**PRÁCTICA 5: LLAMADAS A UNA SUBRUTINA:**

**ÍNDICE**

**Módulos Fuente Comentados ........................................................................................ 3**

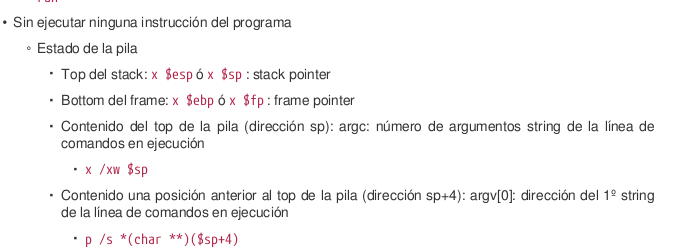
**Módulo sumMtoN\_aviso.c ................................................................................................ 10**

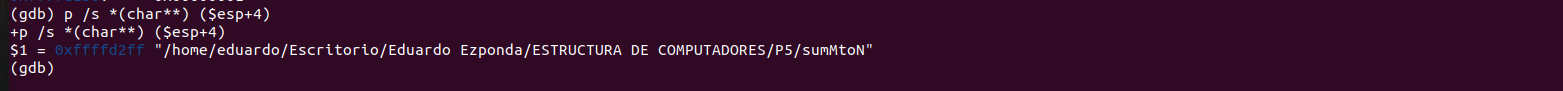
**Módulo sumMtoN\_aviso.c ................................................................................................ 11**

**Comandos de Compilación ............................................................................................. 13**

**Conclusión ...........................................................................................................13**

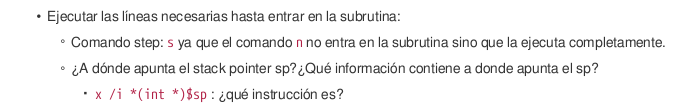
A continuación vamos a realizar una serie de comandos en el gdb para ver cuáles son los contenidos de la pila:

****

****

La única diferencia entre el primer y el tercer comando es que se pide que se imprima en formato hexadecimal y en tamaño word (2 bytes).

Además, al no realizar ninguna instrucción del programa ni ningún push, se estará accediendo a una pila de la computadora distinta a la del programa.

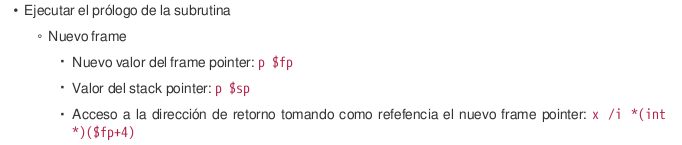


****

Como se puede observar, realizamos un examine en formato instrucción del stack pointer sp.

La instrucción resultante es mov %eax, %ebx.

A través del comando step (s), se accede directamente a la subrutina.

****

****

Tras ejecutar el prólogo de la subrutina, en el que apilamos la antigua dirección del frame pointer para tenerla guardada al final de la subrutina y así por mantener la pila de la misma forma que al principio. Además hacemos que apunte el frame pointer al stack pointer para “crear” una nueva pila encima de la anterior.

Realmente no se crea una nueva pila porque únicamente existe una.

En las imágenes se puede observar el resultado de los dos prints y el examine.

****

Tras realizar toda la subrutina sumMtoN, se devuelve el resultado de la función a través del registro eax. Con lo cual, en este caso, siendo la M 5 y la N 10, el resultado de la operación será 5 + 6 + 7 + 8 + 9 + 10 = 45.



****

Tras ejecutar el epílogo de la subrutina, dejaremos el stack de la misma manera que antes de la ejecución de la llamada a la subrutina.

