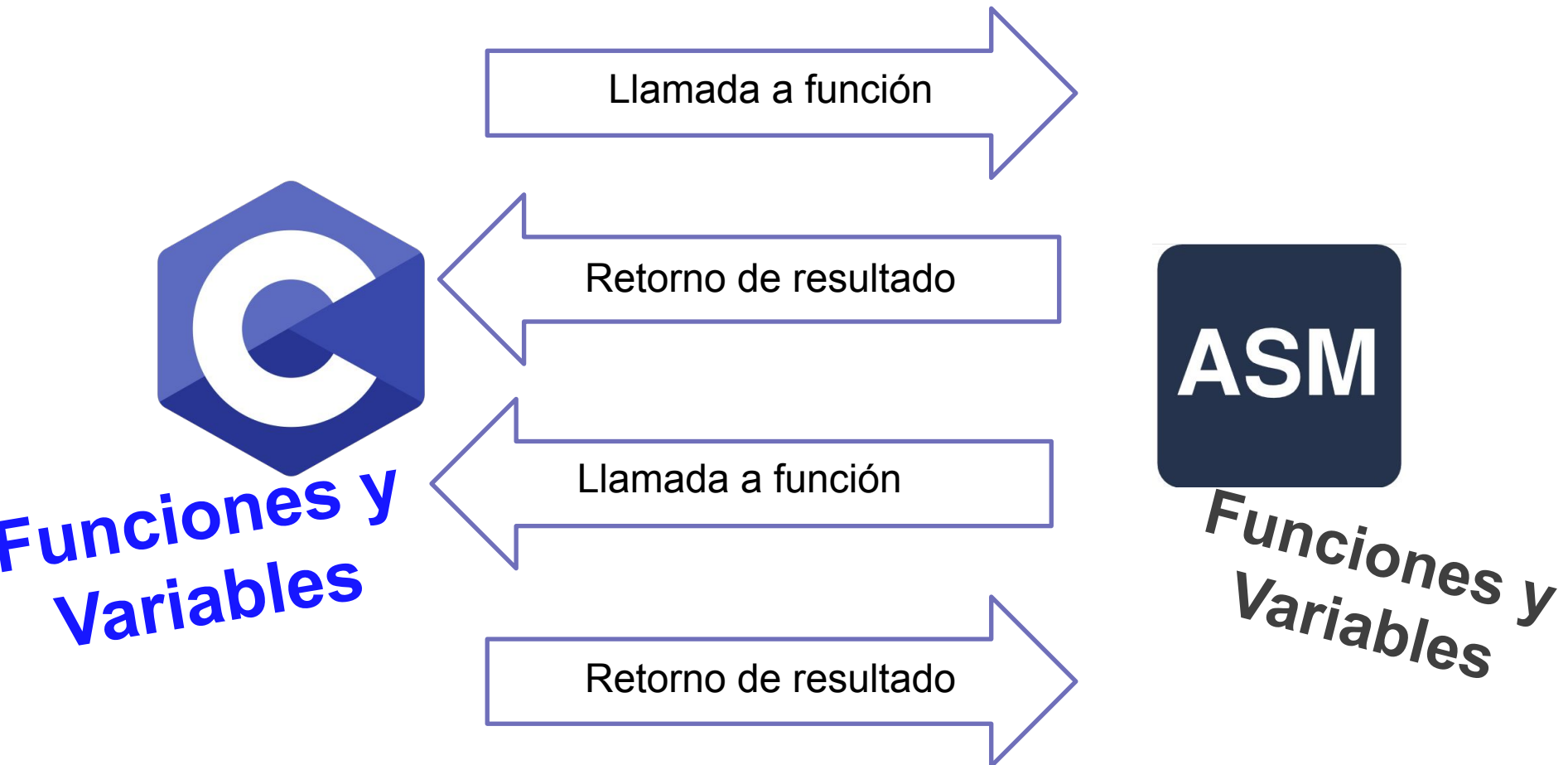




ASM y C

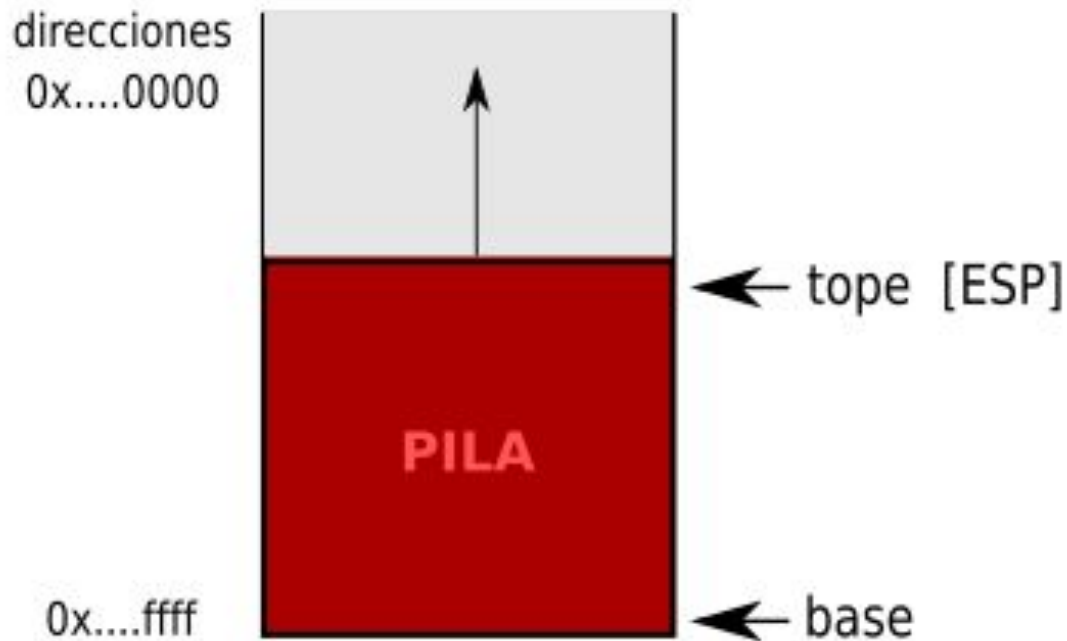
# Mezcla de lenguajes en un ejecutable



**¿Como se soluciona?**

**Con la pila y los registros**

# Repaso de Pila

