@author Rodríguez López, Alejandro // UO281827

@author Fernández Ruiz, Pablo // UO282000

@author Cuesta Loredo, Celia // UO284095

@author Quirós Maneiro, Javier // UO285230

O-21-PL05-10

Ingeniería Informática en Tecnologías de la Información.

Índice

Cuestiones

No.1 *{Dirección de memoria a partir de la cual se sitúa el código de paso de parámetros a la función IsValidAssembly(). También debéis indicar cuál es ese código de paso de parámetros en forma de código máquina y mnemónicos.}*

Las direccione en las que se almacena el paso de parámetros son:

Dirección	Código máquina	Mnemónico
00401346	51	push ecx
00401347	53	push ebx
00401348	50	push eax

Tabla 1 - Direcciones de memoria, códigos máquina y mnemónicos de órdenes.

Para hallarlos, colocamos un *breakpoint* en cualquier línea –por ejemplo, la línea 85- y depuramos.

Al iniciar la depuración y llegar al *breakpoint* hacemos click derecho en el código y seleccionamos la opción 'Ir a desensamblado'. Se abrirá una pantalla diferente llamada 'Desensamblado' donde podremos ver la posición de la memoria a la izquierda del todo, a continuación, el código máquina en hexadecimal, y a la derecha los mnemónicos.

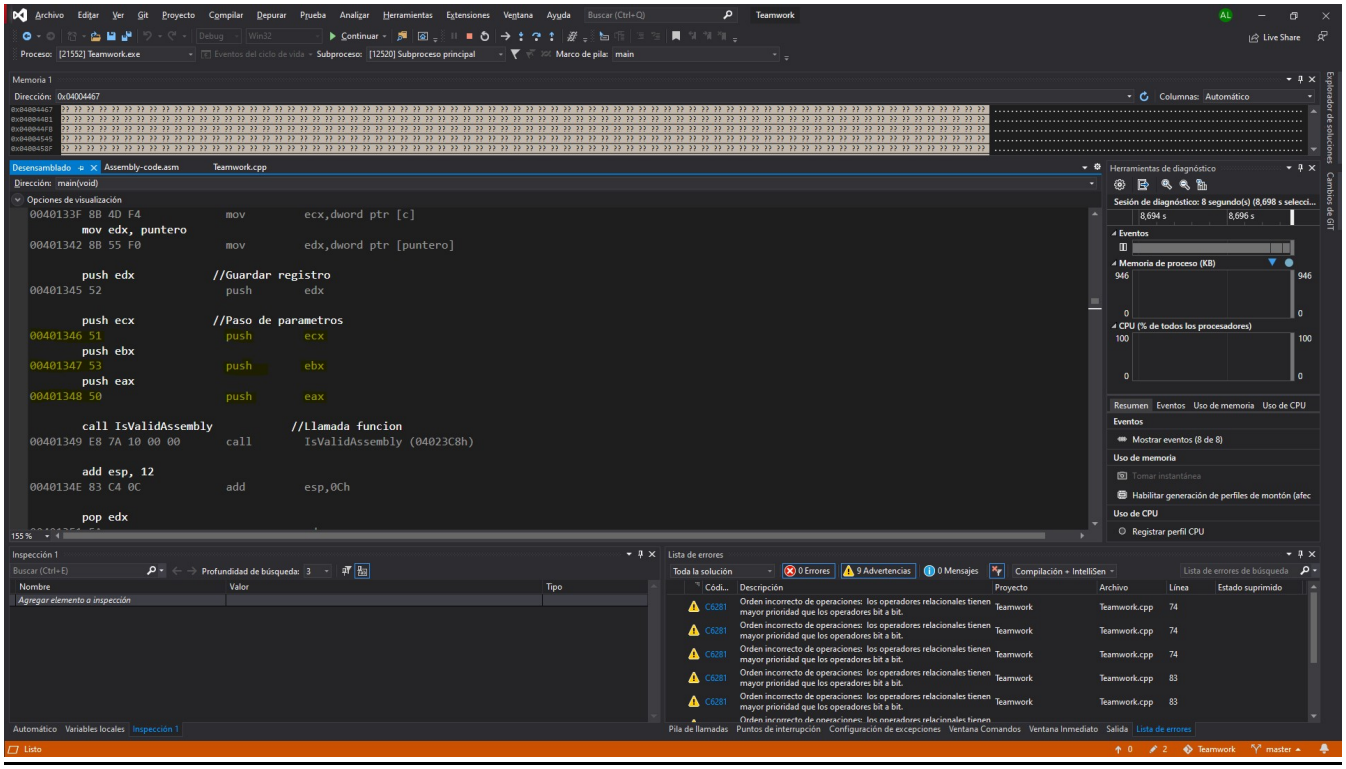

Ilustración 1 - Direcciones de memoria, códigos máquina y mnemónicos de órdenes. Visto desde Visual Studio.

