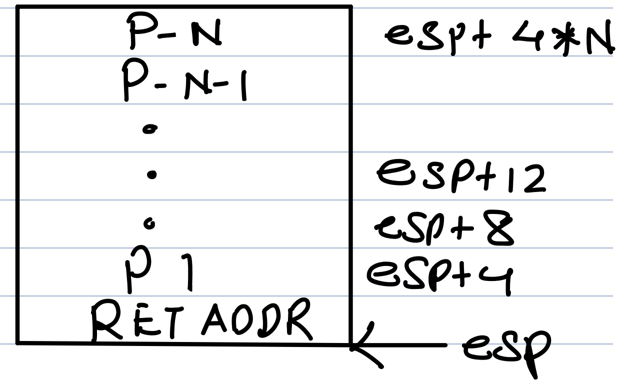
eip8

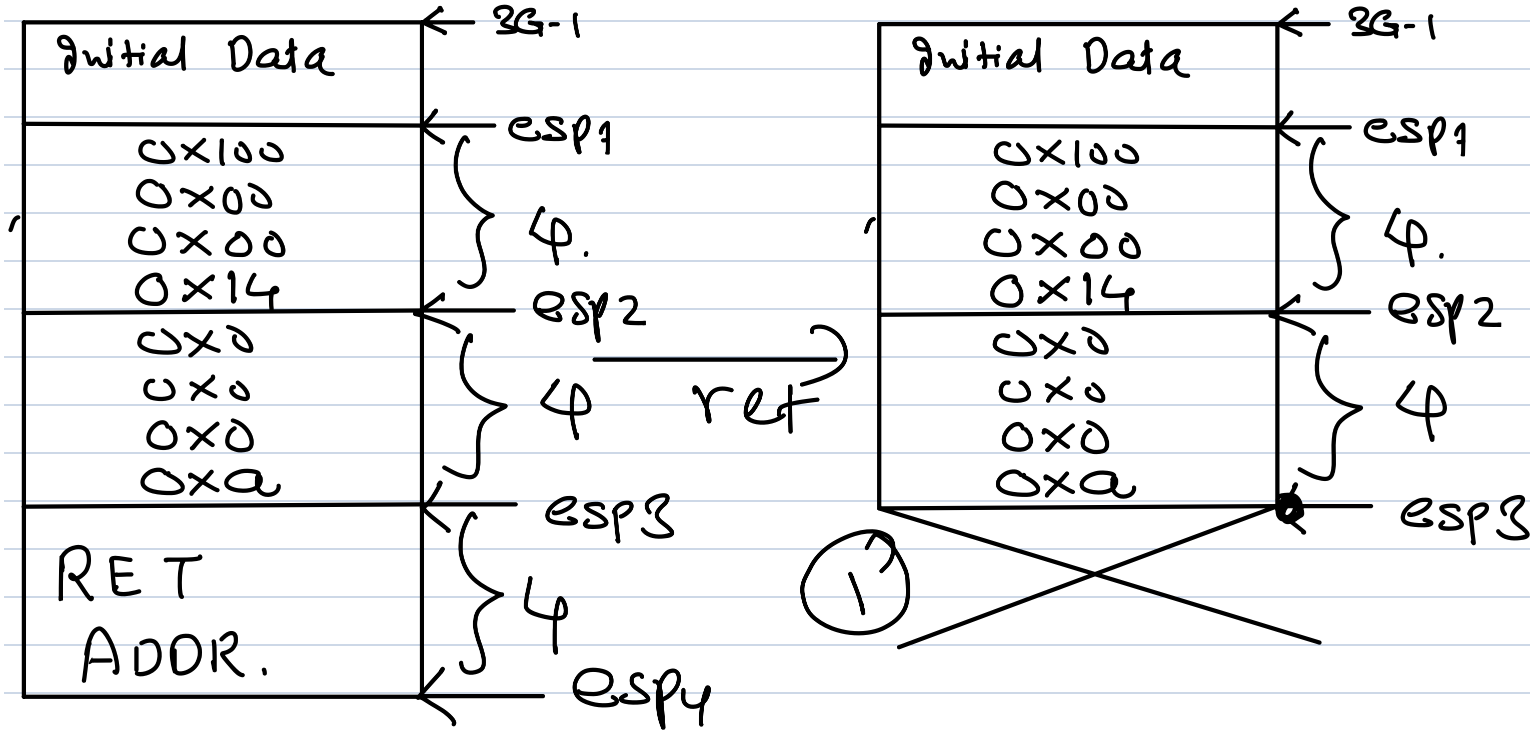
```
int test_function(int n1, int n2){
        return (n1+n2);
}

int main(){
      result = test_function(10,20);
}.
```

```
main:
    pushl $20
    pushl $10
    call test_function
RET
ADDR
addl $8,%esp
movl %eax,result.
movl $0, %eax
ret
test_function:
    movl 4(%esp), %edx
    addl 8(%esp), %edx
    movl %edx, %eax
    ret
```