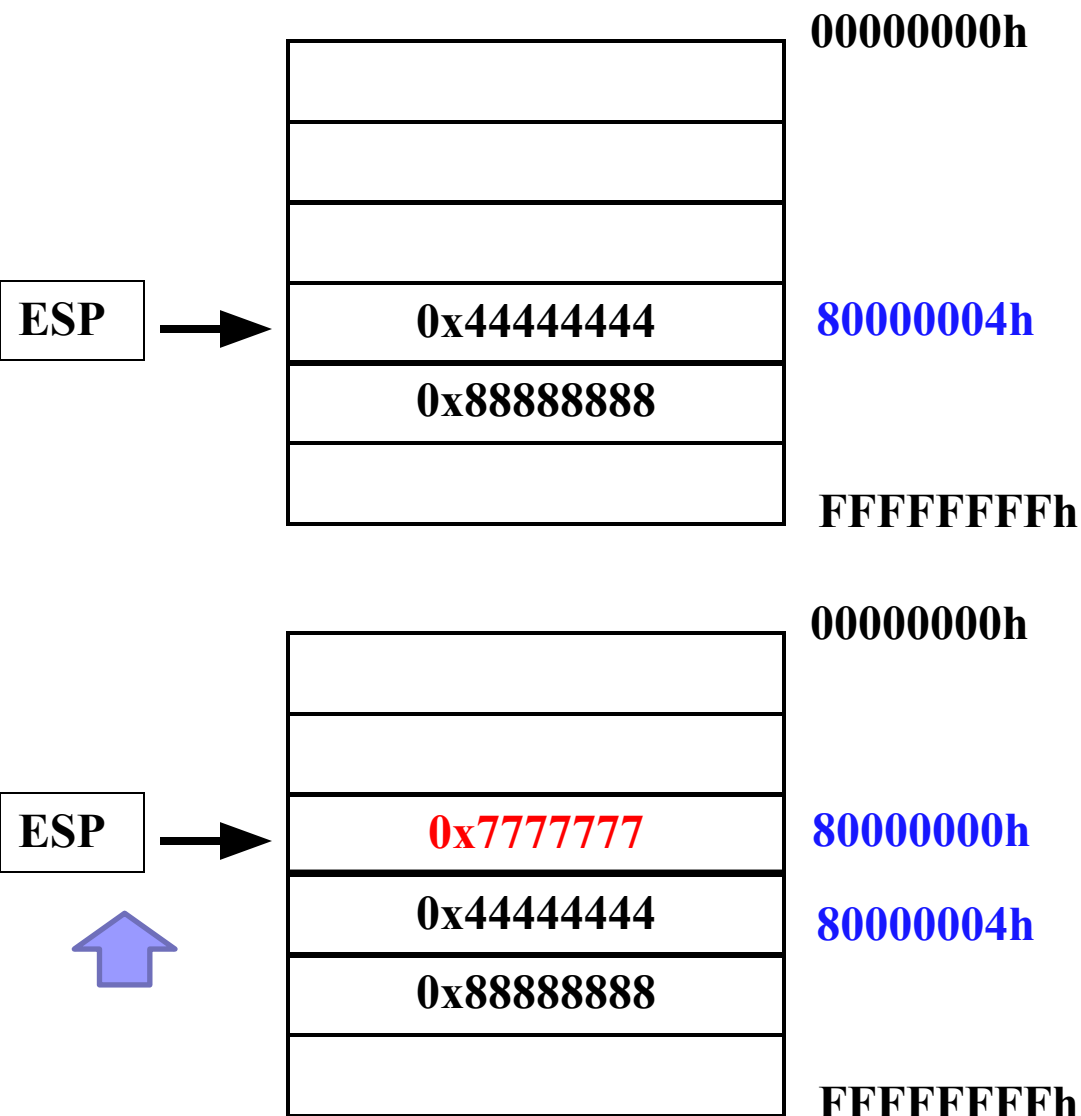


El *stack pointer register* o *extended stack pointer*) apunta al tope de la pila, es decir al último elemento almacenado en ella.

Cuando se almacena un nuevo valor en la pila con **PUSH** el