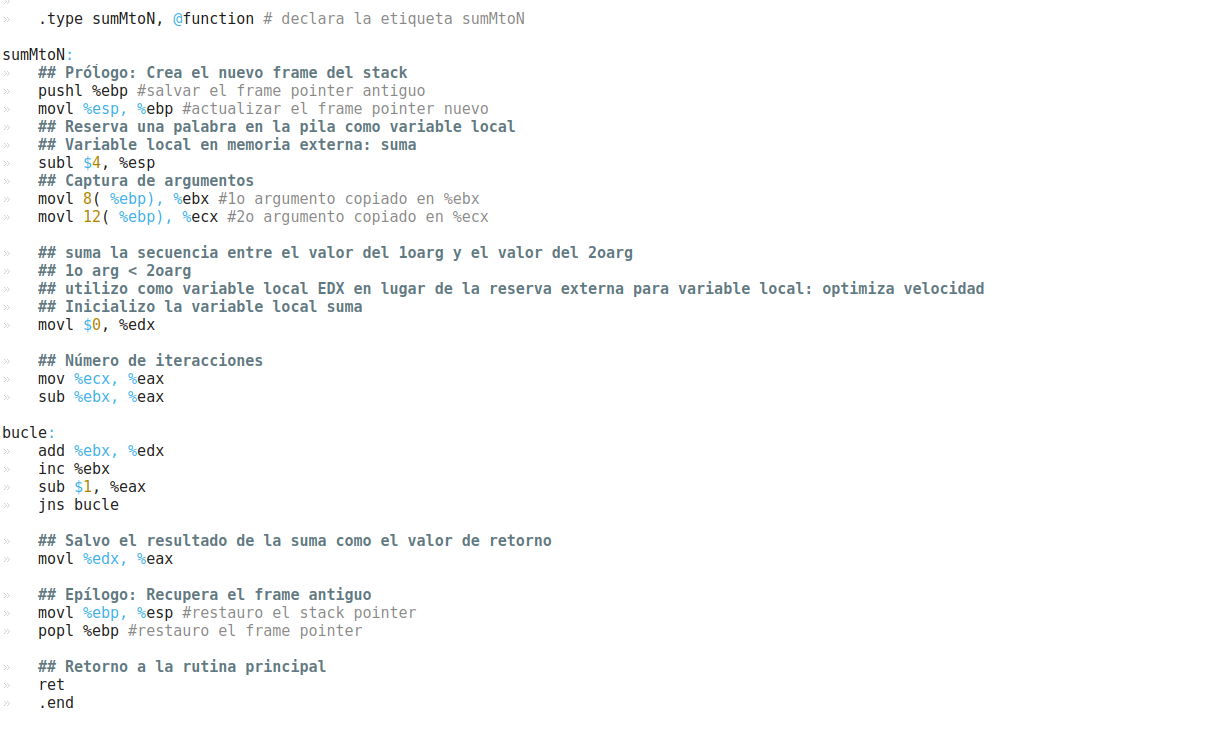
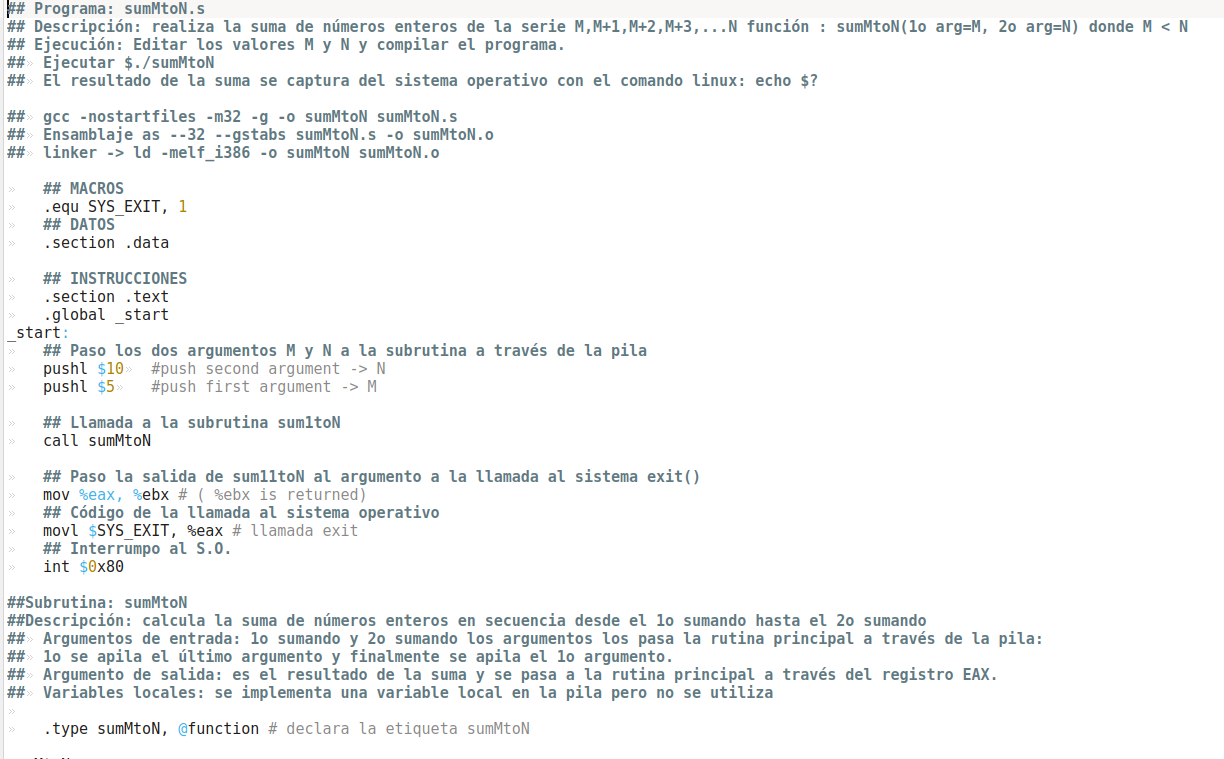
Como se puede observar en la imagen, los resultados de los comandos se encuentran justo debajo.



