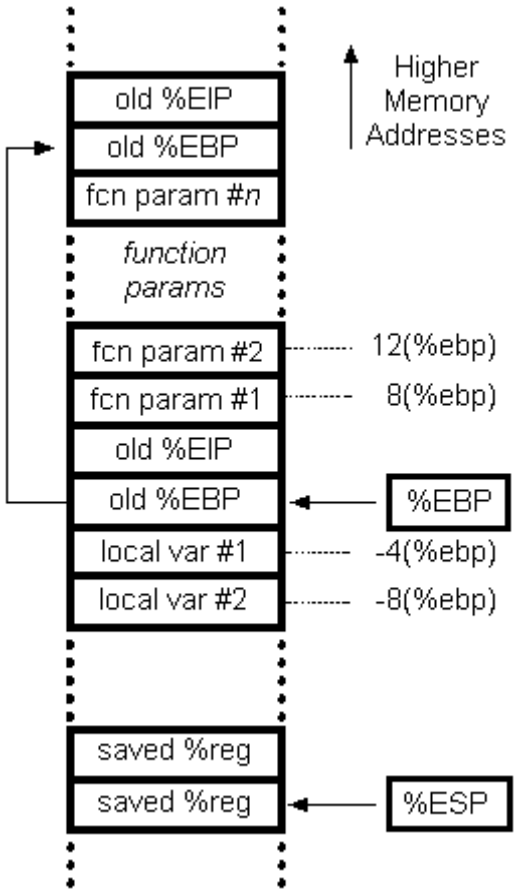
EDUT01A3 - Compilación en C (observando el código generado en ensamblador)

En este ejercicio se trata de analizar el código ensamblador (que tiene una correspondencia directa con código máquina), resultado de la traducción dos programas simples en C.

En el ejercicio, se usa lo que se conoce como “pila” para ir guardando las variables. En el ejemplo **%ebp** es equivalente a **%rpb**

Este registro es un puntero que se utiliza para referenciar a los parámetros y variables locales de una función.



http://maxkalavera.blogspot.com.es/2012/05/manejando-la-pila-de-memoria-en.html

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
int num1;
int num2;
int resultado;

num1=3;
```

EDUT01A3 - Compilación en C (observando el código generado en ensamblador)

```
num2=7;

resultado=num1+num2;

return 0;
}
```

gcc –S suma2.c // el programa gcc genera un archive llamado suma2.s

```
        .file   "suma2.c"
        .text
        .globl  main
        .type   main, @function
main:
.LFB2:
        .cfi_startproc
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        movl    $3, -12(%rbp)
        movl    $7, -8(%rbp)
        movl    -8(%rbp), %eax
        movl    -12(%rbp), %edx
        addl    %edx, %eax
        movl    %eax, -4(%rbp)
        movl    $0, %eax
        popq    %rbp
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
```

EDUT01A3 - Compilación en C (observando el código generado en ensamblador)

Repite el ejercicio con el siguiente código:

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
int num1;
int num2;
int num3;
int resultado;

num1=3;
num2=7;
num3=2;

resultado=num1+num2;
resultado=resultado-num3;

return 0;
}
```

`gcc –S suma2.c //el programa gcc genera un archive llamado suma2.s`

```
                .file   "suma2.c"
        .text
        .globl  main
        .type   main, @function
main:
.LFB2:
        .cfi_startproc
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        movl    $3, -16(%rbp)
        movl    $7, -12(%rbp)
        movl    $2, -8(%rbp)
        movl    -12(%rbp), %eax
        movl    -16(%rbp), %edx
        addl    %edx, %eax
        movl    %eax, -4(%rbp)
        movl    -8(%rbp), %eax
        subl    %eax, -4(%rbp)
        movl    $0, %eax
        popq    %rbp
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE2:
        .size   main, .-main
        .ident  "GCC: (Ubuntu 4.8.2-19ubuntu1) 4.8.2"
        .section        .note.GNU-stack,"",@progbits
```

A la vista de los dos ejemplos,

1. ¿Qué operación realizan las instrucciones marcadas en negrita?

```
        movl    $2, -8(%rbp)
```
Crea un espacio en memoria para almacenar 8 bits

```
        addl    %edx, %eax
```
Asignamos un valor al espacio creado anteriormente

```
        subl    %eax, -4(%rbp)
```
Resta a eax el valor escrito anteriormente.

2. Subraya la parte del código en ensamblador que se corresponde con la instrucción en c: `resultado=resultado-num3;`

```
        subl    %eax, -4(%rbp)
```

3. ¿Cuándo se utilizan los registros %eax y %edx? ¿En qué lugar de la arquitectura Von Neumann se encontrarán?

   En el registro de direcciones y registro de direcciones

4. ¿Qué quiere decir que una pila es de tipo FIFO?
   Que las intrucciones entran y una vez ejecutadas salen