valor del puntero se actualiza para siempre apuntar al tope de la pila.\*

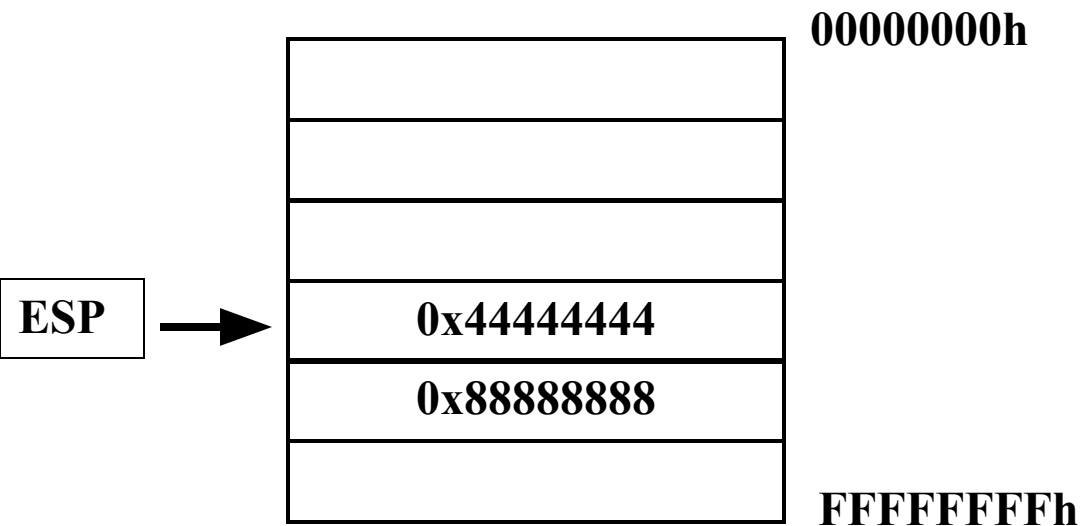
\*<https://fundacion-sadosky.github.io/guia-escritura-exploits/buffer-overflow/1-introduccion.html>