****

La dirección del stack pointer varía porque vuelve a la rutina inicial, y por lo tanto apuntará a donde estaba inicialmente.

**PROGRAMA: sumMtoN.s:**

****

FUNCIONAMIENTO PROGRAMA:

En primer lugar, se declara la macro SYS\_EXIT como 1 para luego llamar posteriormente a exit con el registro eax.

Para llamar a la subrutina sumMtoN, previamente se necesita pasar los valores de M y N. Para ello, se utiliza una pila y por ello se apilan los valores 5 y 10. La pila es una estructura LIFO, con lo que se apilarán los valores continuamente por encima del anterior y se desapilarán los valores introducidos por último lugar. Además, la función se declara como .type sumMtoN, @function.

Se llama a sumMtoN a través de call. A continuación comenzaría el prólogo debido a que al final de la subrutina se tiene que devolver la pila en el mismo estado en el que se encontraba previamente. El contenido de la pila tiene que ser el mismo, y el stack pointer (esp) y frame pointer (ebp) tienen que apuntar a la misma zona de memoria previa.

Por lo tanto, apilamos la dirección del antiguo frame pointer para tenerla guardada para el final y apuntamos el “nuevo” frame pointer a donde apuntaba el stack pointer para “crear” una nueva pila en la zona superior. Este proceso se denomina prólogo.

A partir de ello, capturamos los elementos con una dirección relativa a base. Observando el estado de la pila en la siguiente página, sabiendo que cada tamaño es de 4 bytes, únicamente sumando esos 4 bytes de donde apunta el esp accederemos al segundo y tercer argumento de la pila. Una vez capturados, nos dispondremos a través de la etiqueta bucle a realizar la suma en cada una de las iteraciones hasta que eax sea 0.

Cuando eax sea 0, significa que el bucle ha concluido y por tanto el resultado de la operación en este caso se guarda en el registro edx.

Como es una función, es necesario pasar el valor de retorno al registro eax y realizar el epílogo para dejar la pila de la misma manera previa.