No.2 {*Dirección de la memoria en la que se encuentra la primera cadena que se lee en la primera función indicada en las instrucciones.*}

La dirección de memoria en la que se encuentra la primera cadena que se lee en la primera función es: 0x0019fec8.

Para hallar la dirección colocamos un *breakpoint* en una línea posterior a la declaración de la cadena –por ejemplo, la línea 7- y depuramos.

En la pantalla de 'Inspección' agregamos un elemento a inspección. El elemento sería '&cadena1' ya que 'cadena1' es el nombre de la variable y el & denota que no queremos ver tanto su contenido sino la posición de memoria.

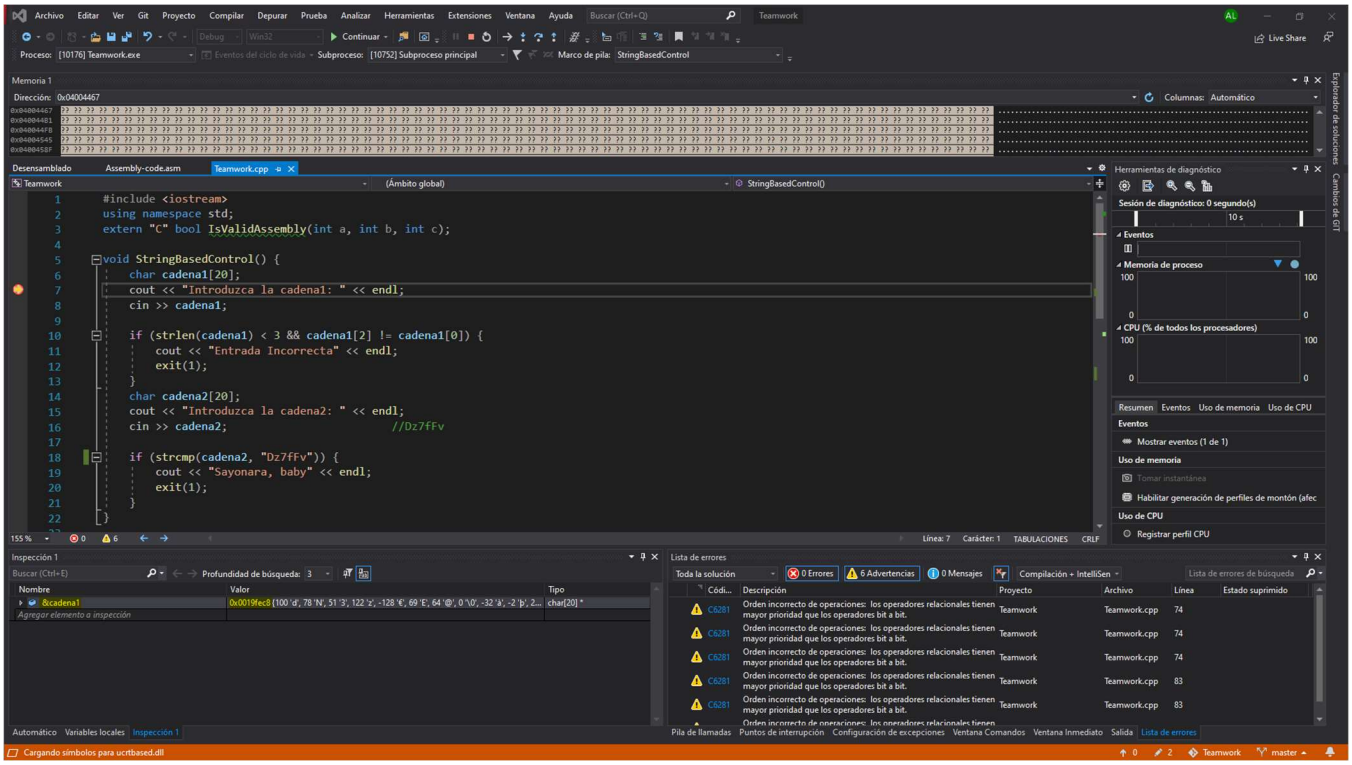

Ilustración 2 - Dirección de memoria de la cadena 'cadena1' visto desde Visual Studio.

No.3{*Dirección de la memoria en la que se sitúa el epílogo de la primera función indicada en las instrucciones y el propio código del epílogo, en forma de código máquina y de mnemónicos.*}

Las direcciones en las que se almacena el epílogo de la primera función son:

Dirección	Código máquina	Mnemónico
0040112B	8B E5	mov esp, ebp
0040112D	5D	pop ebp
0040112E	C3	ret

Tabla 2 - Direcciones de memoria, códigos máquina y mnemónicos de órdenes.

Para hallarlos, colocamos un *breakpoint* en cualquier línea –por ejemplo, la línea 18- y depuramos.