# Repaso de Pila - Push

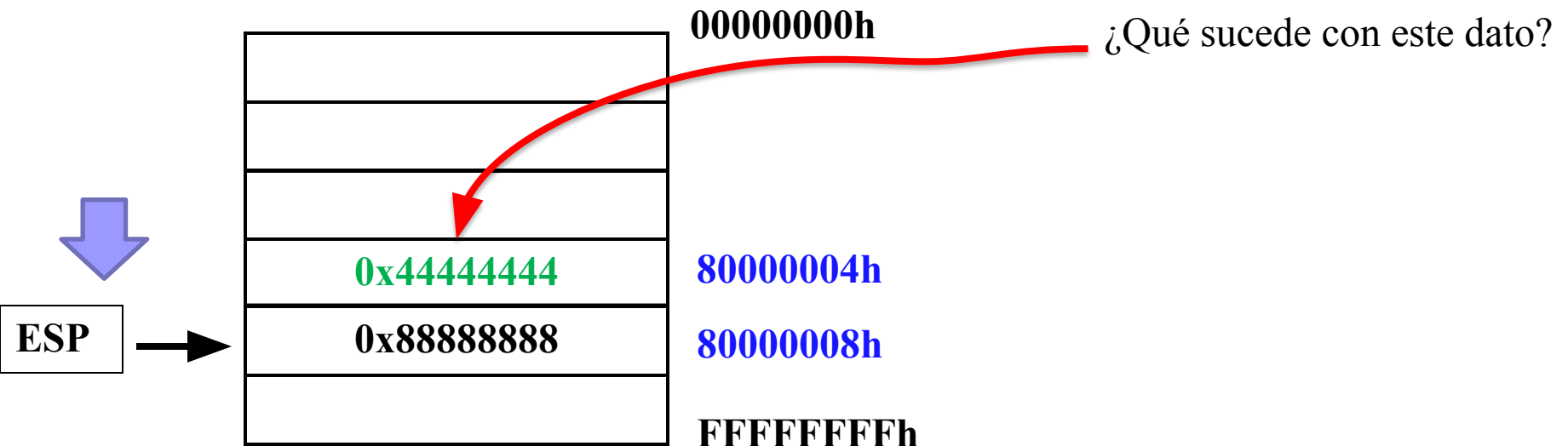


Cuando se ejecuta una instrucción **PUSH**, el procesador decrementa el registro ESP ó RSP y guarda el valor en el stack

