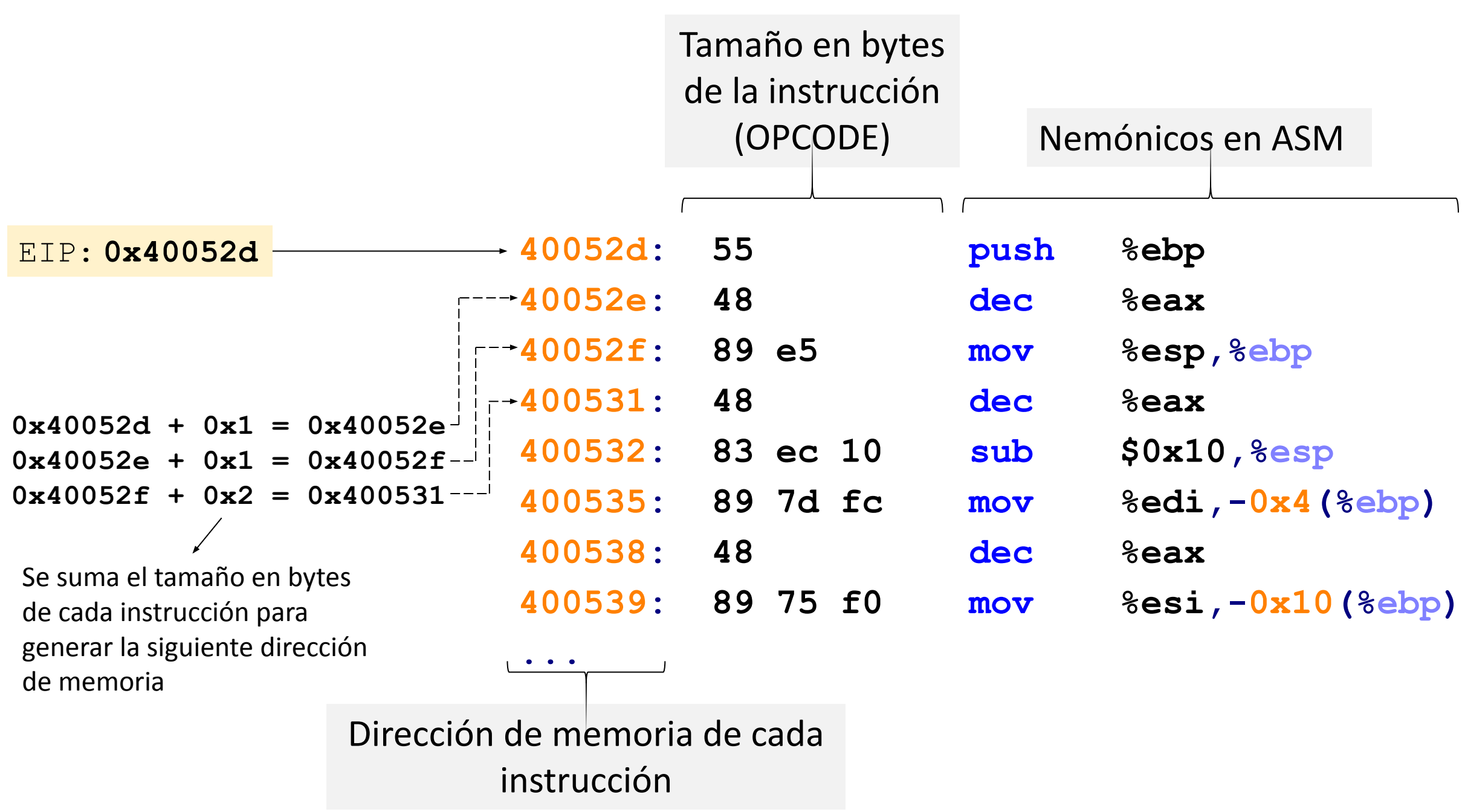
Segmentación de memoria

Un ejemplo práctico

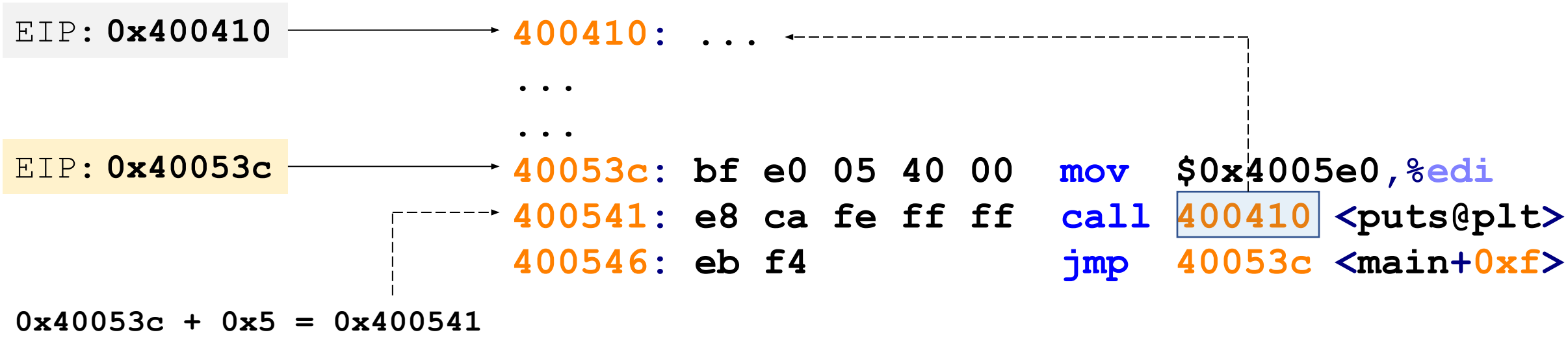
Segmentación de un programa en memoria

- Segmento de código
  - Código del programa
- Ciclo de ejecución del procesador
  1. Leer la instrucción a la que apunta el registro EIP (PC: *Program Counter*)
  2. Sumar a EIP la longitud en bytes de la instrucción leída
     - Esta suma hace que el EIP apunte a la siguiente instrucción en memoria.
  3. Ejecutar la instrucción leída en el paso No. 1
  4. Repetir de nuevo desde el paso No. 1.


Tamaño en bytes de la instrucción (OPCODE)
Nemónicos en ASM
EIP: 0x40052d
40052d: 55 push %ebp
40052e: 48 dec %eax
40052f: 89 e5 mov %esp,%ebp
400531: 48 dec %eax
400532: 83 ec 10 sub $0x10,%esp
400535: 89 7d fc mov %edi,-0x4(%ebp)
400538: 48 dec %eax
400539: 89 75 f0 mov %esi,-0x10(%ebp)
...
0x40052d + 0x1 = 0x40052e
0x40052e + 0x1 = 0x40052f
0x40052f + 0x2 = 0x400531
Se suma el tamaño en bytes de cada instrucción para generar la siguiente dirección de memoria
Dirección de memoria de cada instrucción

Segmentación de un programa en memoria

- Excepción al acceso secuencial
  - Saltos condicionales: **JNZ**, **JNE**, etc.
  - Saltos incondicionales: **JMP**
  - Llamadas a **funciones**: **CALL**



Segmentación de un programa en memoria

- Segmento *stack*
  - Variables locales de una función
  - Parámetros pasados a la función
  - Valor del **EIP** antes del llamado a una función: se *empuja* al stack

Description

Saves procedure linking information on the stack and branches to the called procedure specified using the target operand. The target operand specifies the address of the first instruction in the called procedure. The operand can be an immediate value, a general-purpose register, or a memory location.

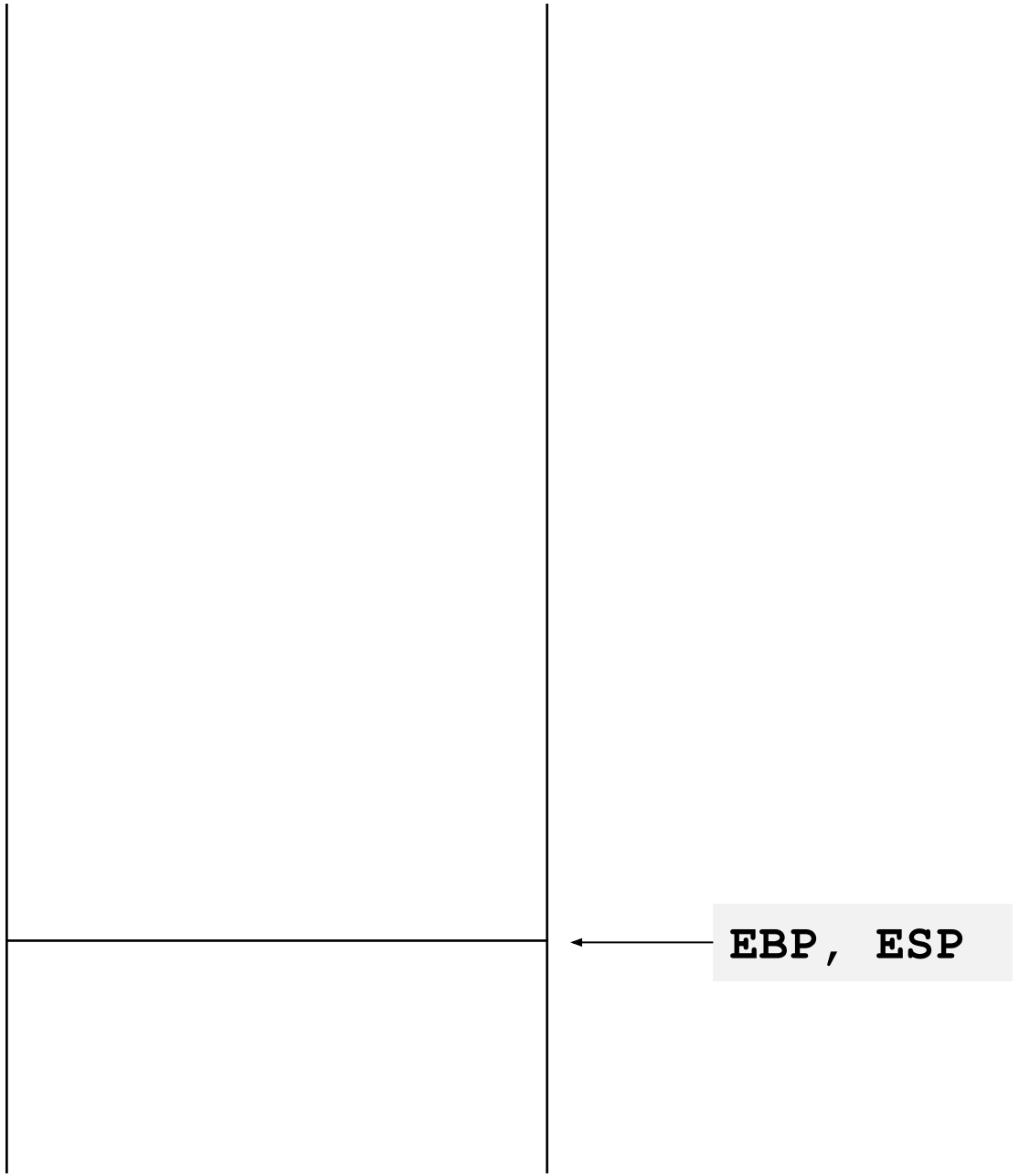
Intel® 64 and IA-32 Architectures Software Developer's Manual

Combined Volumes: 1, 2A, 2B, 2C, 2D, 3A, 3B, 3C, 3D and 4

Segmentación de un programa en memoria

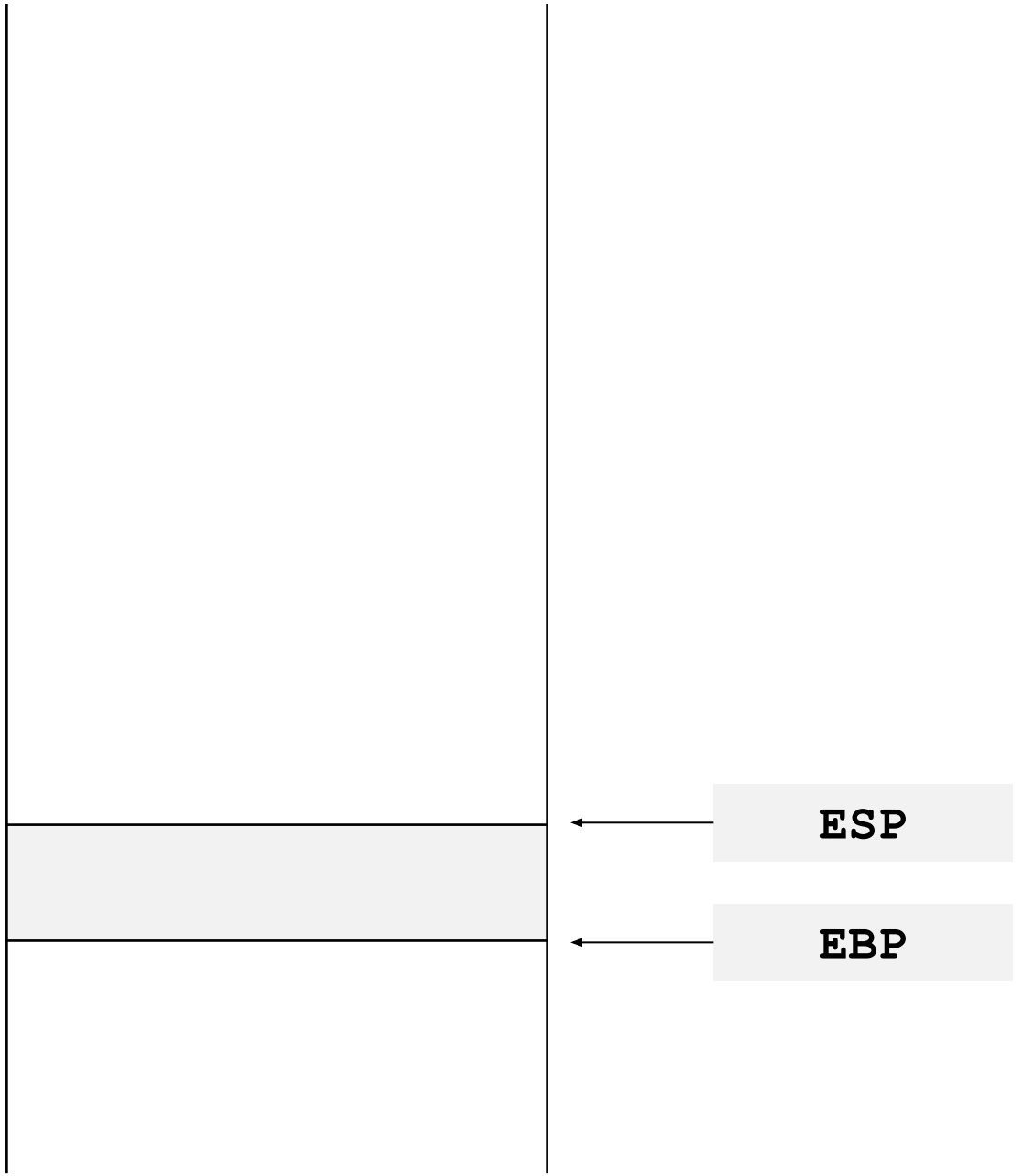
- Registro **ESP** (*stack pointer*) apunta a la dirección de memoria del último elemento en el segmento *stack.*
- Registro **EBP** se usa para indicar el inicio de un *stack frame*.
- La distancia entre el **EBP** y el **ESP** se denomina *stack frame*.
  - Cada función tiene su *stack frame*
- ¿Qué hay en el *stack frame*?
  - Variables locales de una función
  - Parámetros pasados a la función
  - Dirección de retorno (a donde retorna el flujo: valor del **EIP** *empujado*)
  - Apuntador al **EBP** antes del llamado (dónde estaba el **EBP** antes del llamado) .