Al iniciar la depuración y llegar al *breakpoint* hacemos click derecho en el código y seleccionamos la opción 'Ir a desensamblado'. Se abrirá una pantalla diferente llamada 'Desensamblado' donde podremos ver la posición de la memoria a la izquierda del todo, a continuación, el código máquina en hexadecimal, y a la derecha los mnemónicos. Concretamente, el epílogo de la primera función se encuentra tras la llave de cierre '}' y serían las 3 órdenes que devuelven *ebp* de la pila y retornan.

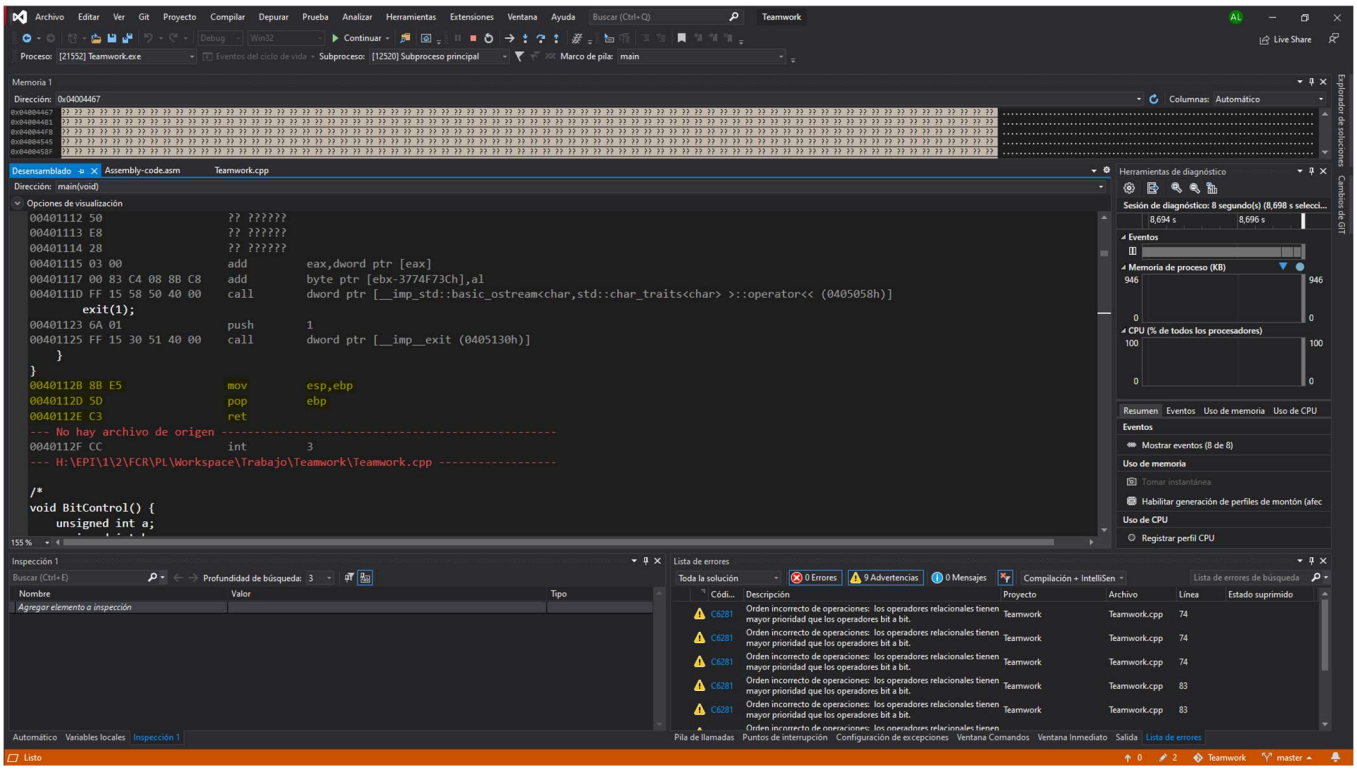

Ilustración 3 - Direcciones de memoria, códigos máquina y mnemónicos de órdenes. Visto desde Visual Studio.

No.4 *{Código máquina de la primera instrucción de ensamblador introducida en el ensamblador en línea de la cuarta función indicada en las instrucciones.}*

Código máquina: 8B 5D FC

Para acceder al código máquina colocamos un *breakpoint* en cualquier línea –por ejemplo, la línea 156- y depuramos.

Al iniciar la depuración y llegar al *breakpoint* hacemos click derecho en el código y seleccionamos la opción 'Ir a desensamblado'. Se abrirá una pantalla diferente llamada 'Desensamblado' donde podremos ver el código máquina.