# Repaso de Pila - Pop



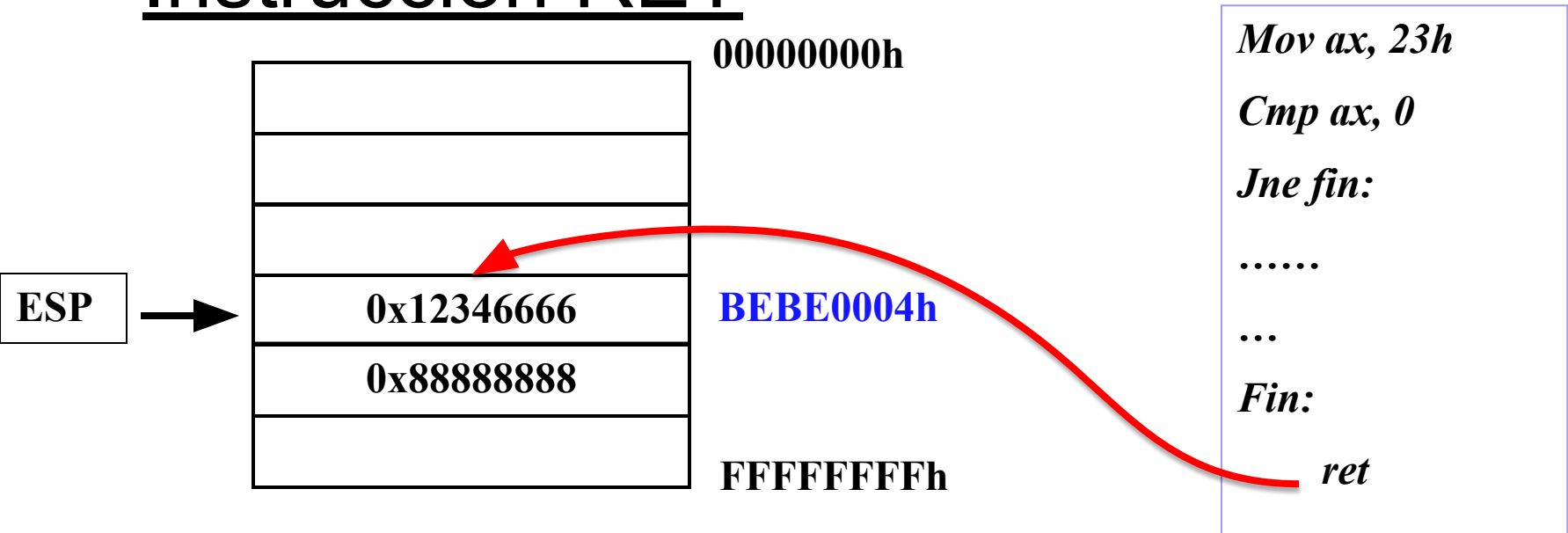
Cuando se ejecuta una instrucción POP, toma el contenido de la dirección ( en este caso el contenido es 0x44444444 ) y luego incrementa el registro ESP ó RSP.





# Instrucción RET

# Instrucción RET



Cuando se ejecuta una instrucción **RET**, el procesador toma el contenido de los apuntado por ESP y salta a esa posición de memoria

En este caso salta a la dirección de memoria 12346666h.