STACK



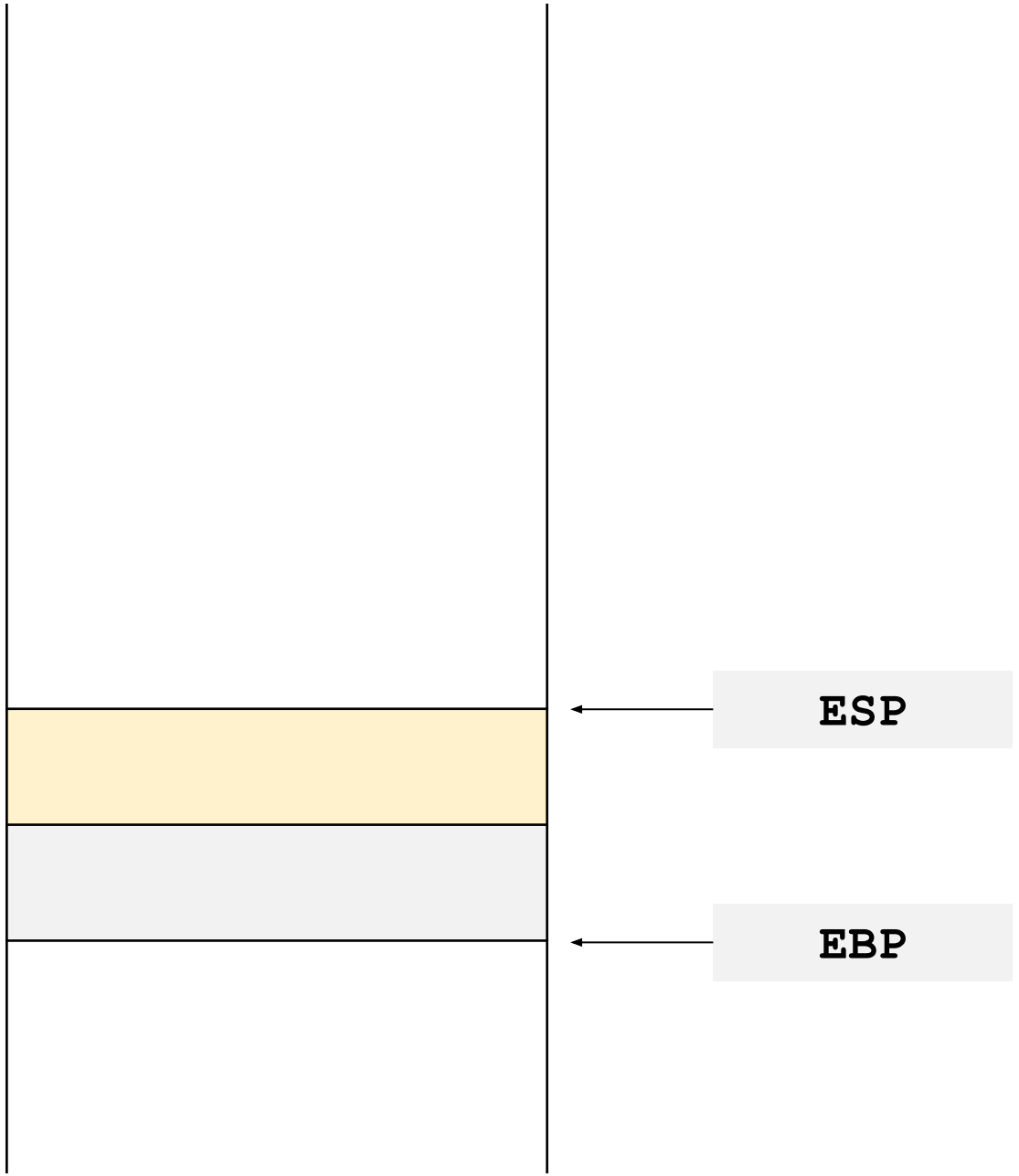
STACK

PUSH



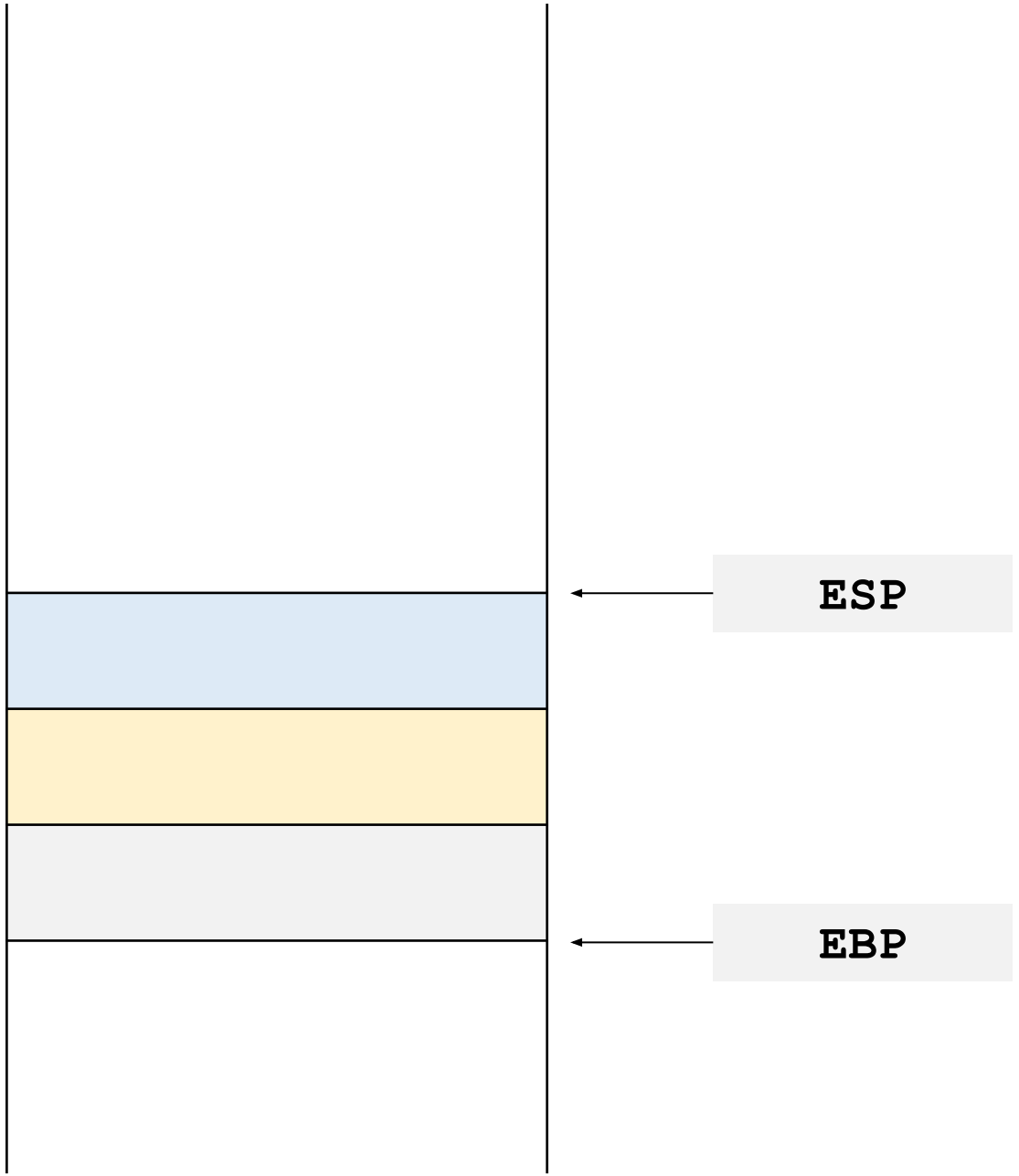
STACK

PUSH



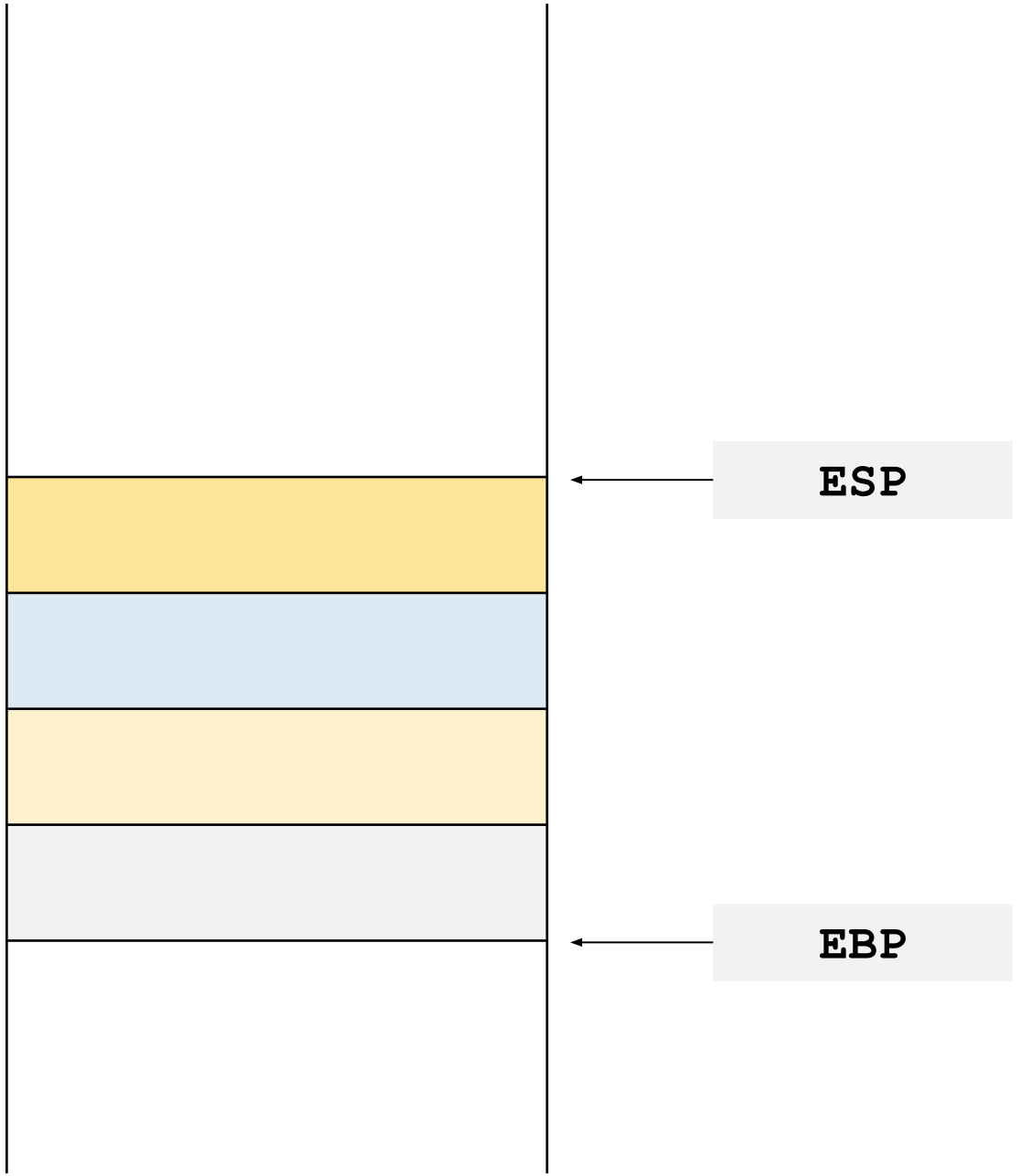
STACK

PUSH

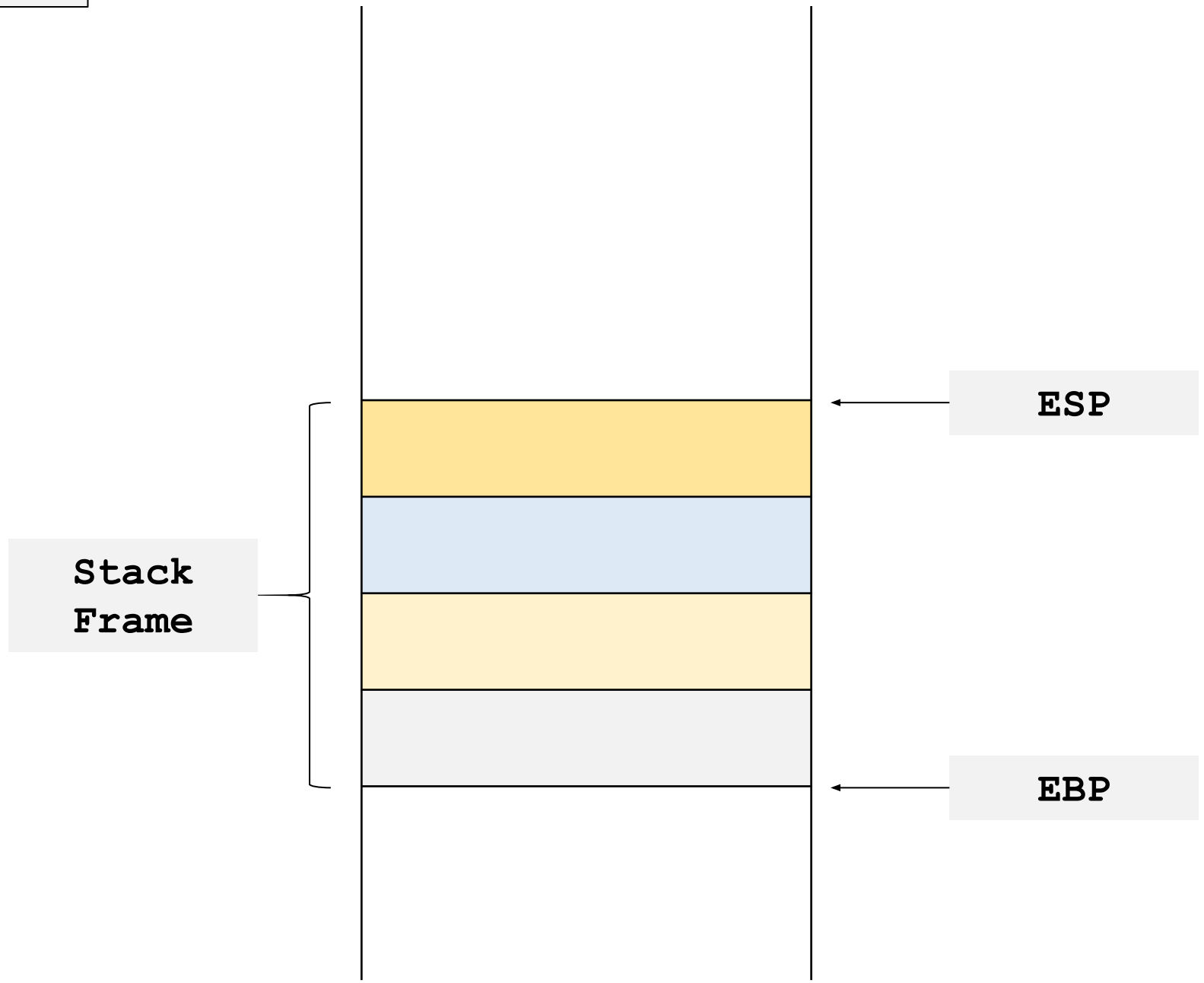


STACK

PUSH

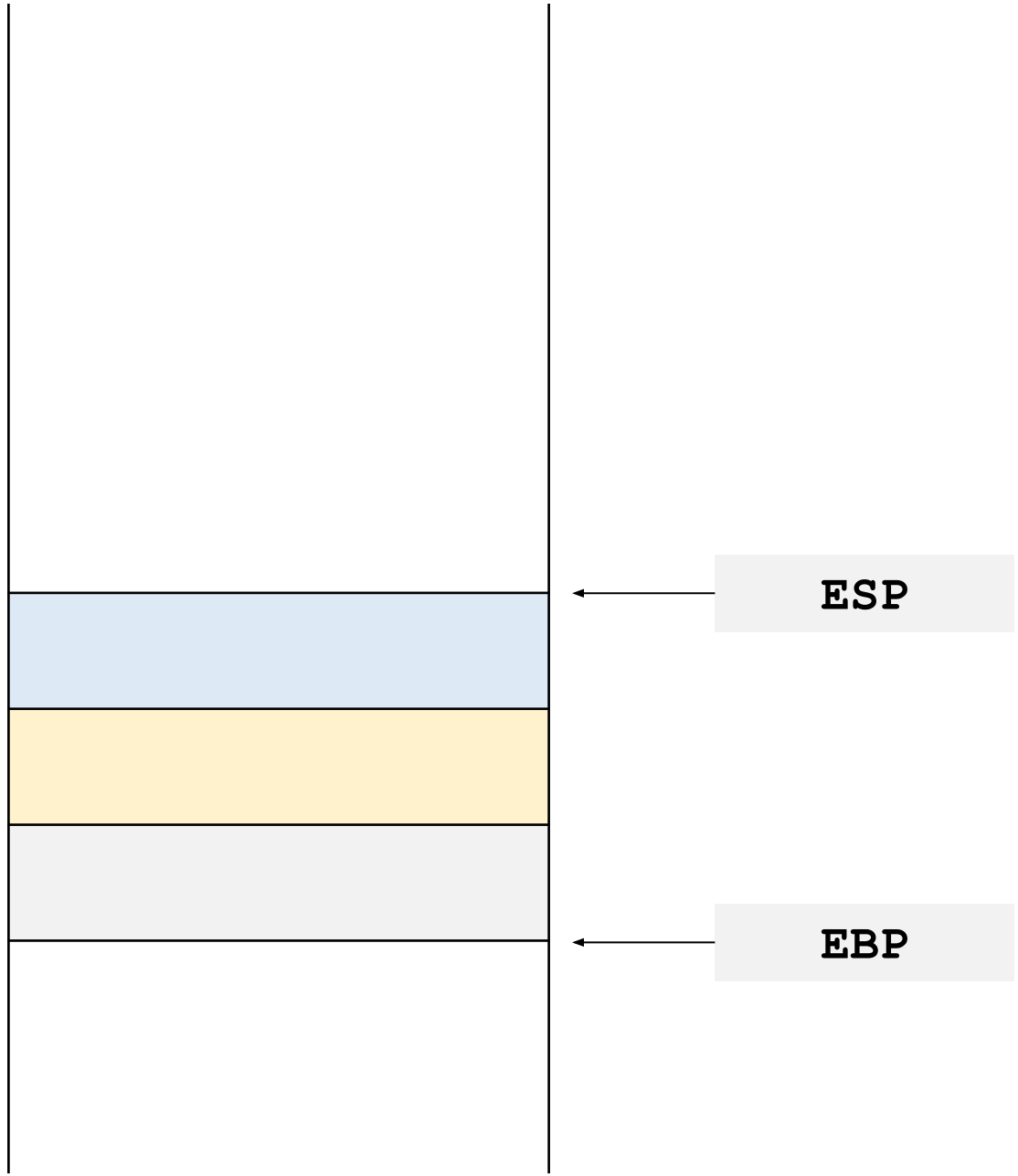


STACK



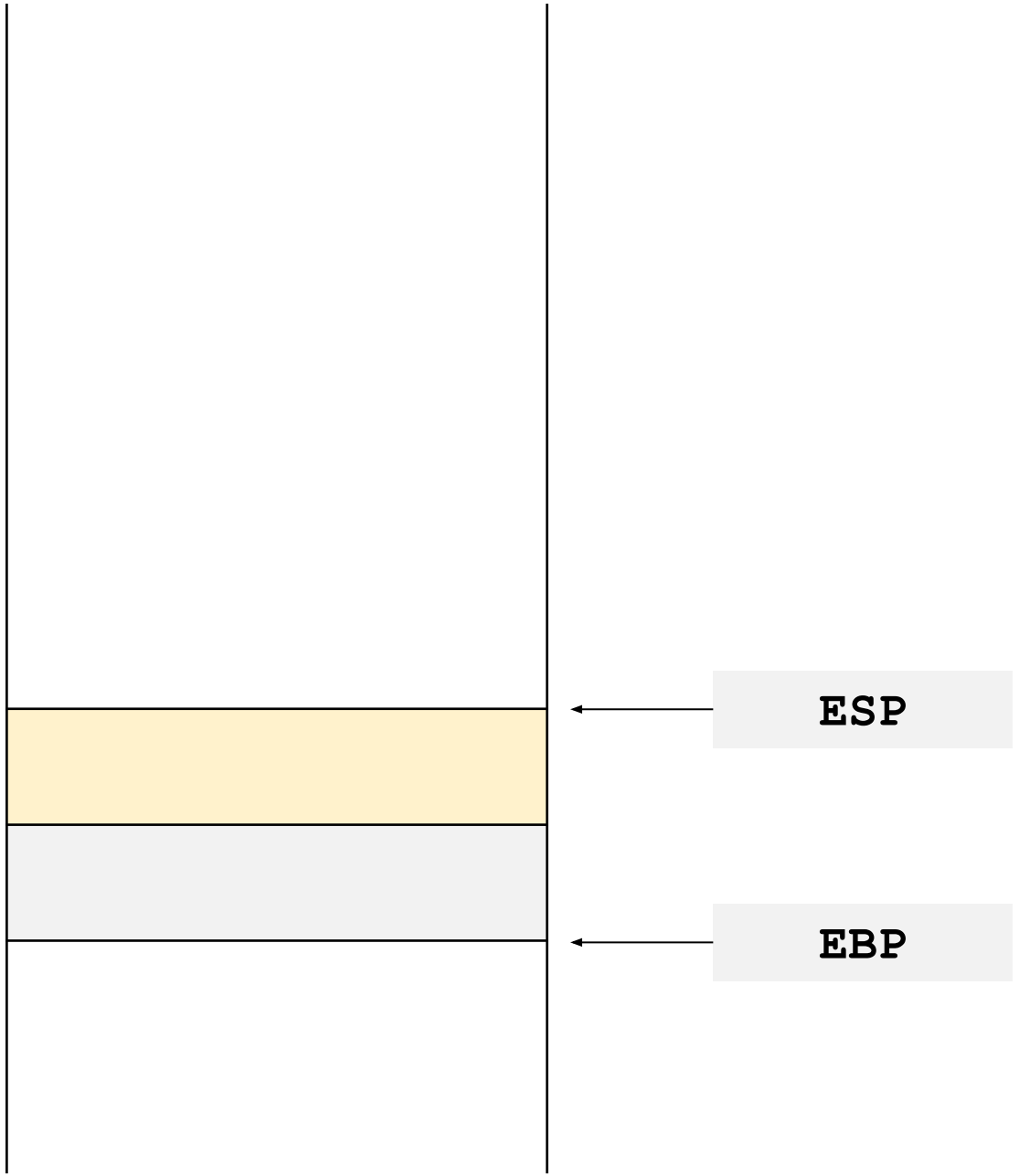
STACK

POP



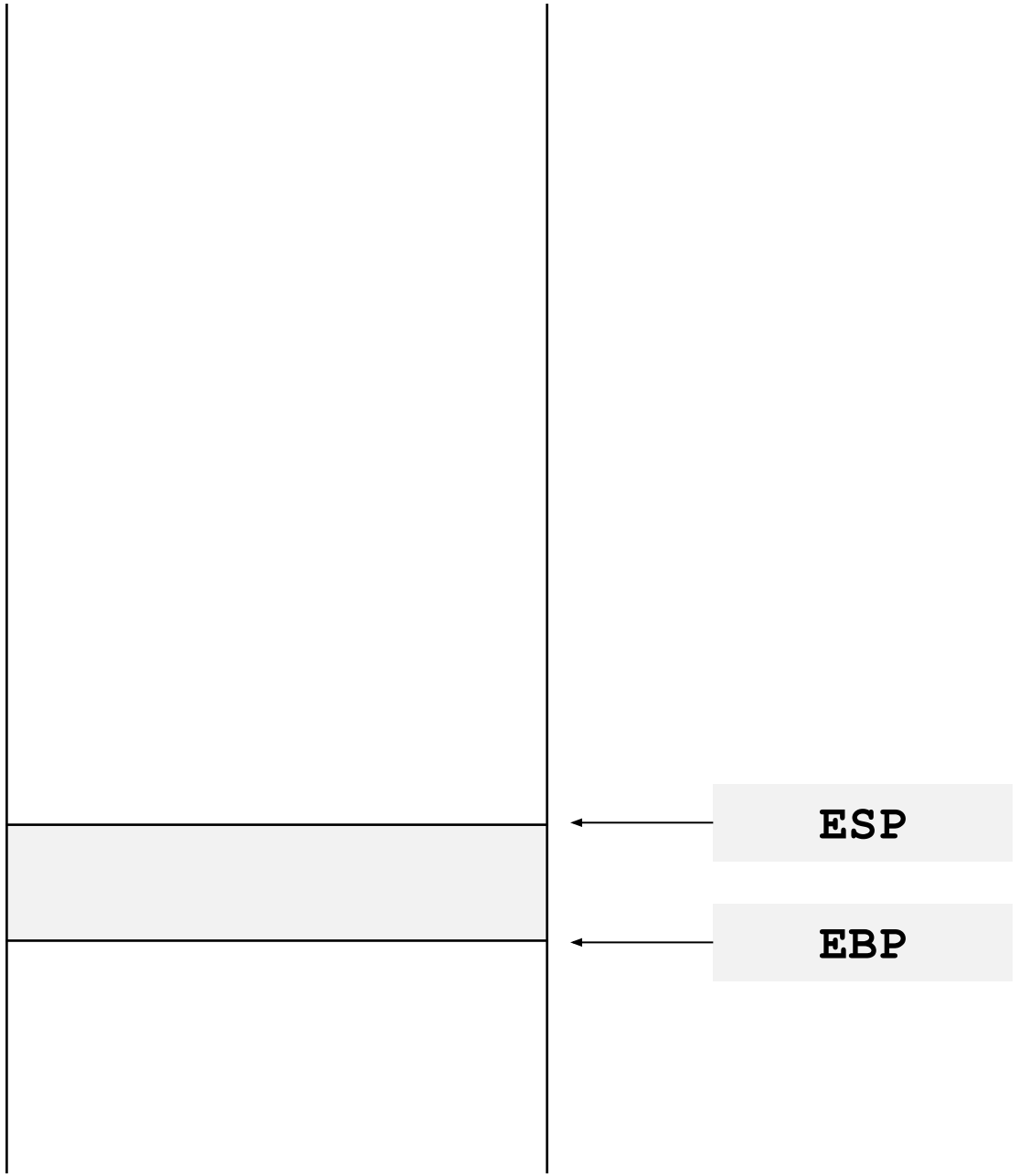
STACK

POP



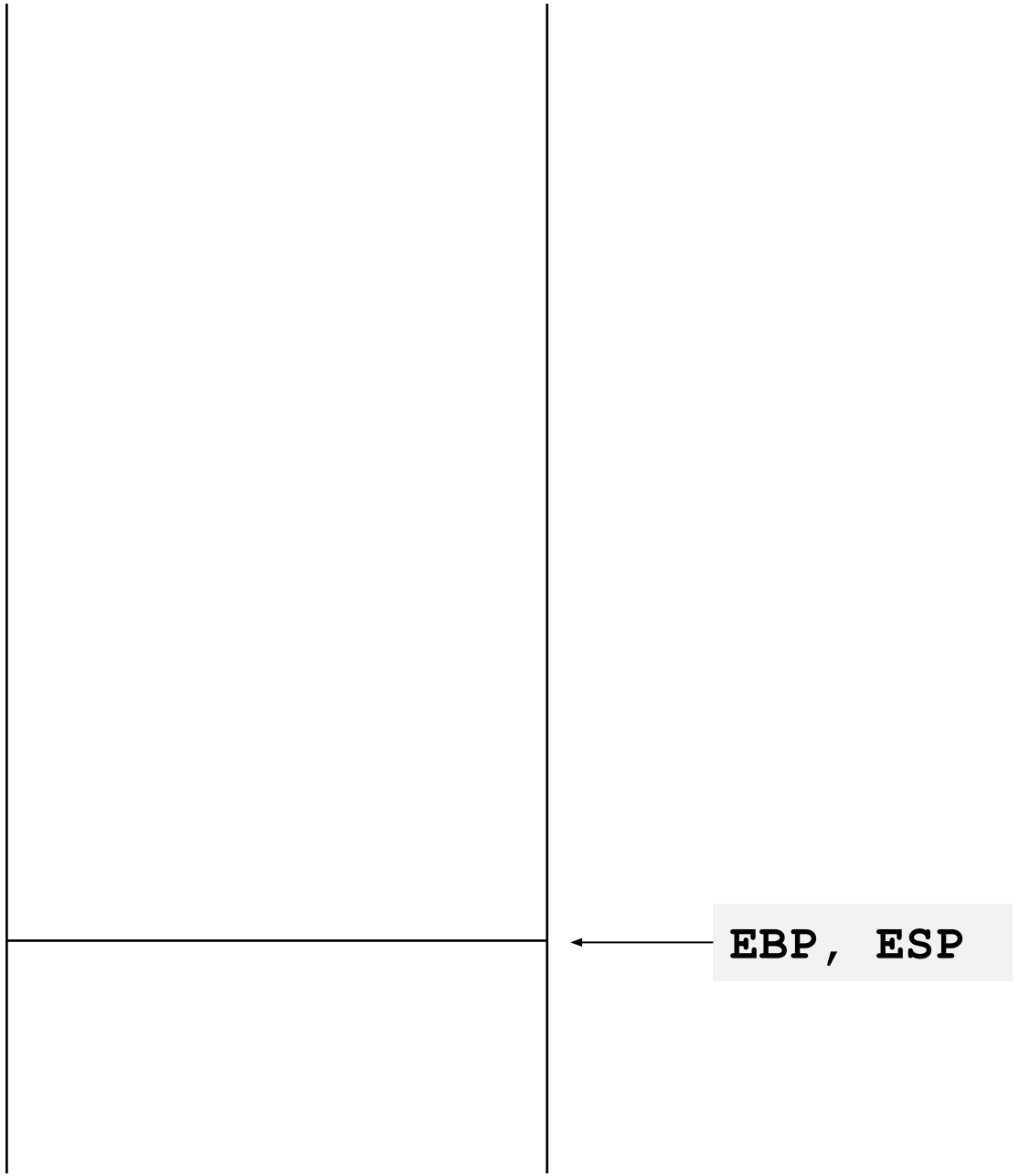
STACK

POP



STACK

POP



```
# gcc -m32 -g -o segmentos segmentos.c
# gdb ejemplo
(gdb) set disassembly-flavor intel
(gdb) disassemble main
```

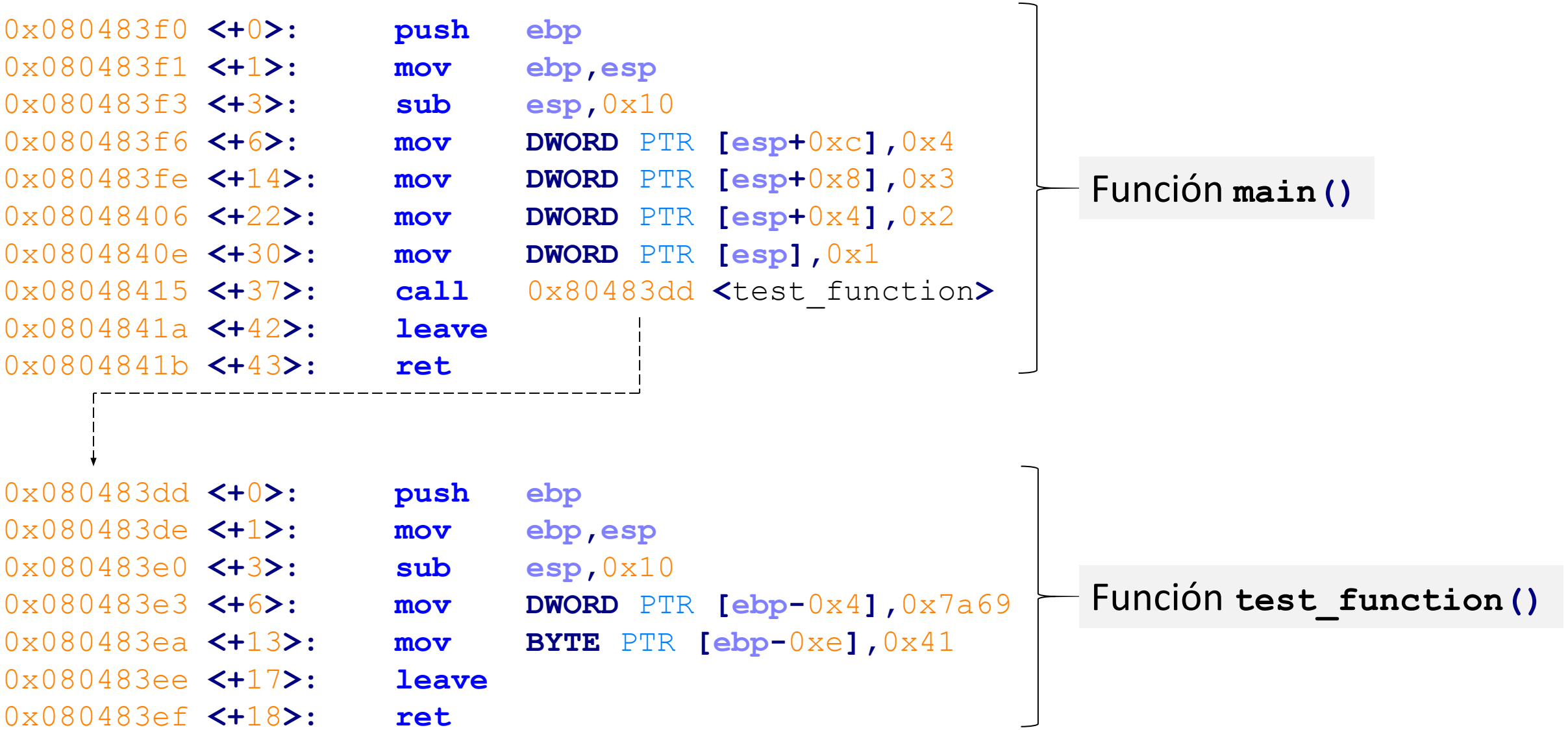
```c
#include <stdlib.h>

void test_function(int a, int b, int c, int d) {
    int flag;
    char buffer[10];
    flag = 31337;
    buffer[0] = 'A';
}

int main() {
    test_function(1, 2, 3, 4);
}
```

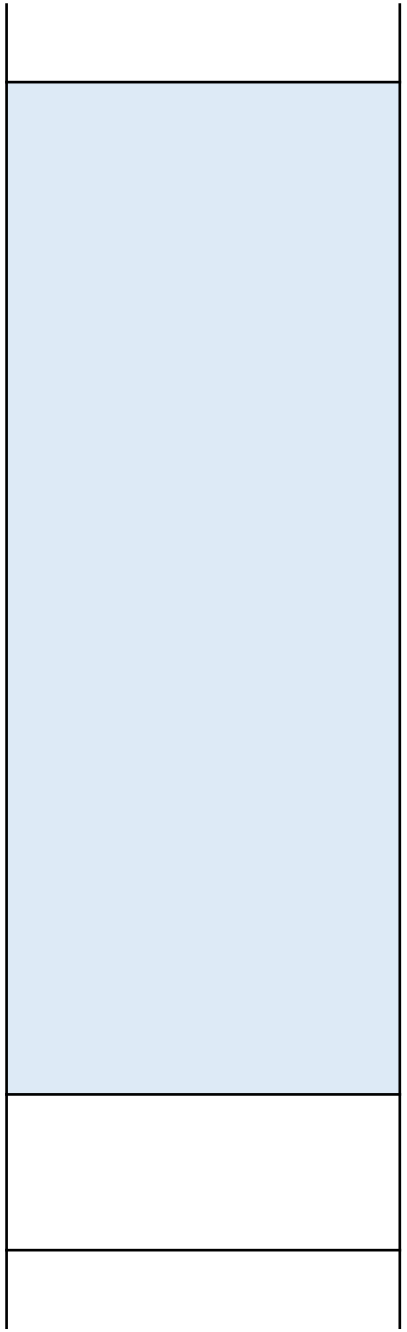