Volvemos a la rutina principal a través de ret. Ya únicamente quedaría la parte final del programa en el que se devuelve al sistema operativo el resultado de la operación (45), y se llama despueś al sistema operativo para que tome el control e interrumpa el programa.

ESTADOS DE LA PILA/STACK :

esp ->10 y ebp ->10

|  |
| --- |
| 10 |



Esp -> 5 y ebp -> 10

|  |
| --- |
| 5 |
| 10 |



**EPÍLOGO:**

esp -> ebp y ebp -> ebp

|  |
| --- |
| ebp |
| 5 |
| 10 |

****

esp -> variable local en memoria externa y ebp -> ebp

|  |
| --- |
|  |
| ebp |
| 5 |
| 10 |

****

**PRÓLOGO:**

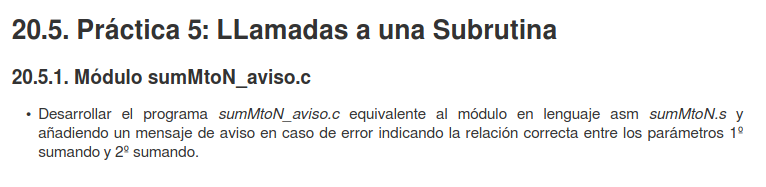
esp -> ebp y ebp -> ebp

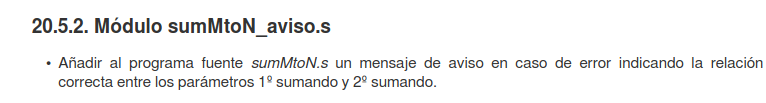
|  |
| --- |
| ebp |
| 5 |
| 10 |

esp -> 5 y ebp -> 10

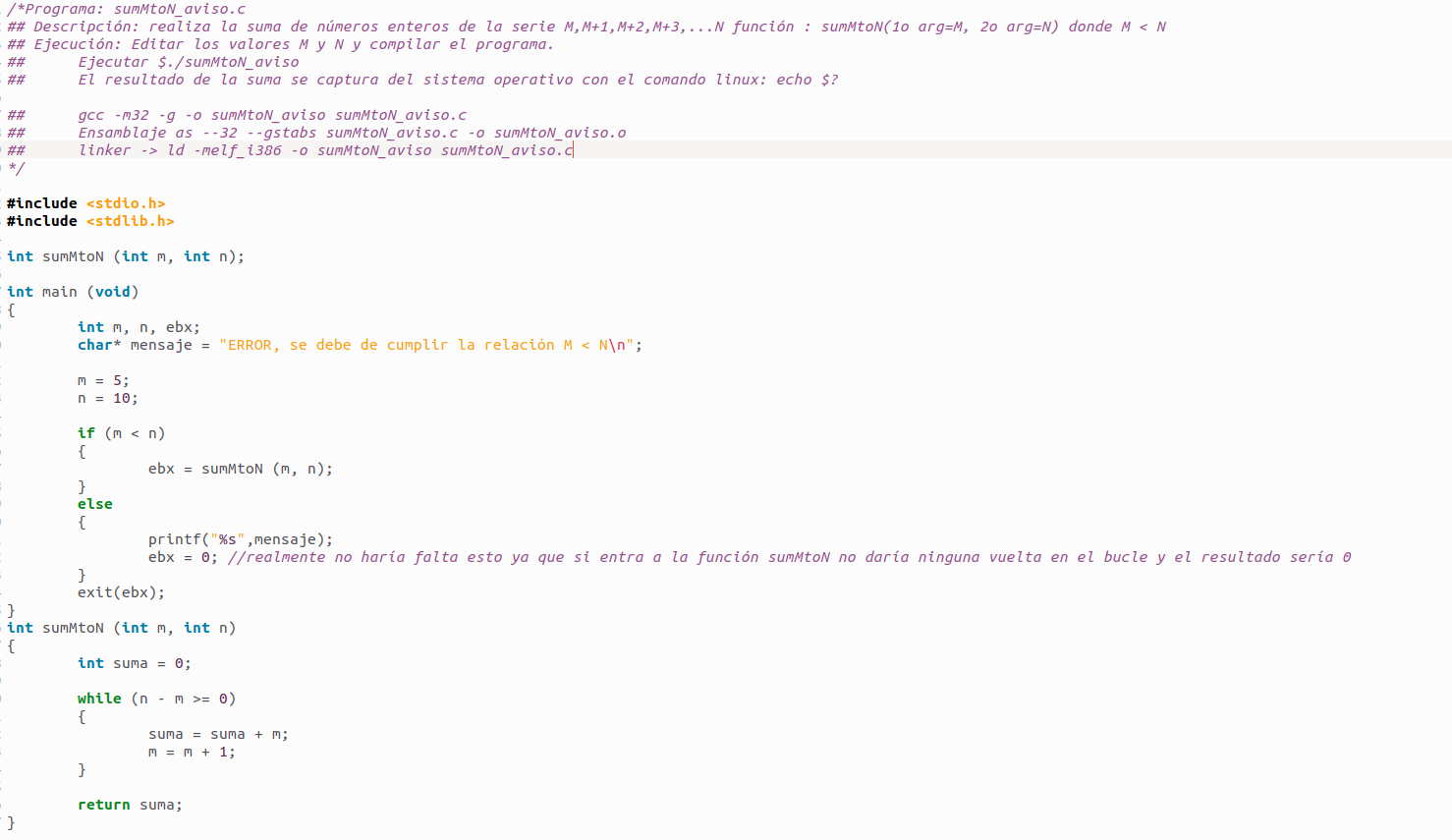
|  |
| --- |
| 5 |
| 10 |

****

****

****

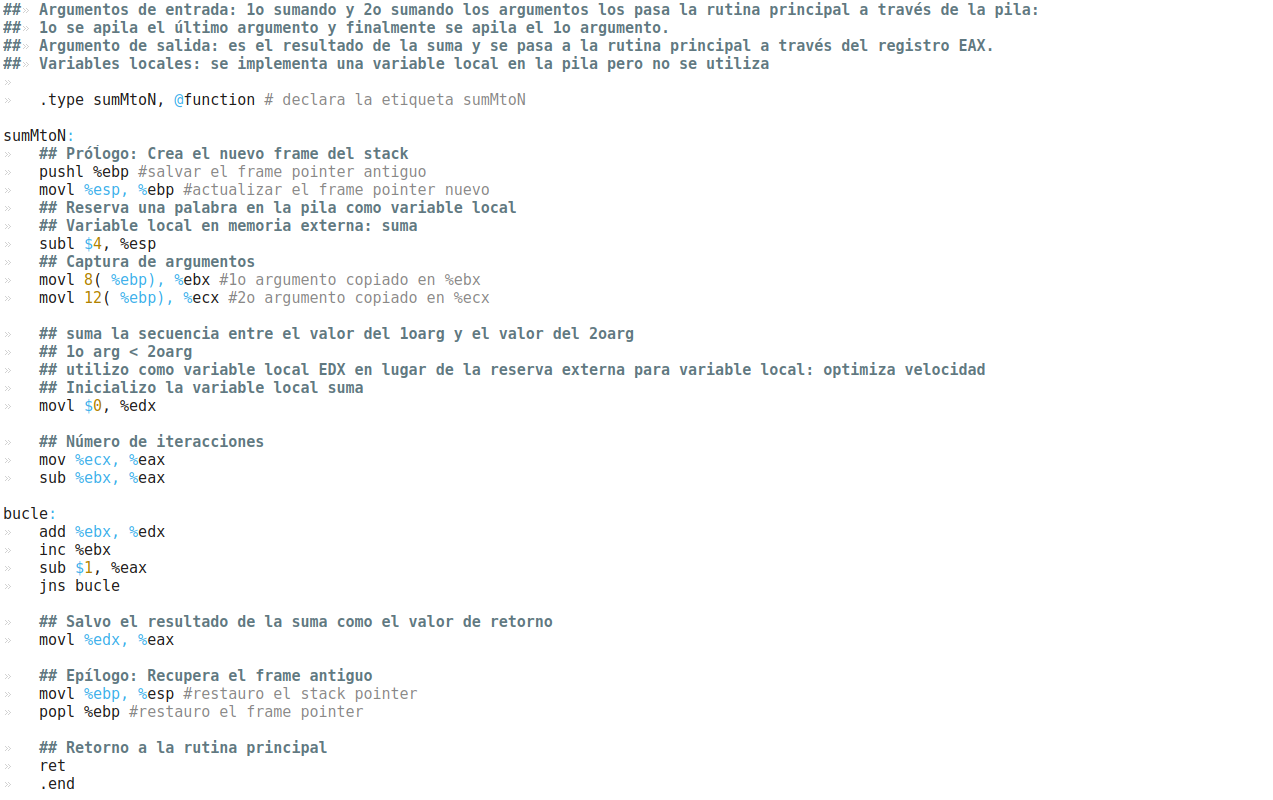
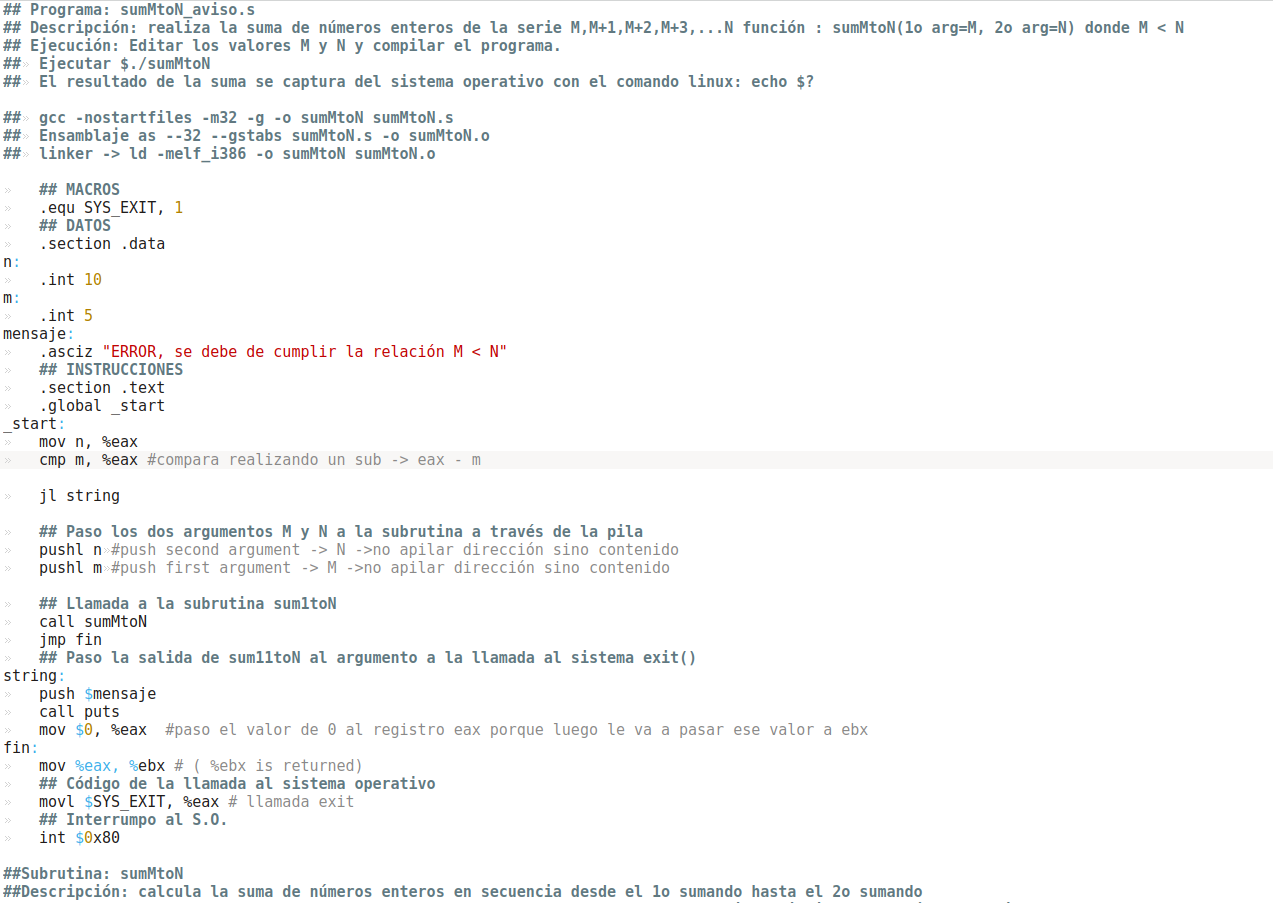
**SUMMTON\_AVISO.C:**