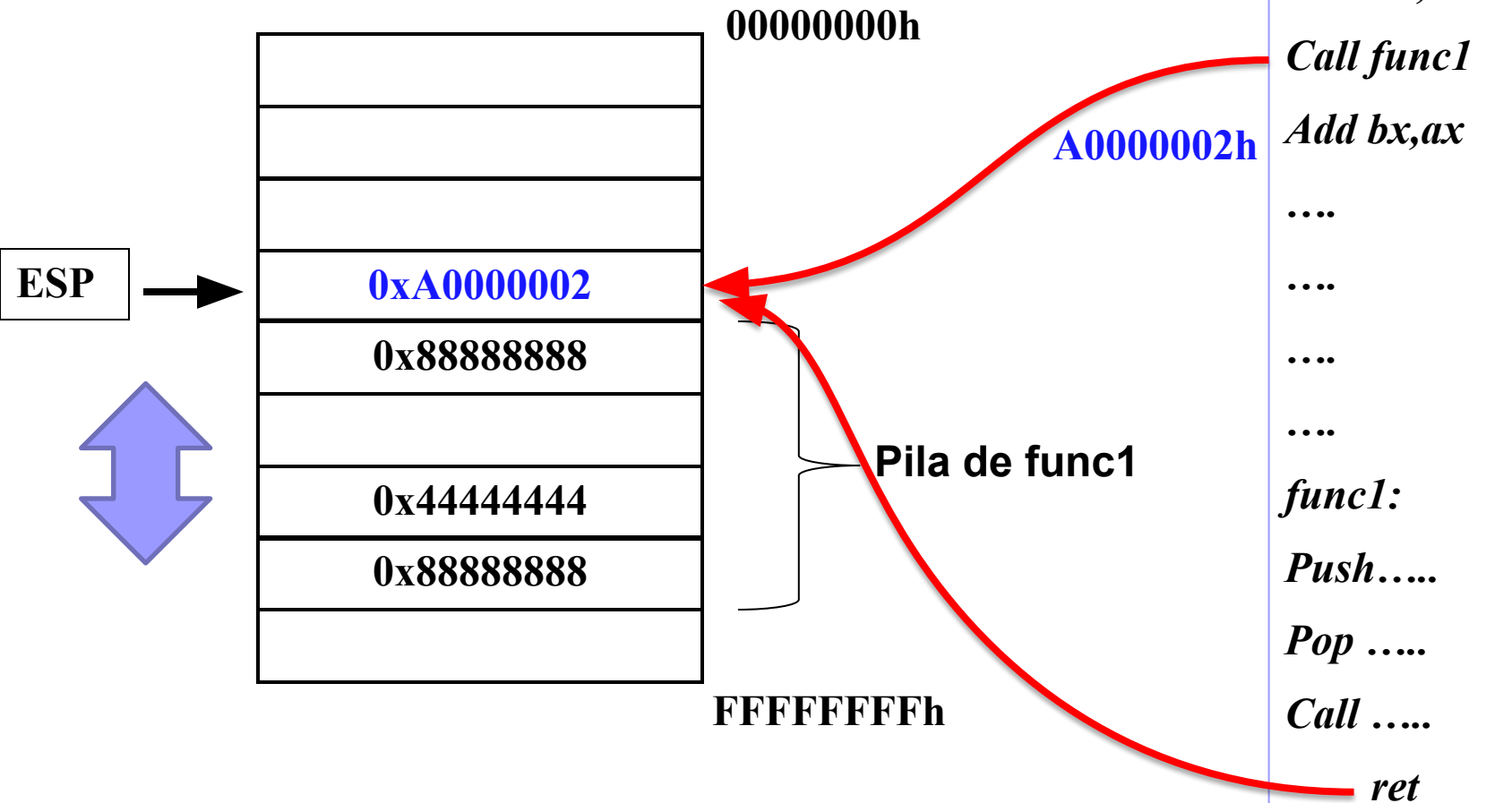
Es equivalente a hacer un JMP a esa dirección.



# Instrucción CALL



# Instrucción RET



La instrucción CALL guarda en la pila la próxima instrucción a ejecutarse ó también llamada dirección de retorno. La función llamada (func1) tiene que dejar la pila sin modificar antes de terminar, así RET puede volver correctamente.

# Análisis de C



Entendamos como funciona el compilador de C para poder mezclarlo con ASM

## Pasaje de parámetros en funciones

```
int funcion_en_C ( int var1, char var2, int *var3 );
```

- Por Valor
- Por Referencia

# Pasaje de argumentos en C

- Según la arquitectura el compilador pasa de manera diferentes **los argumentos de las funciones**
- Arquitectura de **32 bits**
  - Se pasan por la **pila**
- Arquitectura de **64 bits**
  - Se pasan primero por **registros** y luego por la **pila**

# Pasaje de argumentos por registros

- Para pasar argumentos se usan los registros RDI, RSI, RDX, RCX, R8, R9.
- Si la función necesita más parámetros se usa la pila
- Para punto flotante (float, double), xmm0, xmm1, xmm2, xmm3, xmm4, xmm5, xmm6, xmm7

# Pasaje de argumentos por registros

**Registros a preservar:**

**%rbp**

**%rsp**

**%rbx**

**%r12**

**%r13**

**%r15**

Estos registros pertenecen a la función llamadora y deben mantener su valor al terminar la función