Instalar soporte para compilar a 32 bits

```
# yum install libgcc.i686 glibc-devel.i686
```


0x080483f0 <+0>: push ebp
0x080483f1 <+1>: mov ebp,esp
0x080483f3 <+3>: sub esp,0x10
0x080483f6 <+6>: mov DWORD PTR [esp+0xc],0x4
0x080483fe <+14>: mov DWORD PTR [esp+0x8],0x3
0x08048406 <+22>: mov DWORD PTR [esp+0x4],0x2
0x0804840e <+30>: mov DWORD PTR [esp],0x1
0x08048415 <+37>: call 0x80483dd <test_function>
0x0804841a <+42>: leave
0x0804841b <+43>: ret
Función main()
0x080483dd <+0>: push ebp
0x080483de <+1>: mov ebp,esp
0x080483e0 <+3>: sub esp,0x10
0x080483e3 <+6>: mov DWORD PTR [ebp-0x4],0x7a69
0x080483ea <+13>: mov BYTE PTR [ebp-0xe],0x41
0x080483ee <+17>: leave
0x080483ef <+18>: ret
Función test_function()

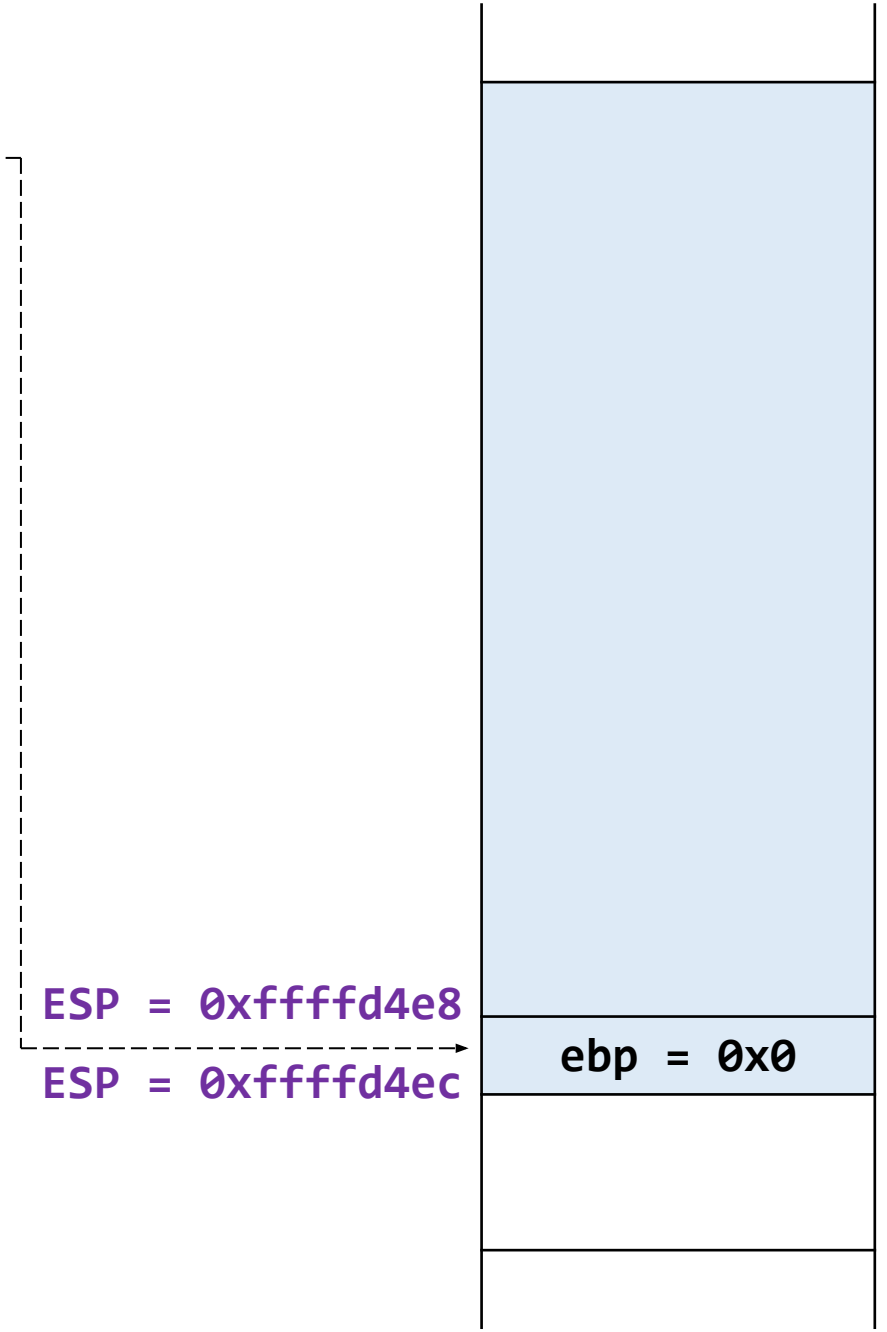
```
(gdb) break main
(gdb) run
(gdb) info proc mappings
(gdb) print $esp
```

```
Start Addr   End Addr       Size     Offset objfile