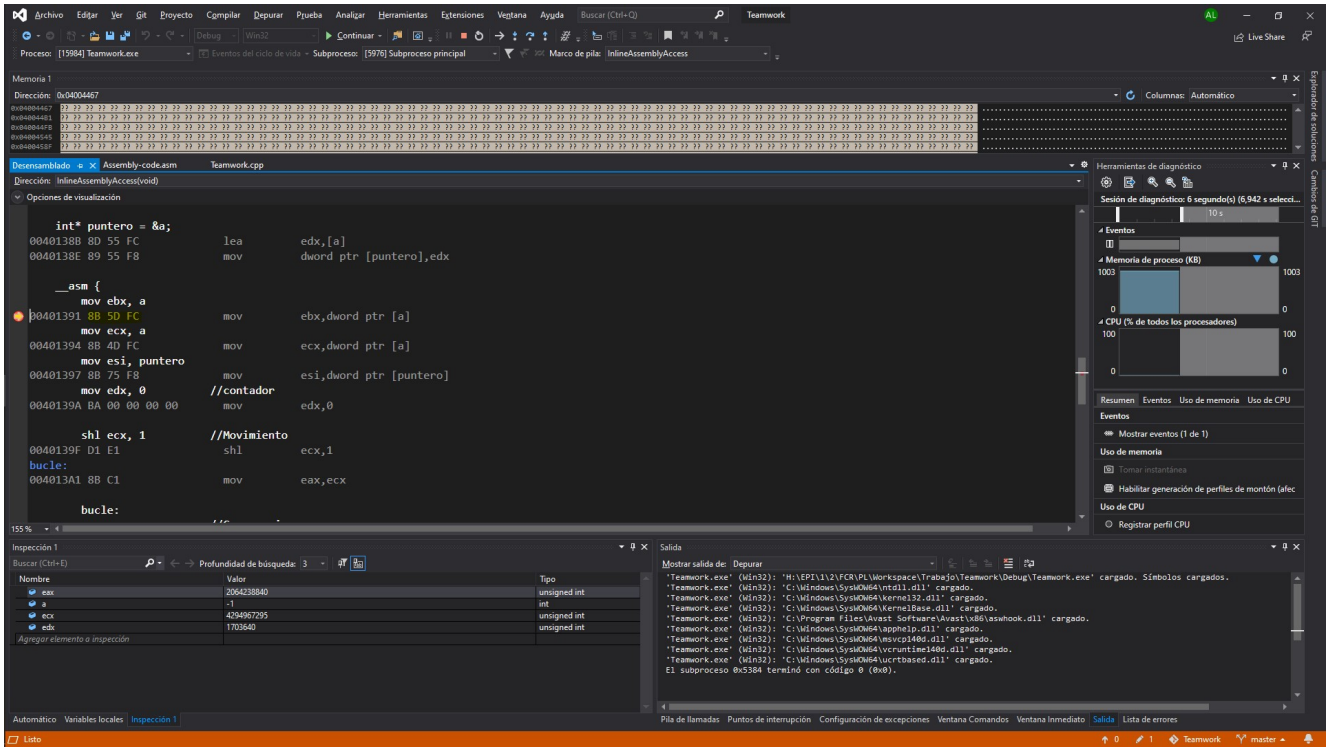

Ilustración 4 - Código máquina de instrucción. Visto desde Visual Studio.

Ejemplos

Entrada válida

A continuación, se presenta un ejemplo de entradas en las que el programa queda validado al completo.

StringBasedControl():

Cadena 1 = “aaaaaaa”

Cadena 2 = “Dz7fFv”

BitControl():

Natural 1 = 32

Natural 2 = 0

CheckInAsmbyFile():

Entero 1 = 0

Entero 2 = 1556774

Entero 3 = 2097151

InlineAssemblyAccess():

Entero = -1

A continuación, presentamos una imagen de la consola, resultando en una ejecución sin errores.