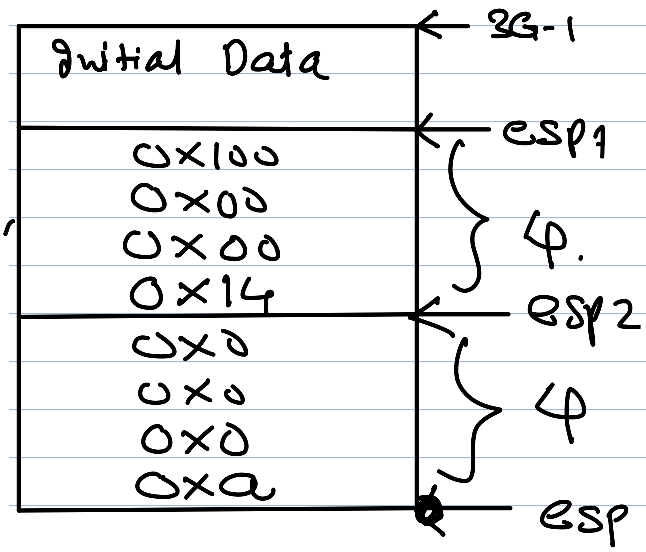
pushl $20

pushl $10

call test_function

2 addl $8, %esp

RET ADDR.



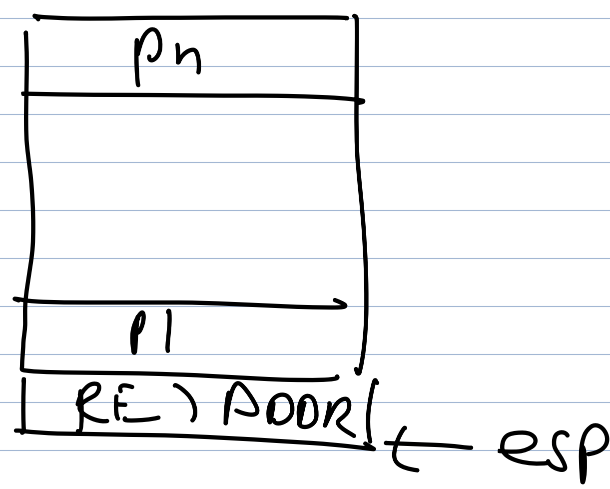
addl $8, %esp

```
pushl $20
pushl $10
call test_function
addl $8, %esp
movl %eax, result
```

```
result = test_function (10, 20);
```

```
ret = fun_name(T1 P1, ..., Tn Pn);
```

```
pushl Pn
.
.
pushl P1
call fun_name
addl $4n, %esp
movl %eax, ret
```



fun_name: 4×k(%esp) $1 \leq k \leq n$.

logic

movl ret → %eax

XWindow | BST | graph → adj
→ BFS, DFS | RB tree

```
int test_function (int n1, int n2)
{       int rs1, rs2;
        rs1 = n1 + n2;
        rs2 = n1 - n2;
        return (rs1 * rs2);
}
```

```
int rs;
int main(void) {
    rs = test_function (10, 20);
    return (0);
}
```

```
.section .bss.
  .comm rs, 4, 4.
.section .text.
main:
      pushl $20
      pushl $10
      call  test_function
R.A.  addl $8, %esp
          movl %eax, rs
```

init_data

init_data	
20	4
10	4
RA.	