 0x8048000  0x8049000     0x1000        0x0 /root/segmentos
 0x8049000  0x804a000     0x1000        0x0 /root/segmentos
 0x804a000  0x804b000     0x1000     0x1000 /root/segmentos
0xf7e03000 0xf7e04000     0x1000        0x0
0xf7e04000 0xf7fc8000   0x1c4000        0x0 /usr/lib/libc-2.17.so
0xf7fc8000 0xf7fc9000     0x1000   0x1c4000 /usr/lib/libc-2.17.so
0xf7fc9000 0xf7fcb000     0x2000   0x1c4000 /usr/lib/libc-2.17.so
0xf7fcb000 0xf7fcc000     0x1000   0x1c6000 /usr/lib/libc-2.17.so
0xf7fcc000 0xf7fcf000     0x3000        0x0
0xf7fd8000 0xf7fd9000     0x1000        0x0
0xf7fd9000 0xf7fda000     0x1000        0x0 [vdso]
0xf7fda000 0xf7ffc000    0x22000        0x0 /usr/lib/ld-2.17.so
0xf7ffc000 0xf7ffd000     0x1000    0x21000 /usr/lib/ld-2.17.so
0xf7ffd000 0xf7ffe000     0x1000    0x22000 /usr/lib/ld-2.17.so
0xfffdd000 0xffffe000    0x21000        0x0 [stack]
```