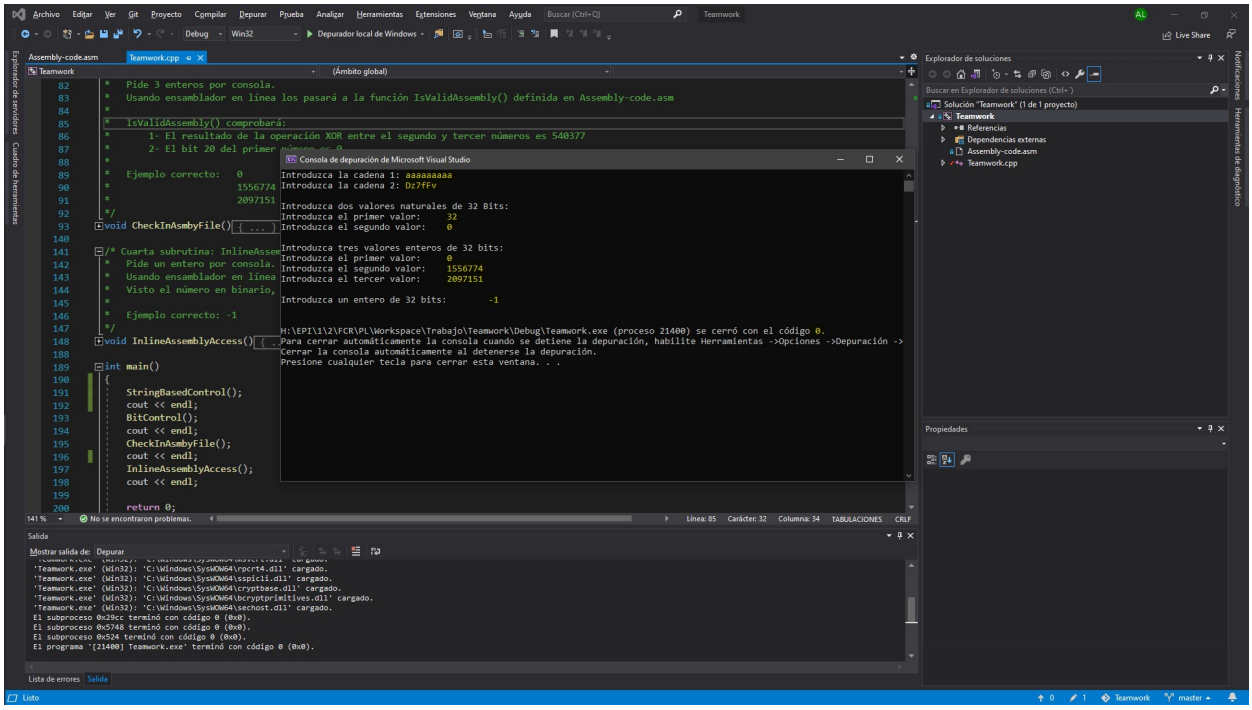

Ilustración 5 - Ejemplo de entrada correcta.

Entrada inválida – Subrutina[0]

En esta sección presentamos algún ejemplo de entrada tal que invalida la subrutina primera.

Cadena 1 = “abbbbb”

La subrutina queda invalidada debido a que el carácter en posición 0 es diferente al carácter en posición 2.

Cadena 2 = “Dz7Ffv”

La subrutina queda invalidad debido a que la segunda cadena no es exactamente “Dz7fFv”.

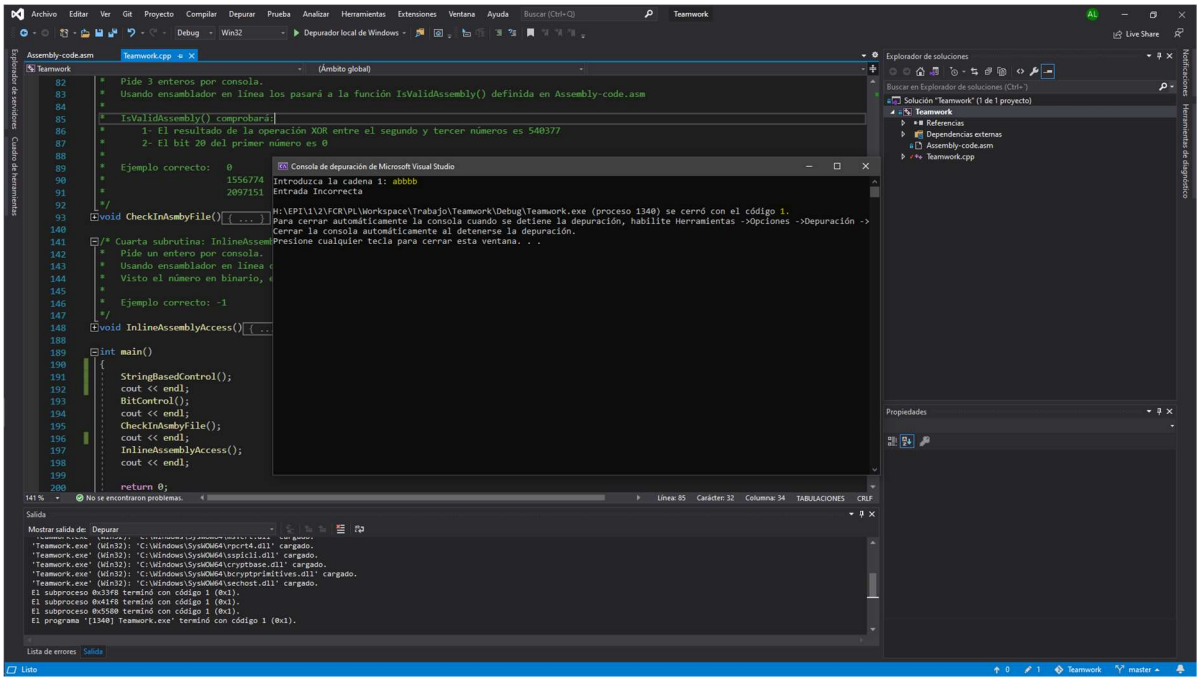

Ilustración 6 - Ejemplo 1 de entrada incorrecta.