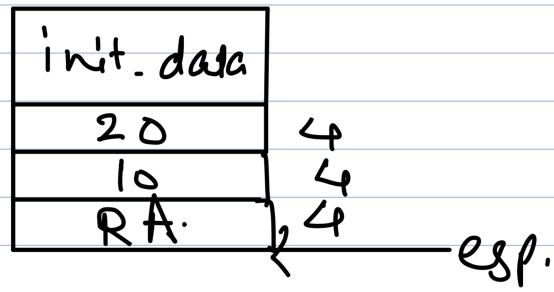
test_function:

```
movl %esp, %ebp
subl $8, %esp
```

$4 \times k(\%ebp)$ $\quad 1 \le k \le n$

$-4k(\%ebp)$ $\quad 1 \le k \le m$

Logic

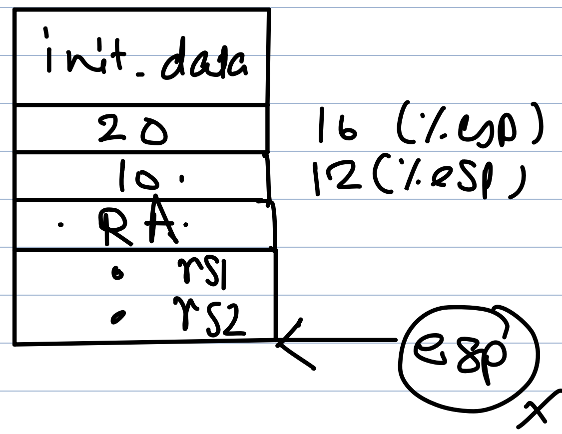


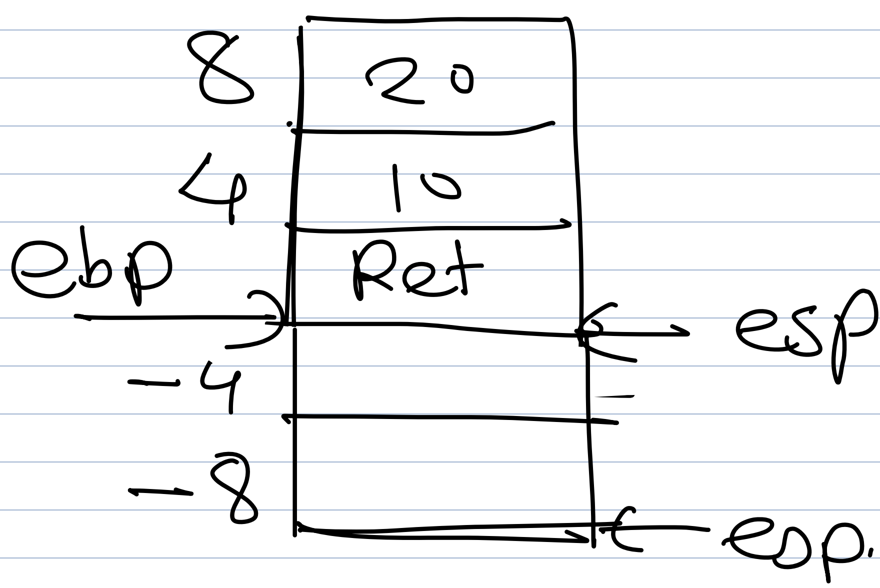
$0(\%esp) \rightarrow R.A.$

$4(\%esp) = P1$

$8(\%esp) = P2.$

```
movl — %eax
movl %ebp, %esp
ret
```





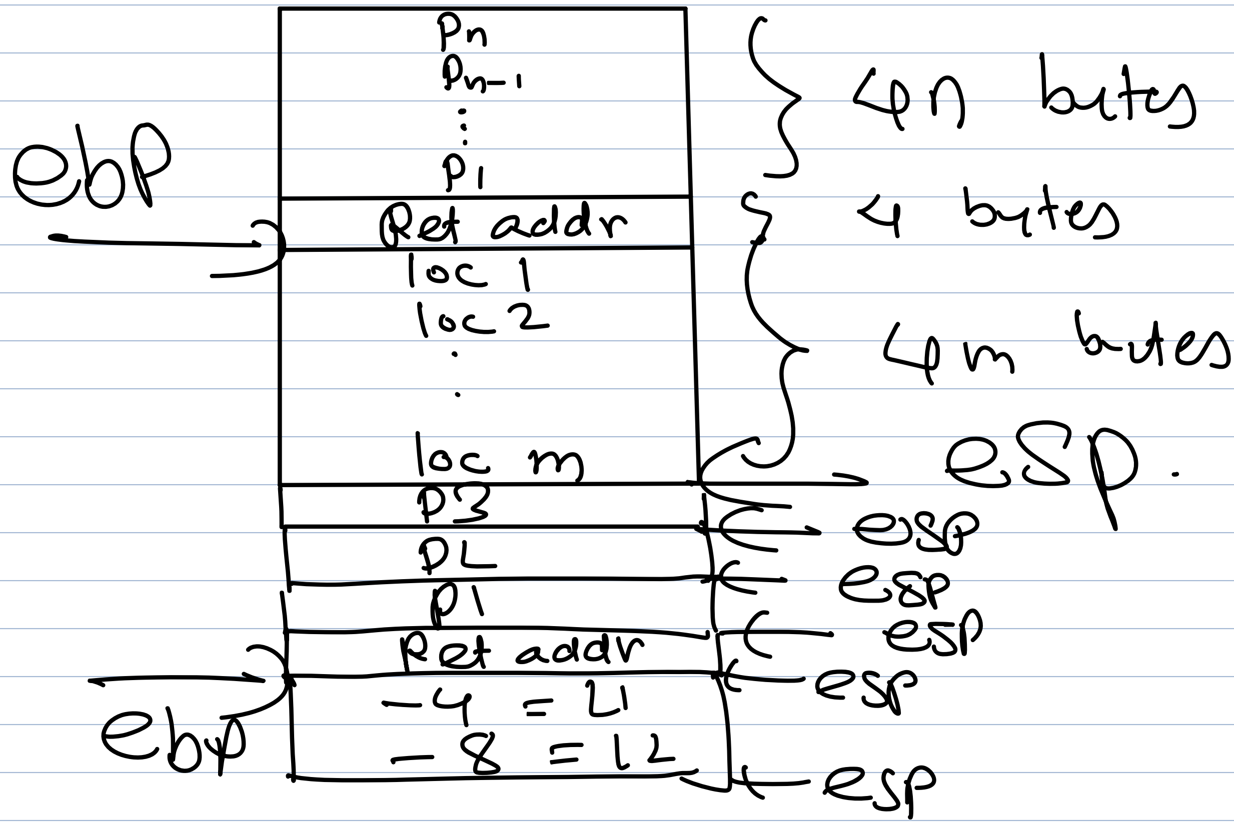
```
ret = fun_name(p1, --, pn);

Tr fun_name(T1 p1, --, Tn pn)
{   // m local variables

      logic

    return V_Tr ;
}
```

Caller.

```
pushl  pn
pushl  pn-1
  .
  .
pushl  p1
call   fun_name
addl   $4n, %esp
movl   %eax, ret
```

Callee:

```
fun_name:
  movl %esp, %ebp
  subl $4m, %esp

    4k(%ebp) = kth param
    -4k(%ebp) = kth local
    logic

  movl —, %eax
  movl %ebp, %esp
  ret
```