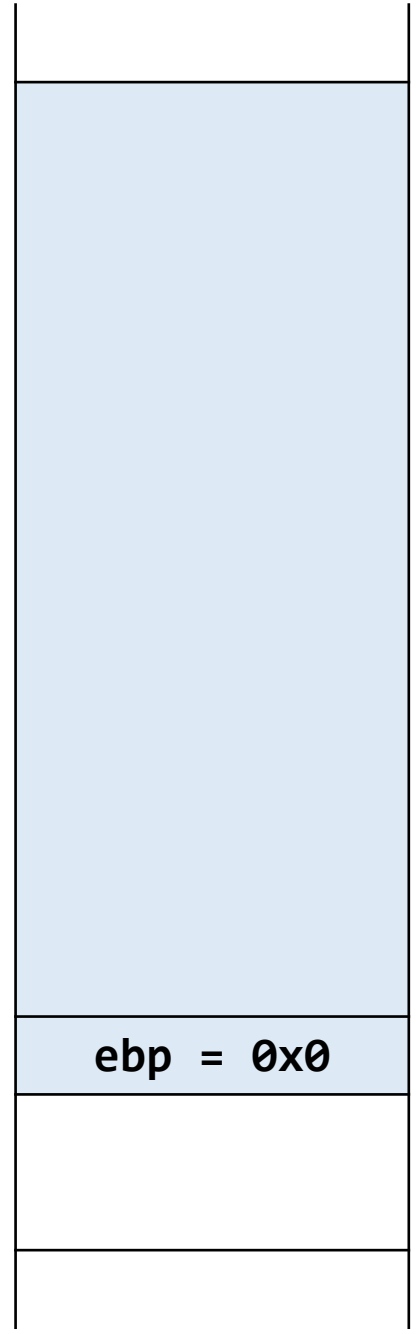
```
0x080483f0 <+0>:     push   ebp
0x080483f1 <+1>:     mov    ebp,esp
0x080483f3 <+3>:     sub    esp,0x10
```

- Inicialmente **EBP = 0x0**
- Inicialmente **ESP = 0xffffd4ec**
- Cuando se hace **push ebp** se empujan **4** bytes al *stack.*
- El **ESP** debe actualizarse restando **4** a la dirección **0xffffd4ec**
  - **ESP – 4 = 0xffffd4e8**
- Ahora **ESP** apunta a la siguiente dirección disponible en el *stack*



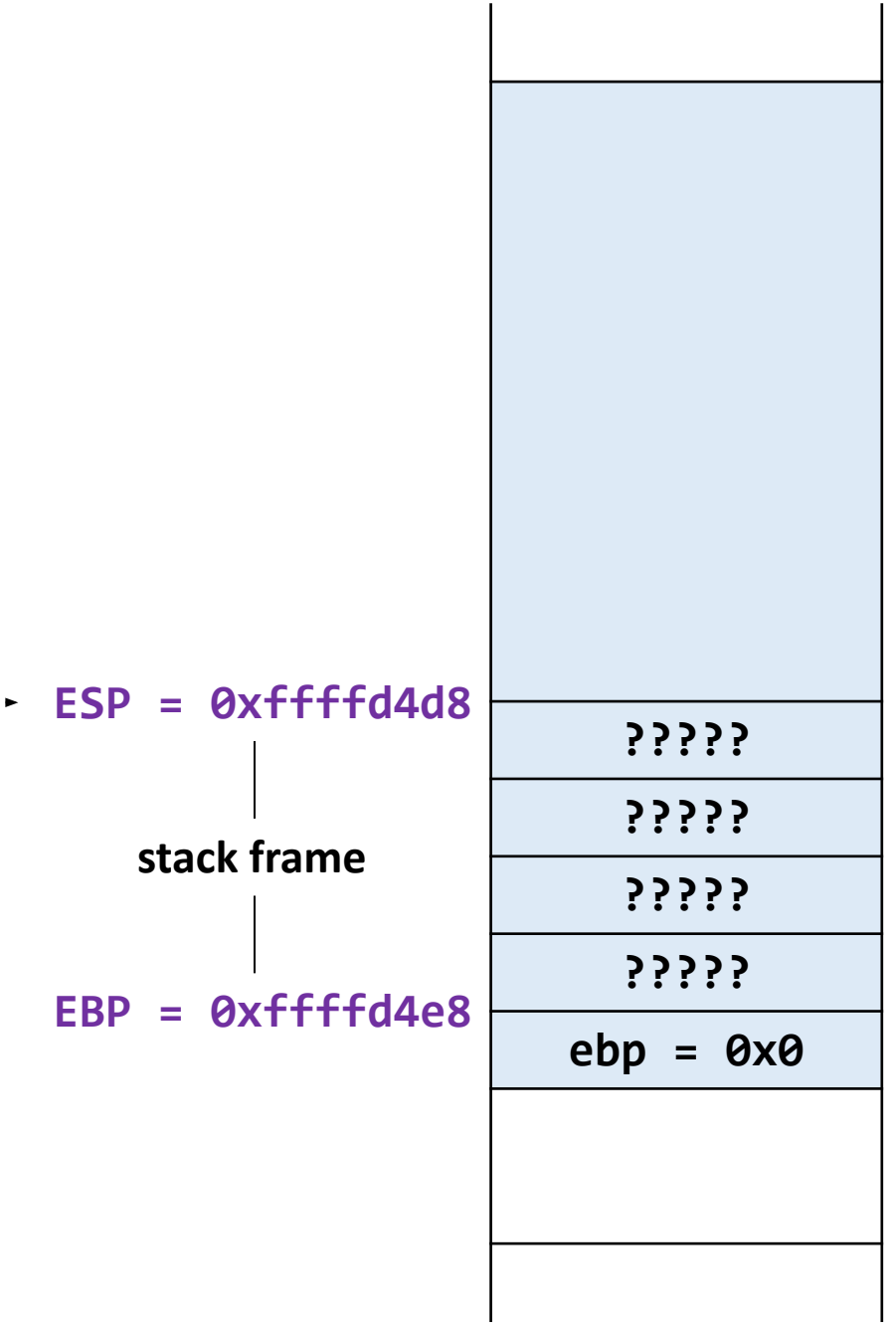
```
0x080483f0 <+0>:     push   ebp
0x080483f1 <+1>:     mov    ebp,esp
0x080483f3 <+3>:     sub    esp,0x10
```

EBP = ESP = 0xffffd4e8



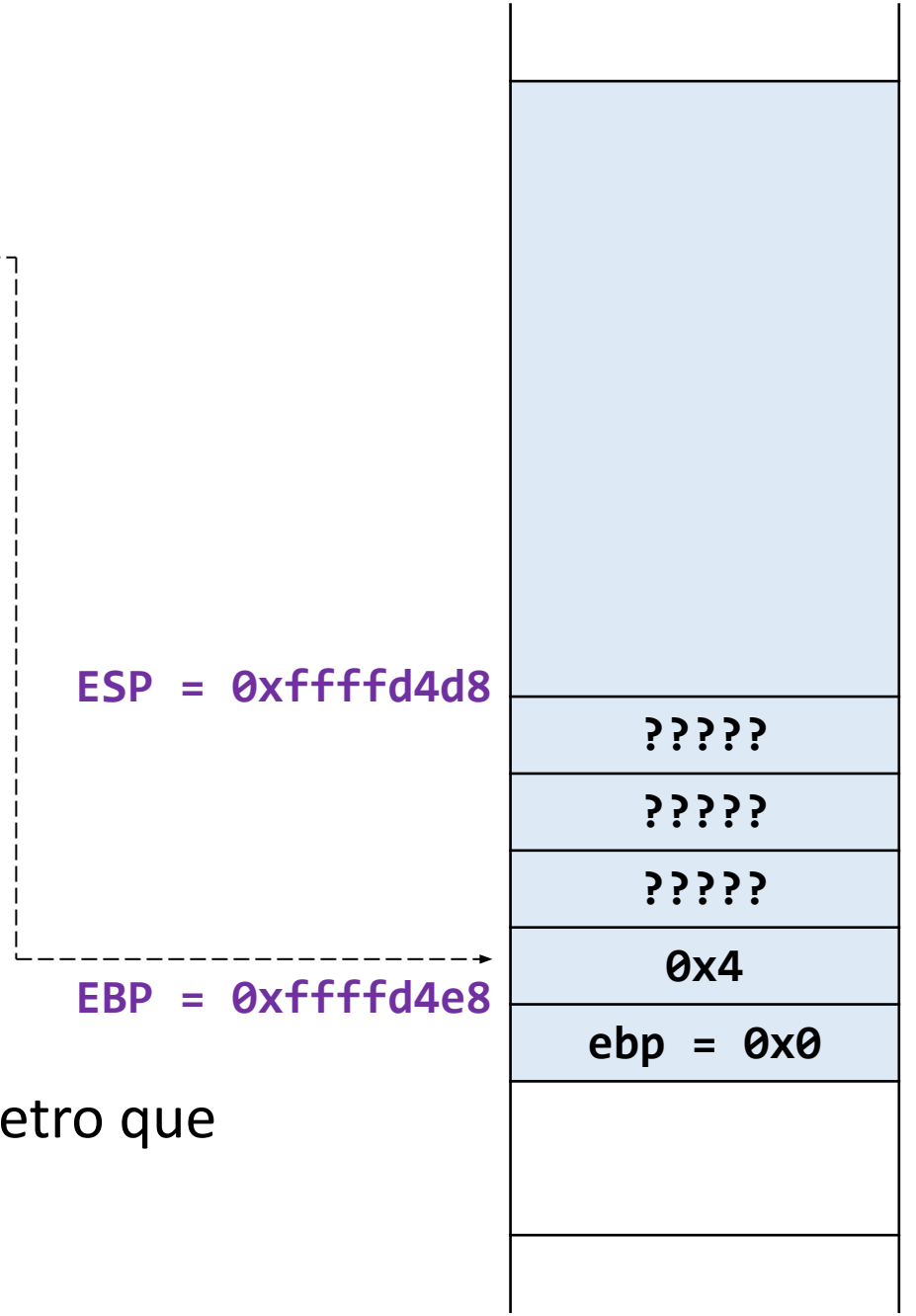
- Se iguala **EBP** a **ESP** para preparar un *stack frame* para la función **`main()`**.
- Estamos analizando las primeras líneas de código de la función **`main()`**.

```
0x080483f0 <+0>:     push   ebp
0x080483f1 <+1>:     mov    ebp,esp
0x080483f3 <+3>:     sub    esp,0x10
```