# Pasaje de argumentos por registros

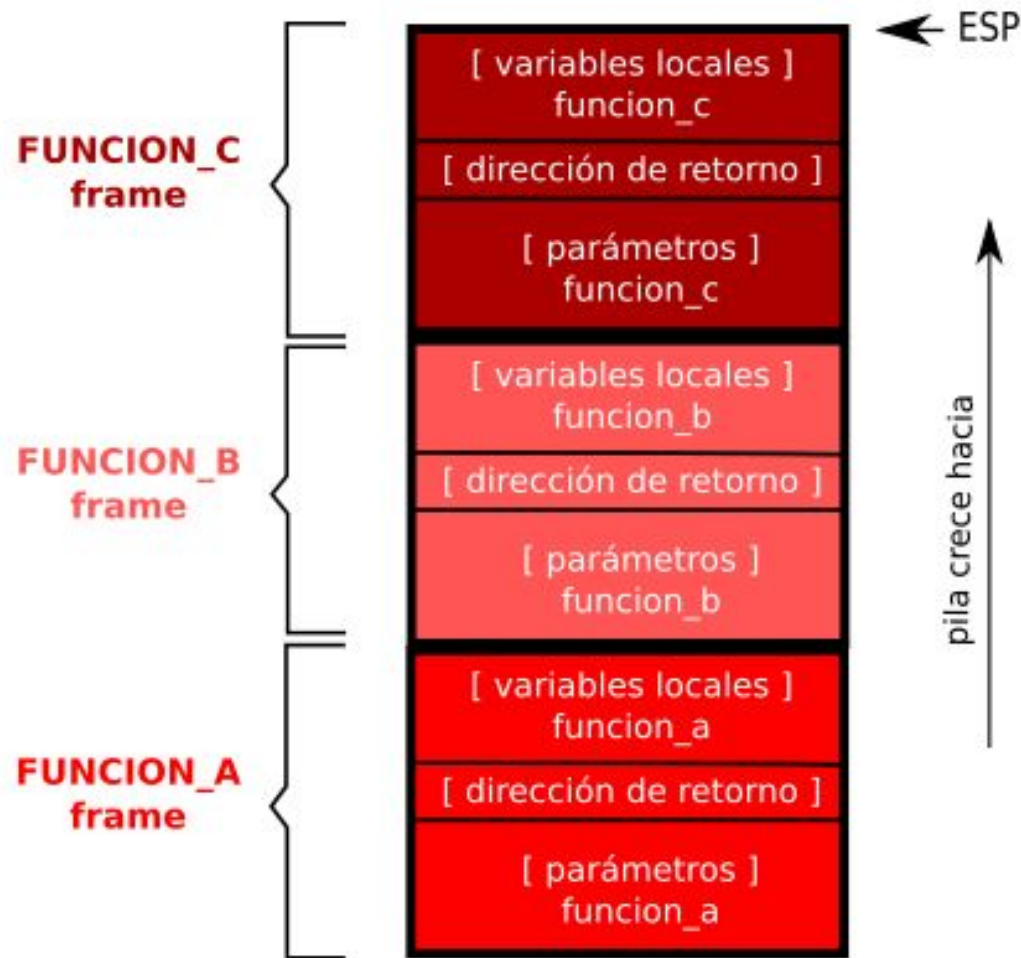
- Según el tipo de dato que se quiere pasar se usan diferentes registros.\*
- Si los argumentos no entran en los registros se usa la pila pasando de derecha a izquierda.

**\*Igual a lo que vimos en las llamadas a sistemas con INT 80 ó SYSCALL**

# Manejo de la pila en C

- La pila se utiliza para pasar parámetros entre funciones.
- Al llamar a una función, ésta utiliza la pila para sus propias instrucciones PUSH y POP, por lo tanto modifica el registro ESP.
- Se utiliza el registro EBP para acceder a los parámetros que puede haber en la pila o a las variables locales, de esta manera no se modifica el registro ESP.