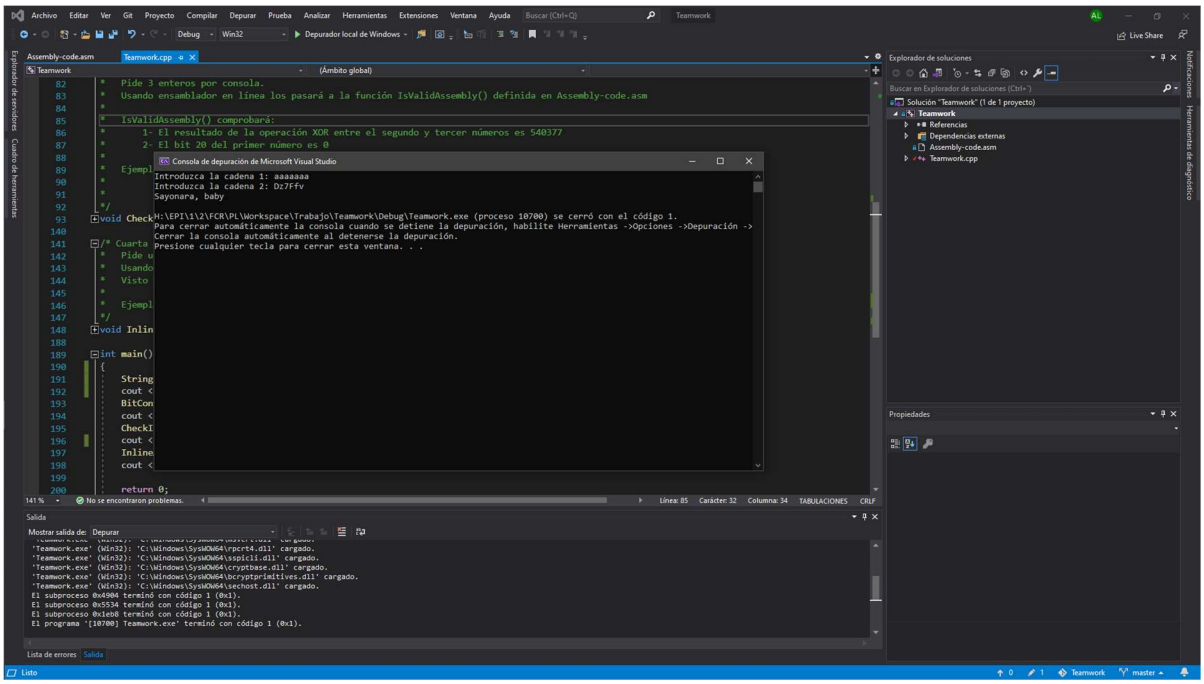

Ilustración 7 - Ejemplo 2 de entrada incorrecta.

Entrada inválida – Subrutina[1]

Se presenta algún ejemplo de entrada inválida para la segunda subrutina.

Natural 1 = 256
Natural 2 = 0
La subrutina queda invalidada debido a que el bit 8 del primer número es diferente al 9 del segundo.

Natural 1 = 0
Natural 2 = 0
La subrutina queda invalidada debido a que los 16 bits más fuertes del primer número más los 16 más débiles del segundo no es mayor que 8489.