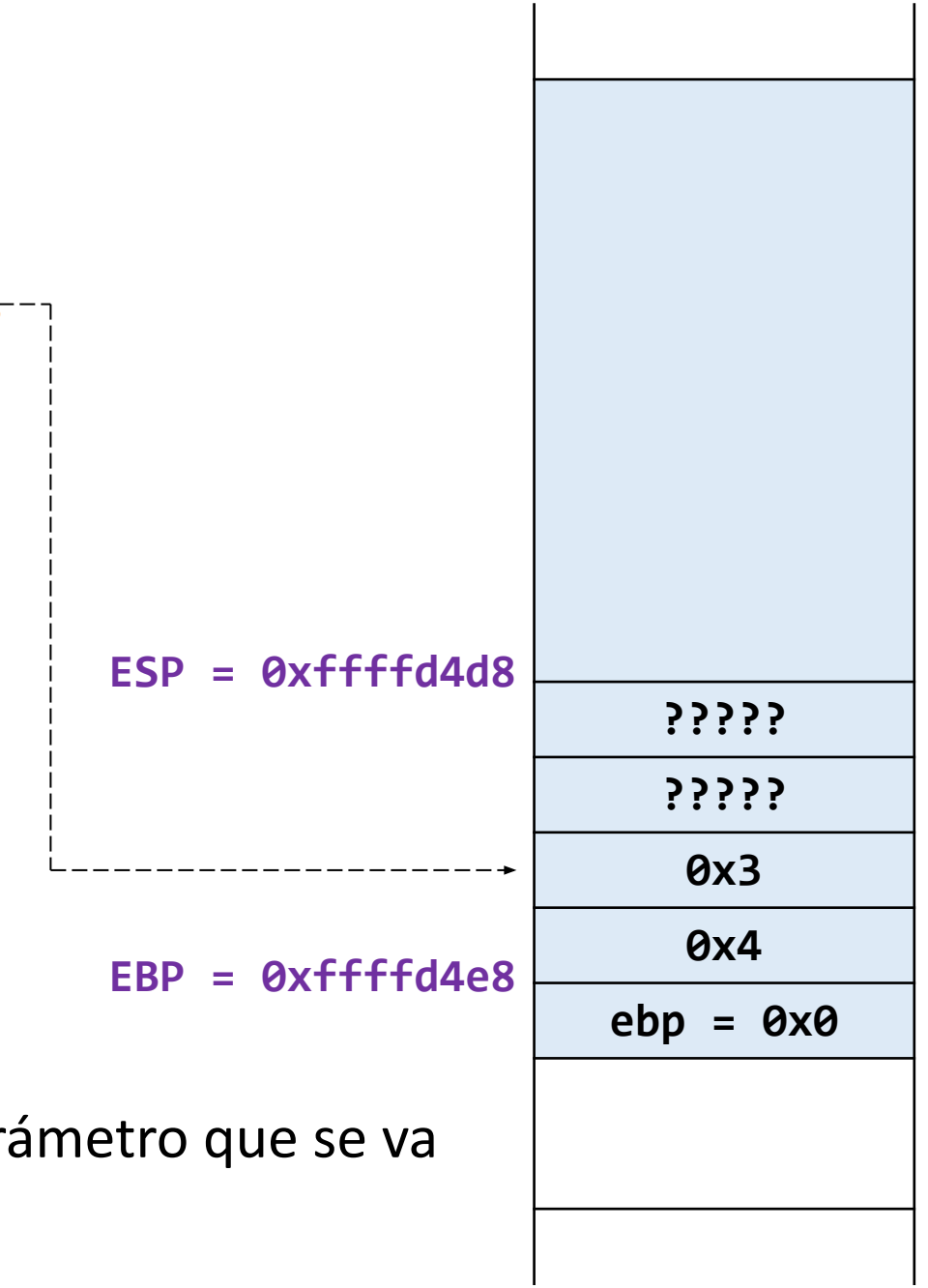
- Se restan 16 bytes (**0x10**) al registro **ESP**. El resultado de la resta se almacena en el registro **ESP**.
  - **ESP - 0x10 = 0xffffd4d8**
- En este punto es donde se crea el *stack frame* para la función **main()**.

```
0x080483f0 <+0>:     push   ebp
0x080483f1 <+1>:     mov    ebp,esp
0x080483f3 <+3>:     sub    esp,0x10
0x080483f6 <+6>:     mov    DWORD PTR [esp+0xc],0x4
0x080483fe <+14>:    mov    DWORD PTR [esp+0x8],0x3
0x08048406 <+22>:    mov    DWORD PTR [esp+0x4],0x2
0x0804840e <+30>:    mov    DWORD PTR [esp],0x1
```



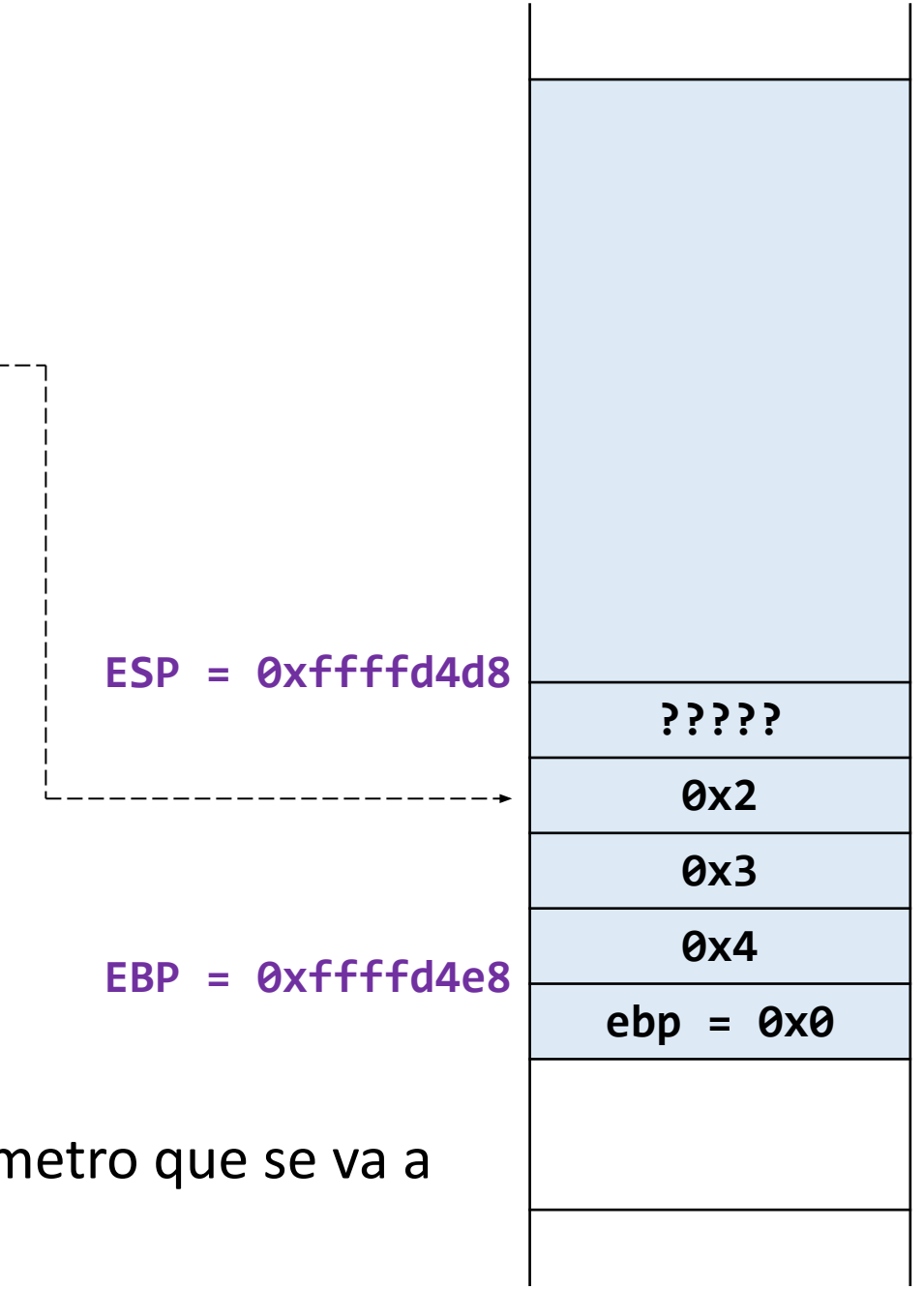
- Se almacena en **ESP + 0xc (ESP + 12)** el último parámetro que se va a pasar a la función **test_function()**.
  - **ESP + 0xc = 0xffffd4e4**

```
0x080483f0 <+0>:     push   ebp
0x080483f1 <+1>:     mov    ebp,esp
0x080483f3 <+3>:     sub    esp,0x10
0x080483f6 <+6>:     mov    DWORD PTR [esp+0xc],0x4
0x080483fe <+14>:    mov    DWORD PTR [esp+0x8],0x3
0x08048406 <+22>:    mov    DWORD PTR [esp+0x4],0x2
0x0804840e <+30>:    mov    DWORD PTR [esp],0x1
```



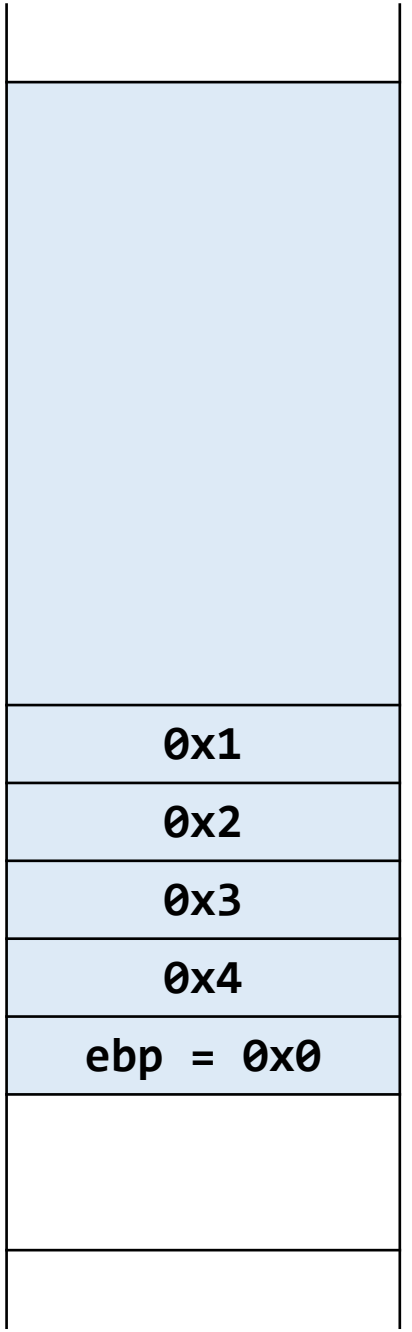
- De manera similar se empuja a la pila el penúltimo parámetro que se va a pasar a la función **`test_function()`**.

```
0x080483f0 <+0>:     push   ebp
0x080483f1 <+1>:     mov    ebp,esp
0x080483f3 <+3>:     sub    esp,0x10
0x080483f6 <+6>:     mov    DWORD PTR [esp+0xc],0x4
0x080483fe <+14>:    mov    DWORD PTR [esp+0x8],0x3
0x08048406 <+22>:    mov    DWORD PTR [esp+0x4],0x2
0x0804840e <+30>:    mov    DWORD PTR [esp],0x1
```



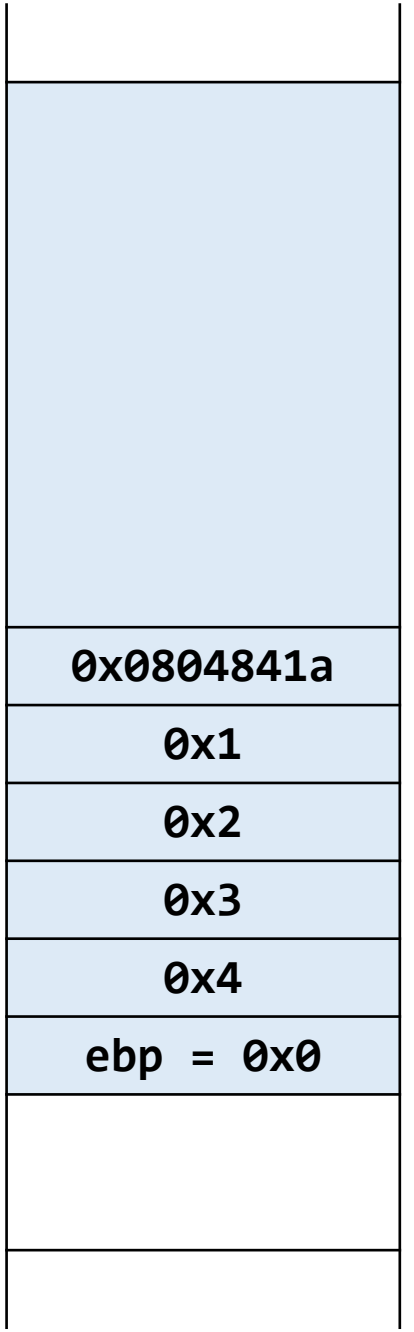
- De manera similar se empuja a la pila el segundo parámetro que se va a pasar a la función **`test_function()`**.

```
0x080483f0 <+0>:     push   ebp
0x080483f1 <+1>:     mov    ebp,esp
0x080483f3 <+3>:     sub    esp,0x10
0x080483f6 <+6>:     mov    DWORD PTR [esp+0xc],0x4
0x080483fe <+14>:    mov    DWORD PTR [esp+0x8],0x3
0x08048406 <+22>:    mov    DWORD PTR [esp+0x4],0x2
0x0804840e <+30>:    mov    DWORD PTR [esp],0x1
```



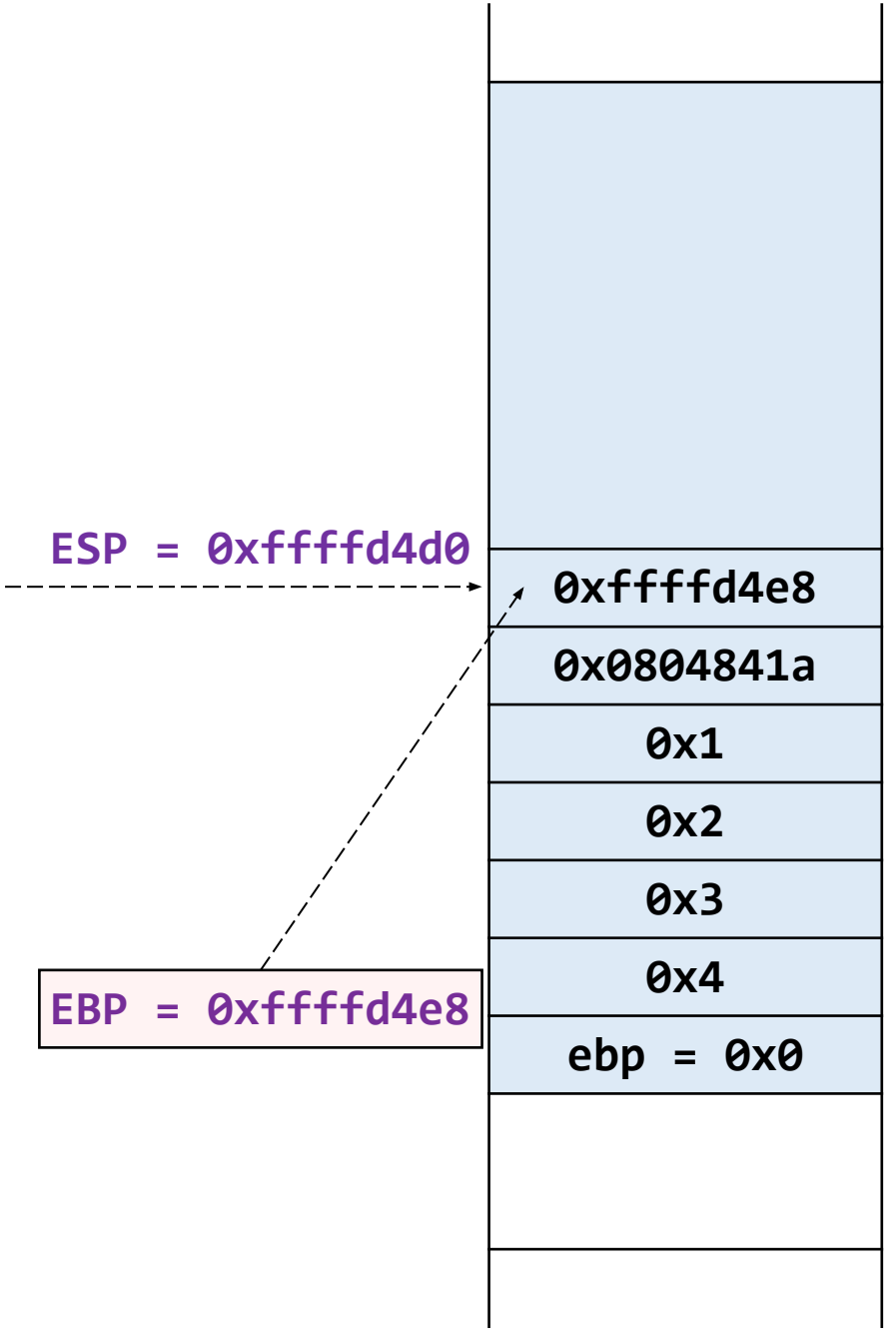
- De manera similar se empuja a la pila el primer parámetro que se va a pasar a la función **`test_function()`**.

```
0x080483f0 <+0>:     push   ebp
0x080483f1 <+1>:     mov    ebp,esp
0x080483f3 <+3>:     sub    esp,0x10
0x080483f6 <+6>:     mov    DWORD PTR [esp+0xc],0x4
0x080483fe <+14>:    mov    DWORD PTR [esp+0x8],0x3
0x08048406 <+22>:    mov    DWORD PTR [esp+0x4],0x2
0x0804840e <+30>:    mov    DWORD PTR [esp],0x1
0x08048415 <+37>:    call   0x80483dd <test function>
0x0804841a <+42>:    leave
0x0804841b <+43>:    ret
```