# Llamados a funciones con pila



- En la arquitectura x86, en el llamado a funciones la pila juega un rol fundamental.
- En este espacio de memoria se almacenan las variables locales de la función llamada, sus argumentos y su dirección de retorno.
- Aparece el concepto de **frame**



# Convenciones en C

Parámetros de una función:

*Por ejemplo:*

***funcion\_en\_C ( param\_a, param\_b, param\_c );***

Los parámetros se pushean en el stack de derecha a izquierda.

En este caso primero el **param\_c** luego el **param\_b** y luego el **param\_a**.

Llamada a la función:

Se ejecuta la instrucción CALL que guarda en el stack el EIP para el retorno y ejecuta un JMP al primer byte de la función.

# Convenciones en C

## Resguardo y actualización de EBP (armado de stack frame)

Una vez en la nueva función se resguarda el valor anterior de EBP. Y luego se le asigna el valor actual del registro ESP. De esta manera se puede utilizar el registro EBP para acceder a los parámetros que quedaron en la pila.

Armado de  
stack  
frame

**Push ebp**  
**Mov ebp,esp**

Desarmado  
de stack  
frame

**Mov esp, ebp**  
**Pop ebp**

## Valores a retornar

Si el valor es menor a 32 bits se retorna en EAX.

- Si es mayor retorna la parte alta en EDX y la parte baja en EAX.
- Si es un dato más complejo ( ej. Estructura de datos ) retorna un puntero formado por EDX:EAX

# Llamada de ASM a C

*; Puts.asm*

*; Programa que imprime utilizando puts*

*global main*

*extern puts*

*section .data*

*mensaje db 'Utilizando puts', 0Ah, 0*

*section .text*

*main:*

*push ebp*

*mov ebp,esp ; genera stack frame*

*push dword mensaje ; parametro para puts*

*call puts ; llamada*

*pop eax ; saca el parámetro de la pila*

*mov esp,ebp*

*pop ebp ; destruye stack frame*

*ret*

# Llamada de C a ASM

```
// cyasm1.c
#include <stdio.h>
extern unsigned int siete( void );
int main(void)
{
    printf("Devuelve el numero siete = %d\n", siete() );
    return 0;
}
```

```
; cyasm1_1.asm
[GLOBAL siete]
[SECTION .text]
siete:
    push    ebp
    mov     ebp,esp
    mov     eax,7
    mov     esp,ebp
    pop     ebp
    ret
```

# Inline Assembler

```
int main(void)
{
    __asm__ ("movl $0x12345678, %eax");
}
```

```
int main (void )
{
    __asm__ ( "movl %eax, %ebx\n\t"
             "movl $56, %esi\n\t"
             "movb %ah, (%ebx)");
}
```

**No usarlo nunca en esta materia**



# Salidas en ASM

- Veamos si se cumplen las convenciones de C analizando lo que genera el compilador.
- Tenemos dos formas de ver el código C convertido en ASM
  - Compilar con “-S”
  - Utilizar GDB

# Ejemplo de Salida en ASM

Dado el siguiente programa en C, generamos su salida en ASM

```
/* asmyc1.c */  
#include <stdio.h>  
#include <stdlib.h>  
int main ()  
{ int numero=10;  
  printf(“Numero vale = %d”, numero );  
  exit(0);  
}
```

```
gcc -S -masm=intel asmyc1.c
```

# Ejemplo de Salida en ASM (32 bits)

Veamos que se creó en ASM por cada línea de C.

```
.file "asmyc1.c"
.section .rodata
.LC0:
.string "Numero vale = %d"
.text
.align 2
.globl main
.type main,@function
main:
    push    ebp
    mov     ebp,esp
    sub     esp,8
    and     esp,-16
```

```
    mov     eax,0
    sub     esp,eax
    mov     [ebp-4], 10
    sub     esp,8
    push    [ebp-4]
    push    .LC0
    call    printf
    add     esp,16
    sub     esp,12
    push    0
    call    exit
.Lfe1:
.size     main,.Lfe1-main
.ident    "GCC: (GNU) 3.2 20020903 "
```