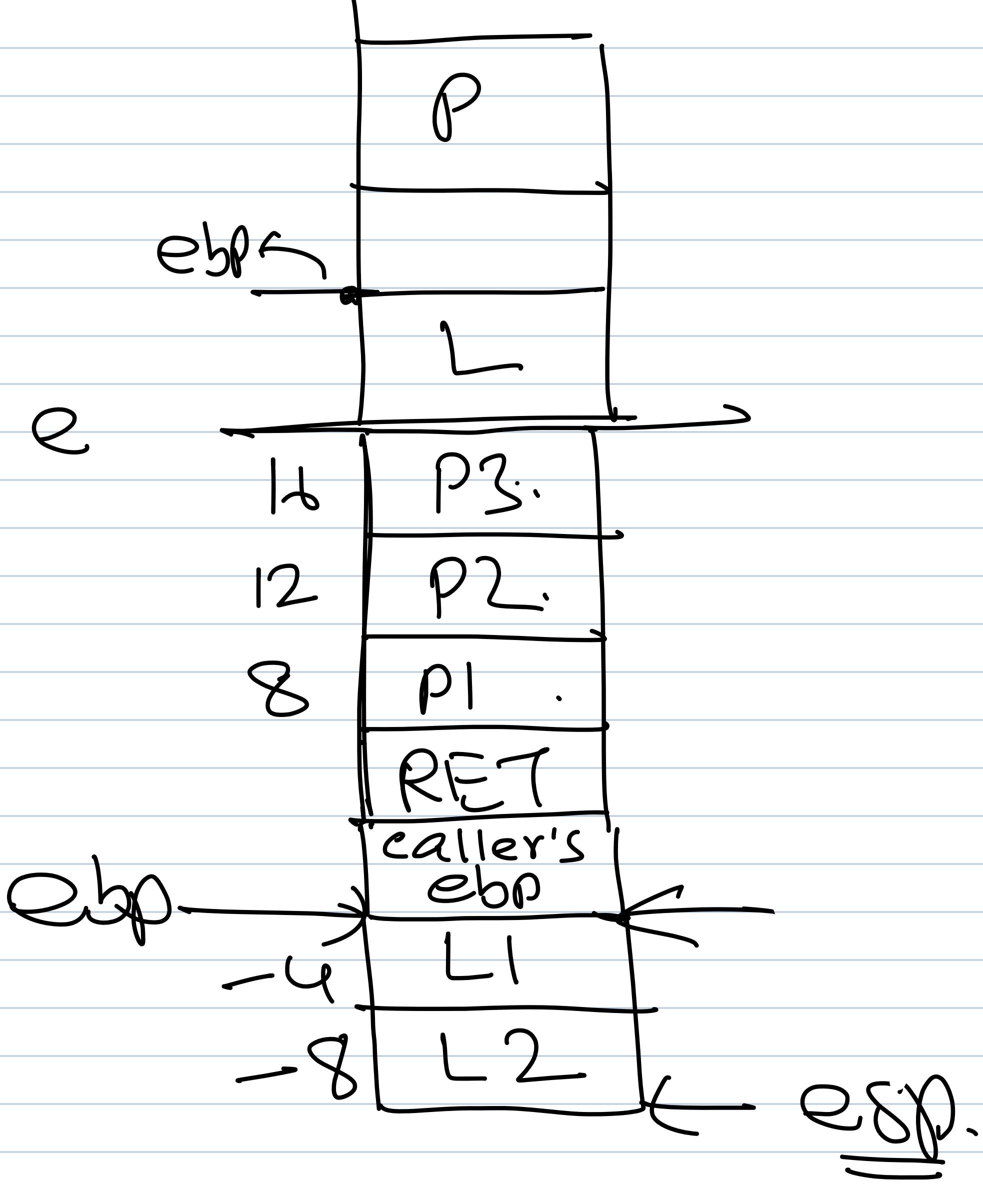

```
pushl %ebp
movl %esp, %ebp
subl $4m, %esp
```

4k+4(%ebp) = kth param

-4k(%ebp) = kth local

logic

```
movl —, %eax
movl %ebp, %esp
popl %ebp
ret
```




```
int f (int a, int b, int c)
{
    int m, q;

    return (—);
}
```

ઉપોદ્ઘાત = Prologue.

વિષયપ્રવેશ =

વિષયવસ્તુ =

ઉપસંહાર = Epilogue.