- Empuja a la pila la dirección posterior al **CALL**
  - **0x0804841a**
- Se actualiza el **ESP** restando al valor actual 4 bytes
  - **ESP = ESP - 4 = 0xffffd4d4**
- **EIP** apunta ahora a **0x80483dd**
  - La primera instrucción de la función **test_function()**.

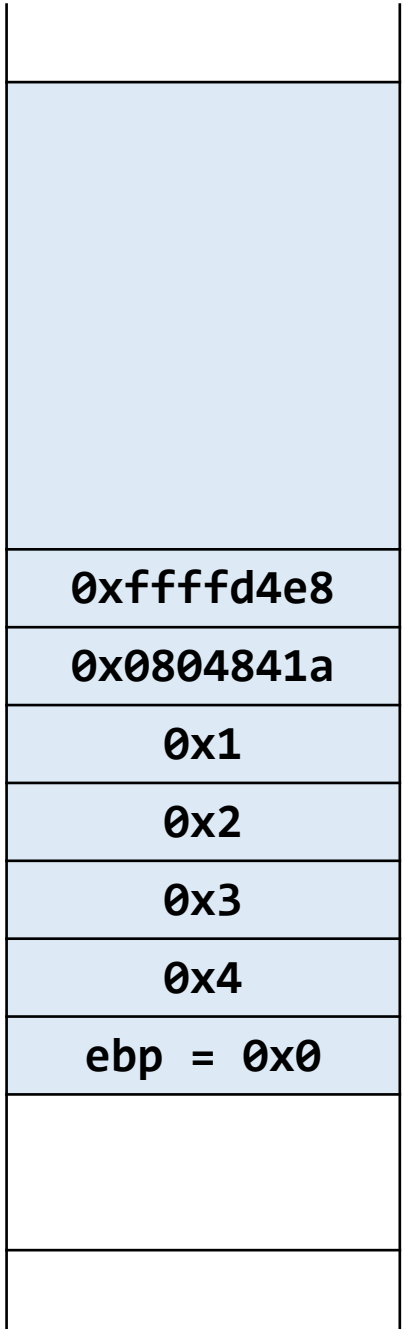
```
0x080483dd <+0>:    push   ebp
0x080483de <+1>:    mov    ebp,esp
0x080483e0 <+3>:    sub    esp,0x10
0x080483e3 <+6>:    mov    DWORD PTR [ebp-0x4],0x7a69
0x080483ea <+13>:   mov    BYTE PTR [ebp-0xe],0x41
0x080483ee <+17>:   leave
0x080483ef <+18>:   ret
```



- Ahora estamos en el código de la función **`test_function()`**
- Se empuja a la pila la dirección actual del **EBP**
  - Se está preparando otro *stack frame*: el de la función **`test_function()`**
- Se actualiza el **ESP** restando al valor actual 4 bytes
  - **`ESP = ESP - 4 = 0xffffd4d0`**

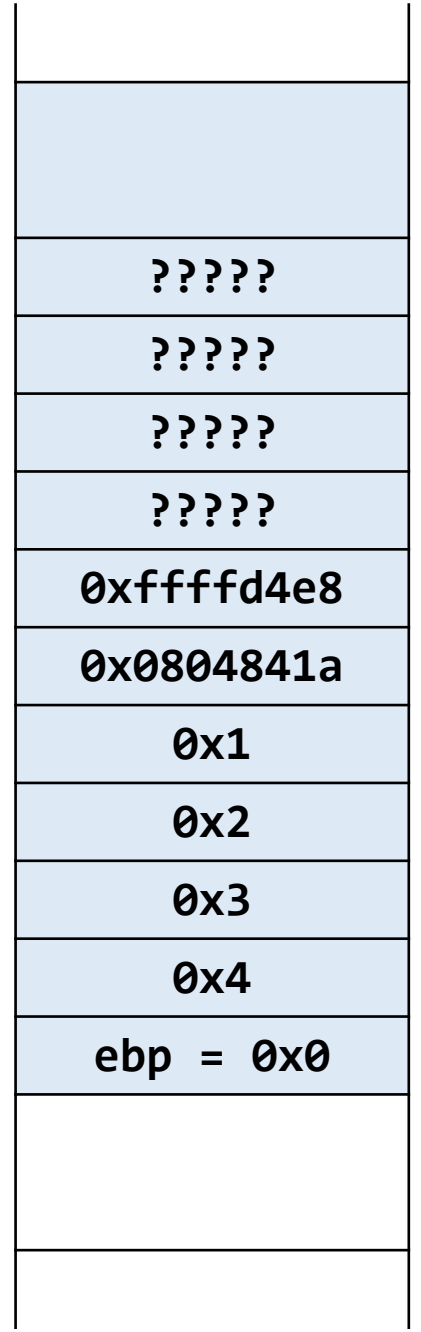
```
0x080483dd <+0>:     push   ebp
0x080483de <+1>:     mov    ebp,esp
0x080483e0 <+3>:     sub    esp,0x10
0x080483e3 <+6>:     mov    DWORD PTR [ebp-0x4],0x7a69
0x080483ea <+13>:    mov    BYTE PTR [ebp-0xe],0x41
0x080483ee <+17>:    leave
0x080483ef <+18>:    ret
```

EBP = ESP = 0xffffd4d0



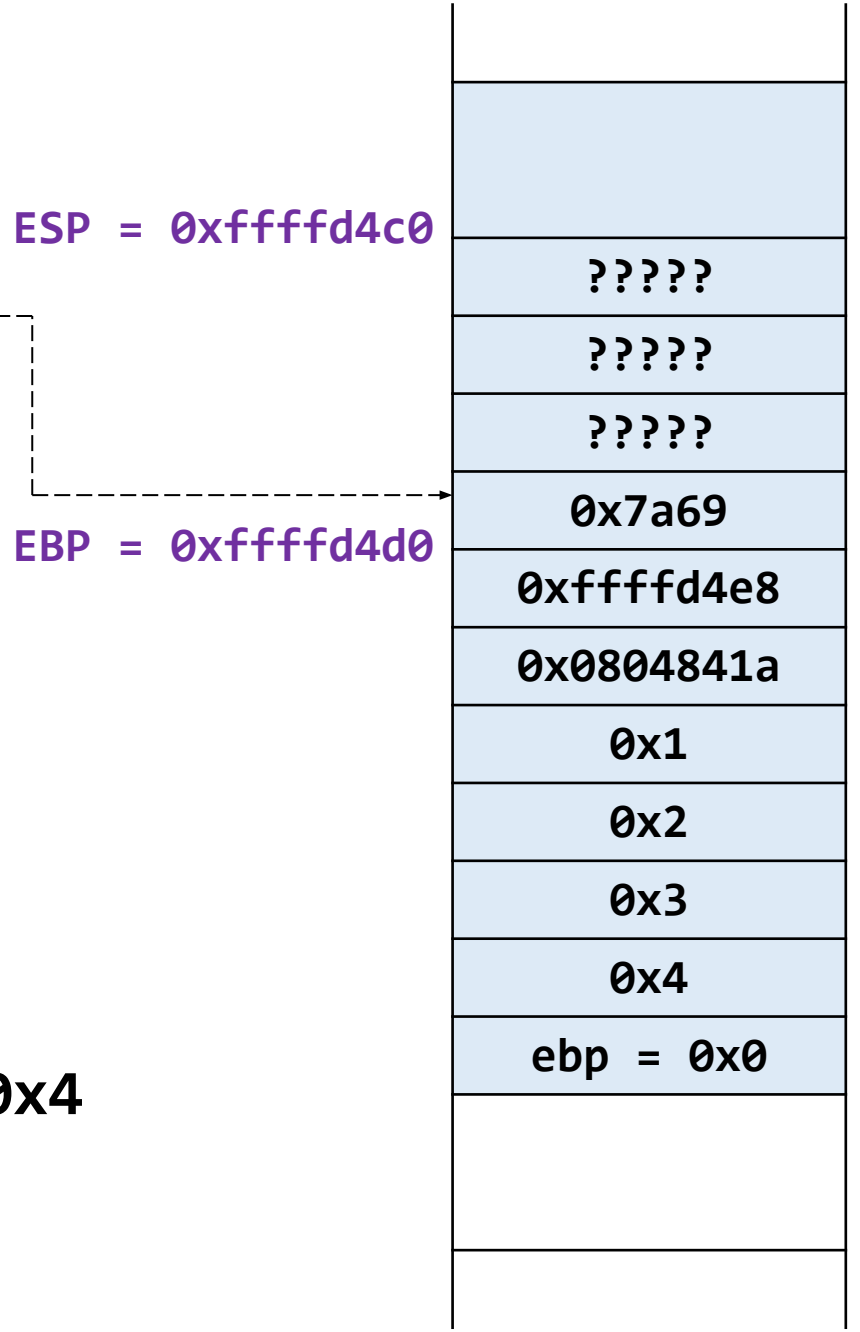
- Se asigna al **EBP** el valor del **ESP**
  - Prólogo de la función preparando un *stack frame*.

```
0x080483dd <+0>:     push   ebp
0x080483de <+1>:     mov    ebp,esp
0x080483e0 <+3>:     sub    esp,0x10 ----------------------> ESP = 0xffffd4c0
0x080483e3 <+6>:     mov    DWORD PTR [ebp-0x4],0x7a69
0x080483ea <+13>:    mov    BYTE PTR [ebp-0xe],0x41
0x080483ee <+17>:    leave
0x080483ef <+18>:    ret
                                                  EBP = 0xffffd4d0
```



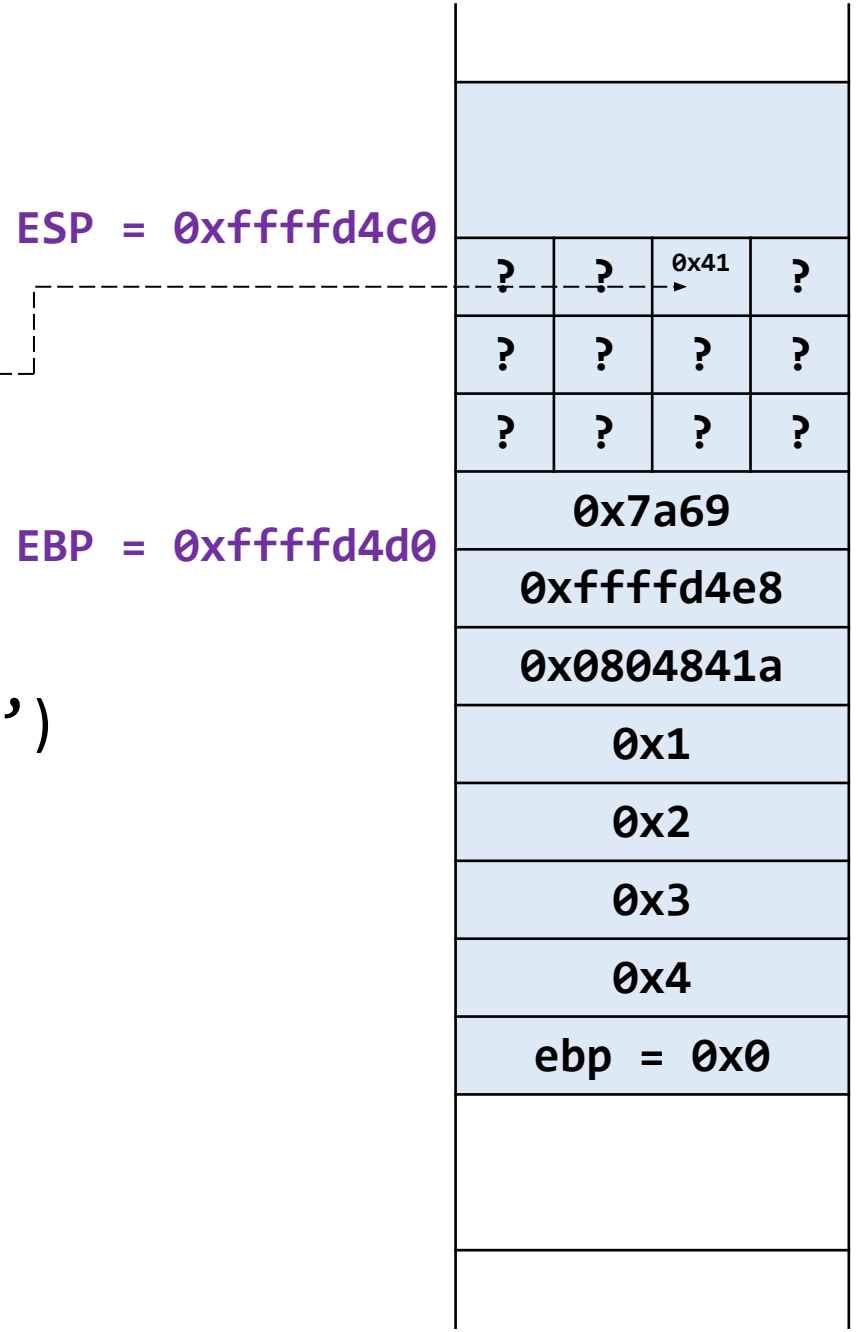
- Se crea un *stack frame* de 16 bytes (**0x10**)
  - Se restan 16 bytes al valor del **ESP**, el resultado se almacena en **ESP**
  - **ESP - 0x10 = 0xffffd4c0**

```
0x080483dd <+0>:     push   ebp
0x080483de <+1>:     mov    ebp,esp
0x080483e0 <+3>:     sub    esp,0x10
0x080483e3 <+6>:     mov    DWORD PTR [ebp-0x4],0x7a69
0x080483ea <+13>:    mov    BYTE PTR [ebp-0xe],0x41
0x080483ee <+17>:    leave
0x080483ef <+18>:    ret
```