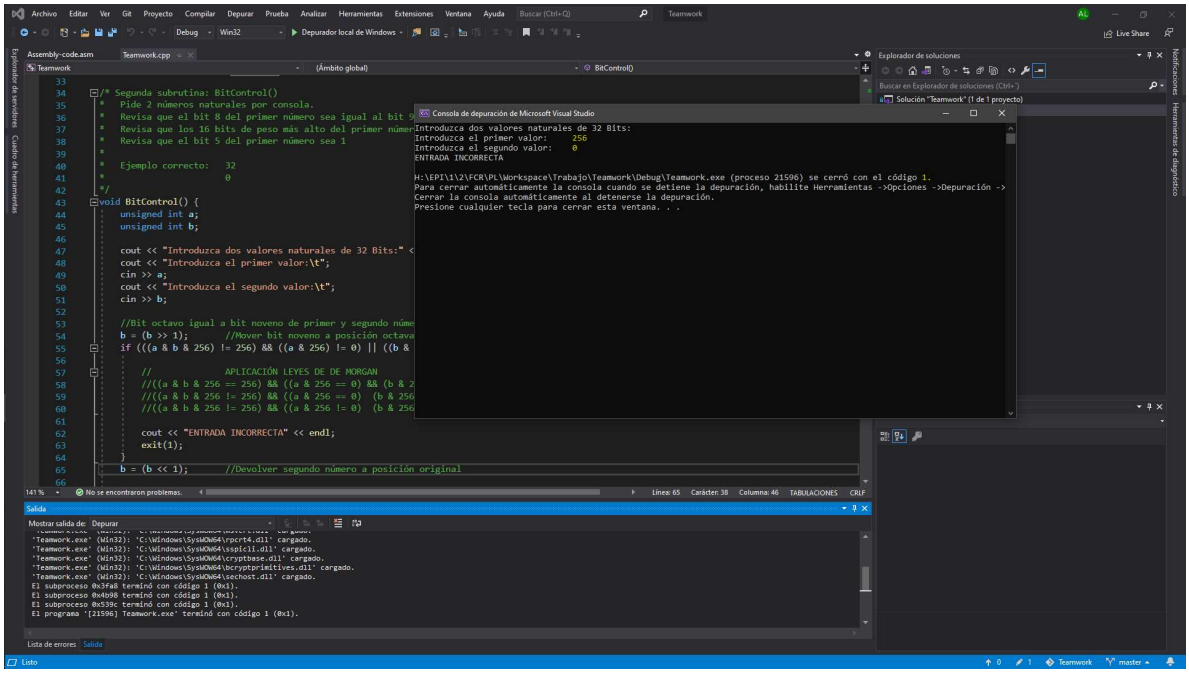

Ilustración 8 - Ejemplo 3 de entrada incorrecta.

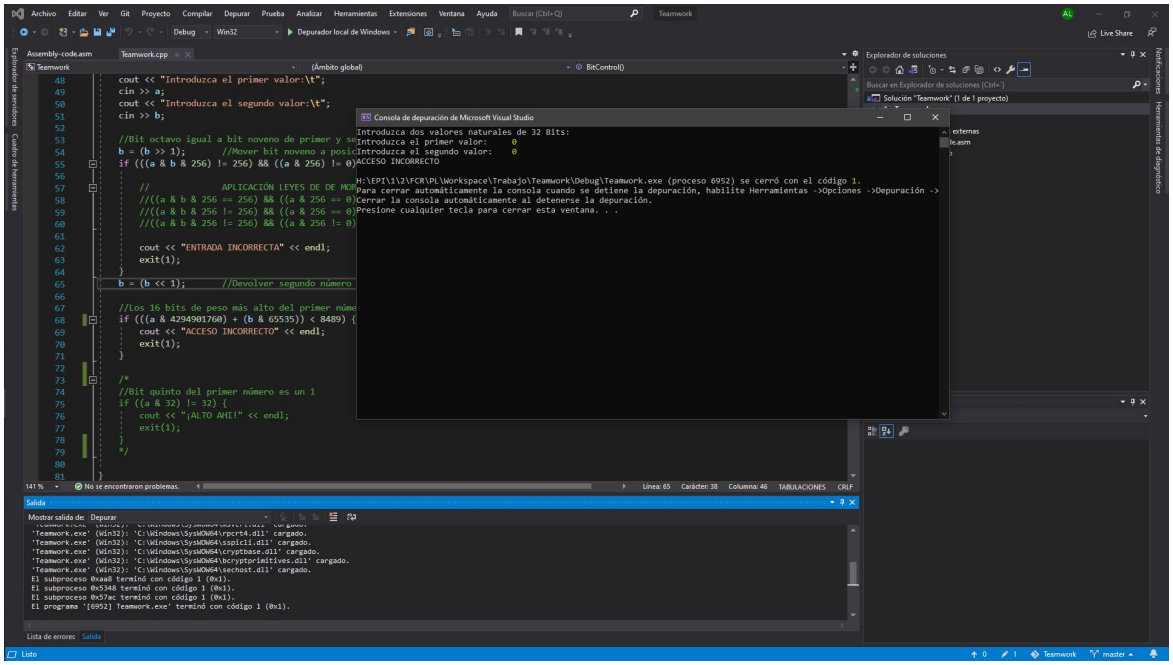

Ilustración 9 - Ejemplo 4 de entrada incorrecta.

Entrada inválida – Subrutina[2]

A continuación, se encuentran algunos ejemplos de entradas inválidas para la tercera subrutina.

Entero 1 = 1048576
La subrutina queda invalidada debido a que el primer entero tiene un 1 en su bit 20.

Entero 2 = 508198
Entero 3 = 540377
La subrutina queda invalidada debido a que el resultado de la operación xor entre el segundo y tercer valor es diferente de 540377.