****

El funcionamiento del programa sumMtoN\_aviso.c es similar al sumMtoN solo que se le añade un comentario en el caso de que no se cumpla la condición y devolviendo con el exit el valor 0.

En el caso de que se cumpla la condición, no se imprimirá dicho mensaje por pantalla y se devolverá al sistema operativo el valor de la suma.

**SUMMTON\_AVISO.S:**

****

En el siguiente programa se realizan una serie de variaciones del programa sumMtoN.s para conseguir el objetivo de mostrar por pantalla el contenido de una string avisando al usuario de que ha introducido los datos incorrectamente al no cumplir la relación M < N.

Además, se le pasa al sistema operativo el valor de 0 en ese caso que implica que ha habido un error.

Para incluir estos objetivos, es necesario pasar a un registro, en este caso eax, el valor de N o M y compararlo con el otro a través de la instrucción cmp. En el caso que M sea estrictamente menor que N, el programa continuará con su correcto funcionamiento, similar al de sumMtoN.s

En el caso de que no se cumpla la condición, se mostrará por pantalla la cadena de caracteres mensaje, para que el usuario conozca el error en la introducción de los datos.

**COMPILACIÓN:**

gcc -m32 -nostartfiles -g -o sumMtoN\_sumMtoN.s

gcc -m32 -nostartfiles -g -o sumMtoN\_aviso sumMtoN\_aviso.s

gcc -m32 -g -o sumMtoN\_aviso sumMtoN\_aviso.c

Cabe destacar el uso de -m32 para usar una máquina de 32 bits, y -g para cargar la tabla de

símbolos. Además se ha añadido el -nostartfiles porque en este caso se utiliza la etiqueta \_start.

**CONCLUSIÓN:**

Durante esta práctica se han realizado llamadas a subrutinas, en este caso sumMtoN. Para ello es necesario realizar un call y una declaración previa de la función pasando al final de la función el valor de retorno con el registro eax. Además, la única forma de pasar los argumentos necesarios a la subrutina es a través de la stack. Se apilan los argumentos con la instrucción push, y se desapilan con la instrucción pop (se podrá añadir un sufijo a cada uno de ellos para determinar el tamaño).

Por último, como es necesario dejar la pila de la misma manera que se ha encontrado previamente a la llamada de la subrutina, se realizará un prólogo y un epílogo para completar dicho fin.

En el prólogo se apilará la dirección antigua del frame pointer para tenerla guardada, y se creará una nueva pila en la parte superior de la anterior pila.

En el epílogo se dejará la pila como se ha encontrado previamente realizando las instrucciones oportunas dependiendo del uso que se haya llevado a cabo de la pila en la subrutina.

Por último, se crean dos programas tanto en asm como en lenguaje c en los que se imprime un mensaje de error en el caso de que no se cumpla la condición M < N.