- Almacena en **EBP-0x4** el valor de la variable **flag**.
  - 0x7a69 = 31337 (decimal)
  - La dirección de memoria de la variable **flag** es **EBP-0x4**

```
0x080483dd <+0>:     push   ebp
0x080483de <+1>:     mov    ebp,esp
0x080483e0 <+3>:     sub    esp,0x10
0x080483e3 <+6>:     mov    DWORD PTR [ebp-0x4],0x7a69
0x080483ea <+13>:    mov    BYTE PTR [ebp-0xe],0x41
0x080483ee <+17>:    leave
0x080483ef <+18>:    ret
```

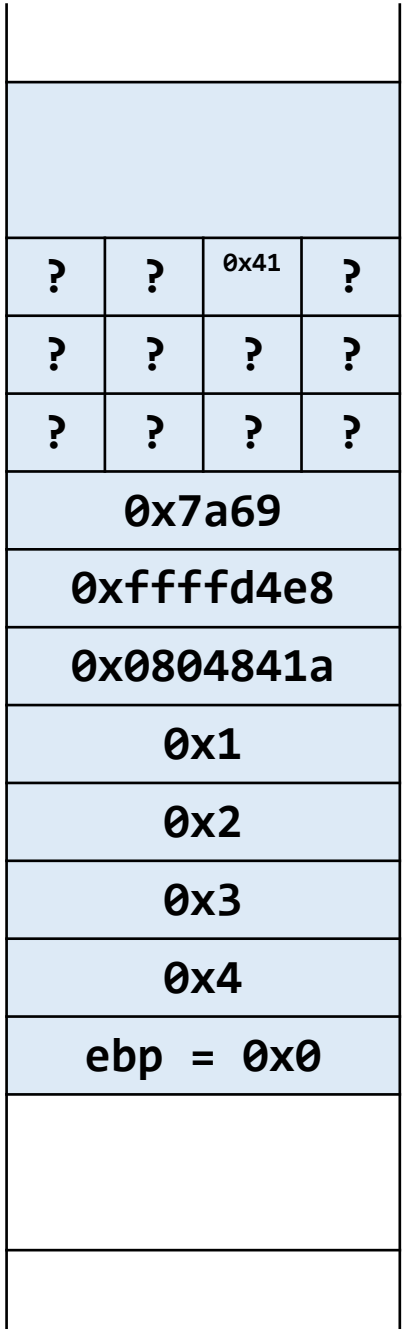


- Almacena en **EBP-0xe** (14) el valor 0x41 (ASCII 65 = ‘A’)
  - **EBP-0xe = 0xffffd4c2**
- El arreglo **buffer** empieza en esa dirección de memoria
  - **buffer[0] = 'A'**
- Sobran dos bytes al principio
  - Stack frame de 16 bytes
  - Arreglo de 10 bytes
  - Número entero **flag** de 4 bytes

```
0x080483dd <+0>:     push   ebp
0x080483de <+1>:     mov    ebp,esp
0x080483e0 <+3>:     sub    esp,0x10
0x080483e3 <+6>:     mov    DWORD PTR [ebp-0x4],0x7a69
0x080483ea <+13>:    mov    BYTE PTR [ebp-0xe],0x41
0x080483ee <+17>:    leave
0x080483ef <+18>:    ret
```

ESP = 0xffffd4c0

EBP = 0xffffd4d0



Description

Releases the stack frame set up by an earlier ENTER instruction. The LEAVE instruction copies the frame pointer (in the EBP register) into the stack pointer register (ESP), which releases the stack space allocated to the stack frame. The old frame pointer (the frame pointer for the calling procedure that was saved by the ENTER instruction) is then popped from the stack into the EBP register, restoring the calling procedure's stack frame.

A RET instruction is commonly executed following a LEAVE instruction to return program control to the calling procedure.

LEAVE—High Level Procedure Exit

Opcode	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
C9	LEAVE	ZO	Valid	Valid	Set SP to BP, then pop BP.
C9	LEAVE	ZO	N.E.	Valid	Set ESP to EBP, then pop EBP.
C9	LEAVE	ZO	Valid	N.E.	Set RSP to RBP, then pop RBP.

Intel® 64 and IA-32 Architectures Software Developer's Manual

Combined Volumes: 1, 2A, 2B, 2C, 2D, 3A, 3B, 3C, 3D and 4

Prólogo y epílogo de las funciones

- Prólogo

```
push ebp
mov ebp,esp
sub esp, <N bytes>
```

- Epílogo

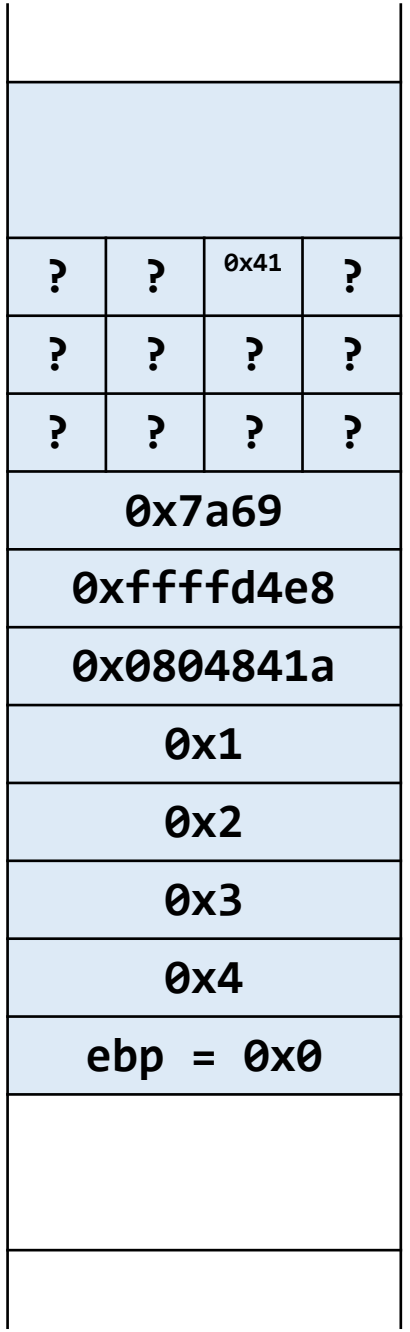
```
mov esp,ebp
pop ebp
ret
```

`leave`

```
0x080483dd <+0>:     push   ebp
0x080483de <+1>:     mov    ebp,esp
0x080483e0 <+3>:     sub    esp,0x10
0x080483e3 <+6>:     mov    DWORD PTR [ebp-0x4],0x7a69
0x080483ea <+13>:    mov    BYTE PTR [ebp-0xe],0x41
0x080483ee <+17>:    leave
0x080483ef <+18>:    ret
```

```
mov esp,ebp
pop ebp
ret
```

ESP = EBP = 0xffffd4d0



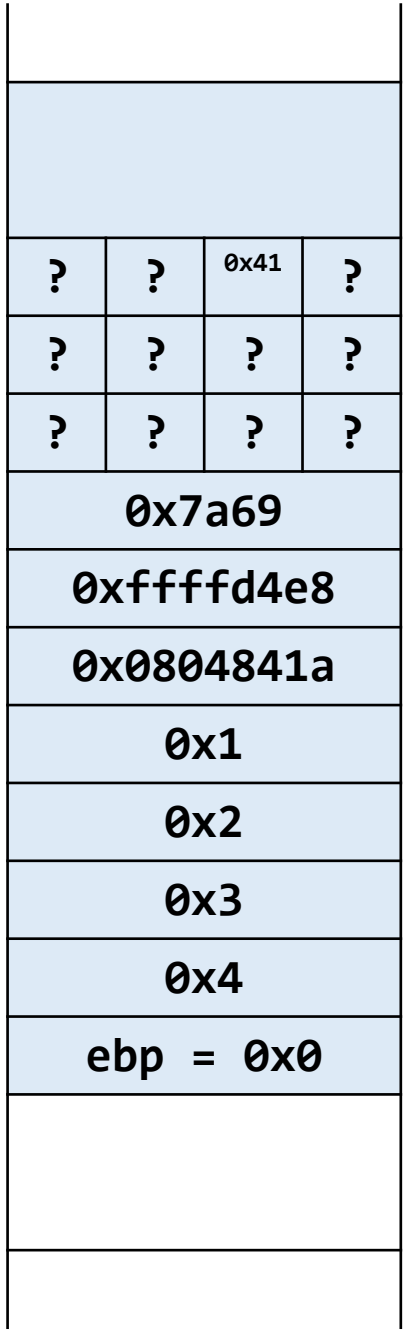
- Se destruye el *stack frame* que se había creado para la función **`test_function()`**.
- No se eliminan los datos de memoria.

```
0x080483dd <+0>:     push   ebp
0x080483de <+1>:     mov    ebp,esp
0x080483e0 <+3>:     sub    esp,0x10
0x080483e3 <+6>:     mov    DWORD PTR [ebp-0x4],0x7a69
0x080483ea <+13>:    mov    BYTE PTR [ebp-0xe],0x41
0x080483ee <+17>:    leave
0x080483ef <+18>:    ret
```

```
mov esp,ebp
pop ebp
ret
```

ESP = 0xffffd4d4

EBP = 0xffffd4e8



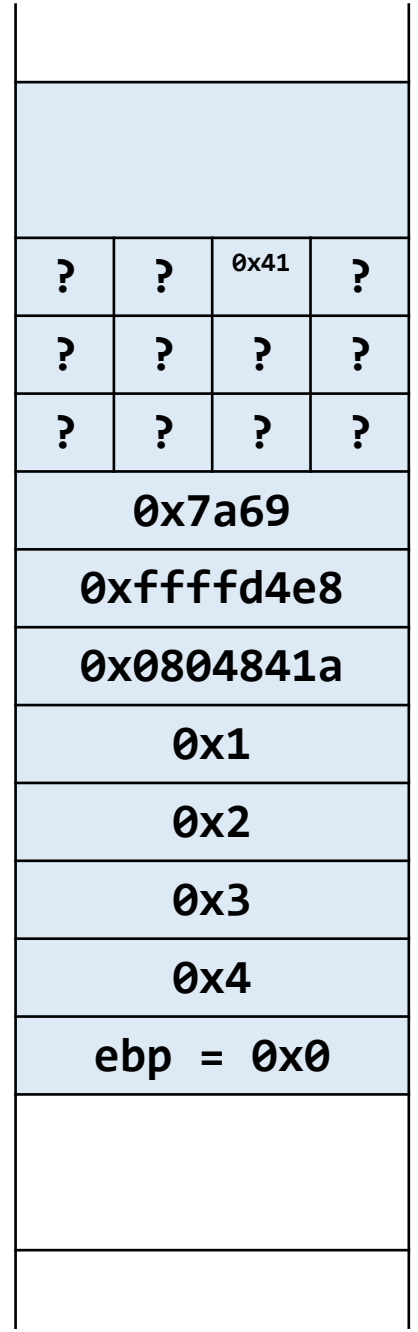
- La instrucción **pop ebp**
  - Saca de la pila lo apuntado por **ESP**
  - Lo que sale de la pila se asigna al registro **EBP**
  - Actualiza **ESP**: **ESP + 4 = 0xffffd4d0 + 4 = 0xffffd4d4**
  - Esta operación restaura el valor del **EBP** anterior al llamado de la función **test_function()**.

```
0x080483dd <+0>:     push   ebp
0x080483de <+1>:     mov    ebp,esp
0x080483e0 <+3>:     sub    esp,0x10
0x080483e3 <+6>:     mov    DWORD PTR [ebp-0x4],0x7a69
0x080483ea <+13>:    mov    BYTE PTR [ebp-0xe],0x41
0x080483ee <+17>:    leave
0x080483ef <+18>:    ret
```

```
mov esp,ebp
pop ebp
ret
```

ESP = 0xffffd4d8

EBP = 0xffffd4e8



- La instrucción **ret**
  - Saca de la pila la dirección apuntada por el **ESP**
  - Lo que sale de la pila se asigna al registro **EIP**
    - **EIP = 0x0804841a**
  - Actualiza ESP: **ESP + 4**
    - **ESP + 4 = 0xffffd4d8**

- En este punto la ejecución sigue en la dirección apuntada por EIP
- Instrucción posterior al CALL

```
...
0x08048415 <+37>:   call 0x80483dd <test_function>
0x0804841a <+42>:   leave
0x0804841b <+43>:   ret
```

- Se recuperó el *stack frame* de la función **`main()`** ya que la ejecución retorna a la función **`main()`** a la instrucción posterior al llamado a la instrucción **`test_function()`**.