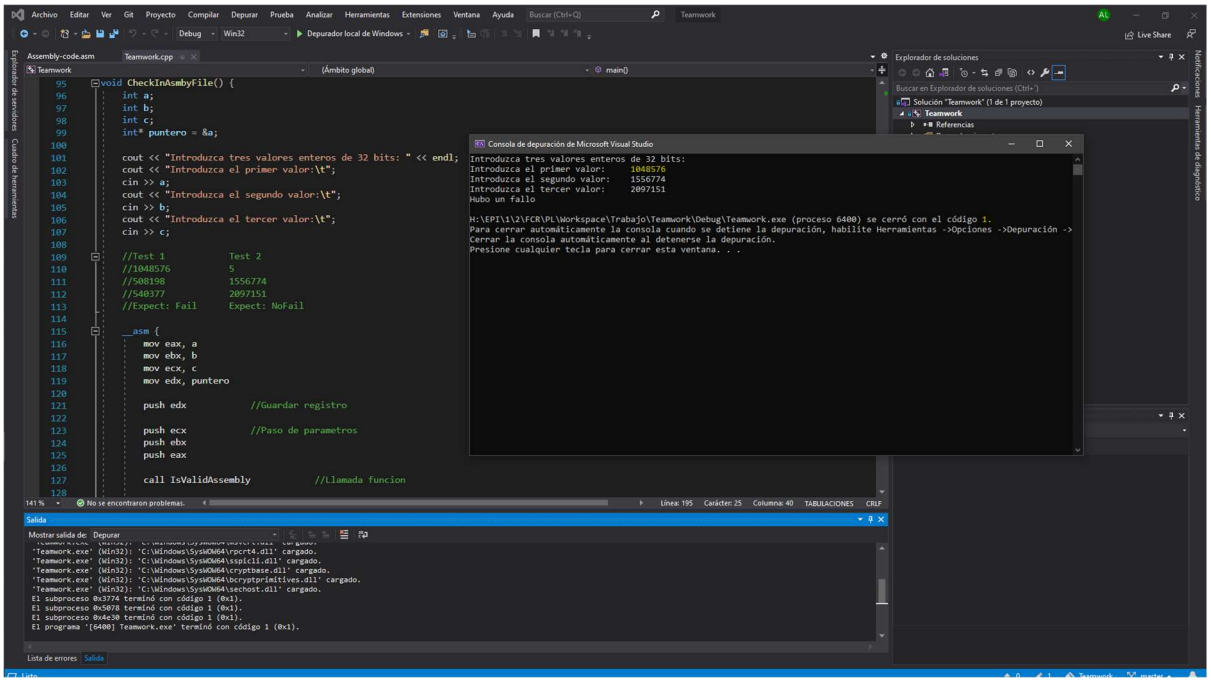

Ilustración 10 - Ejemplo 5 de entrada inválida.

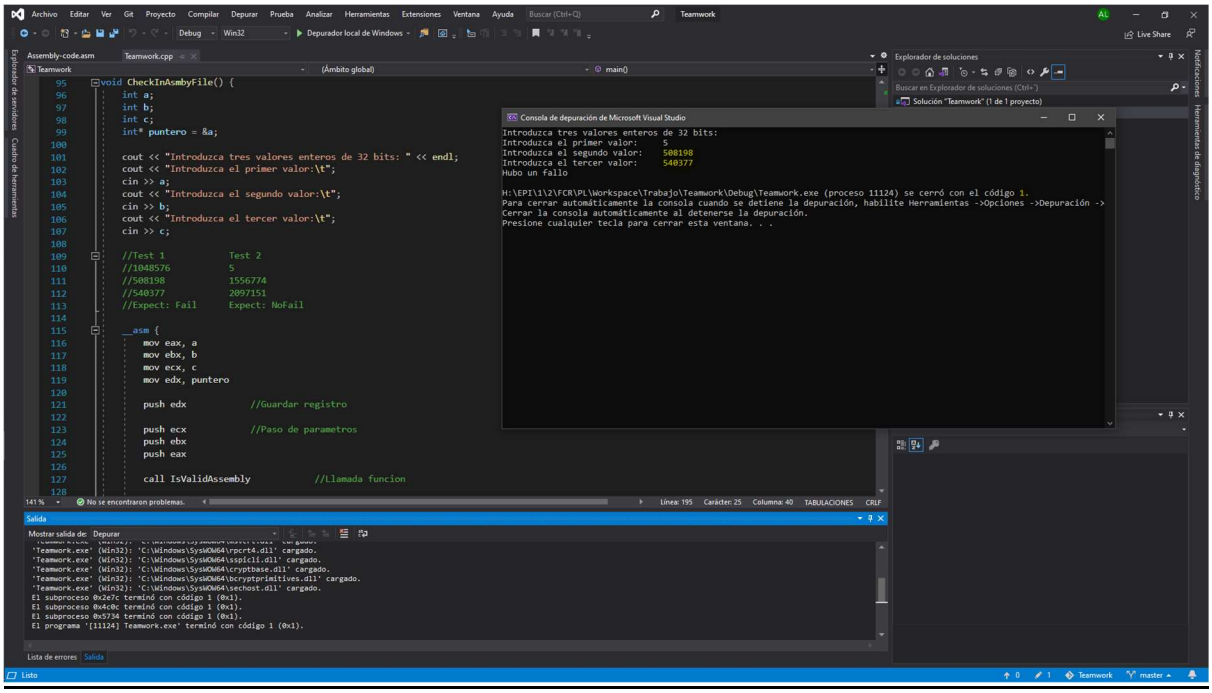

Ilustración 11 - Ejemplo 6 de entrada inválida.

Entrada inválida – Subrutina[3]

Finalmente, presentamos un ejemplo de entrada inválida para la última subrutina.

Entero = 4

La subrutina queda invalidada debido a que el entero visto como binario de 32 bits no es capicúa.

0000 0000 0000 0000 0000 0000 0000 0100

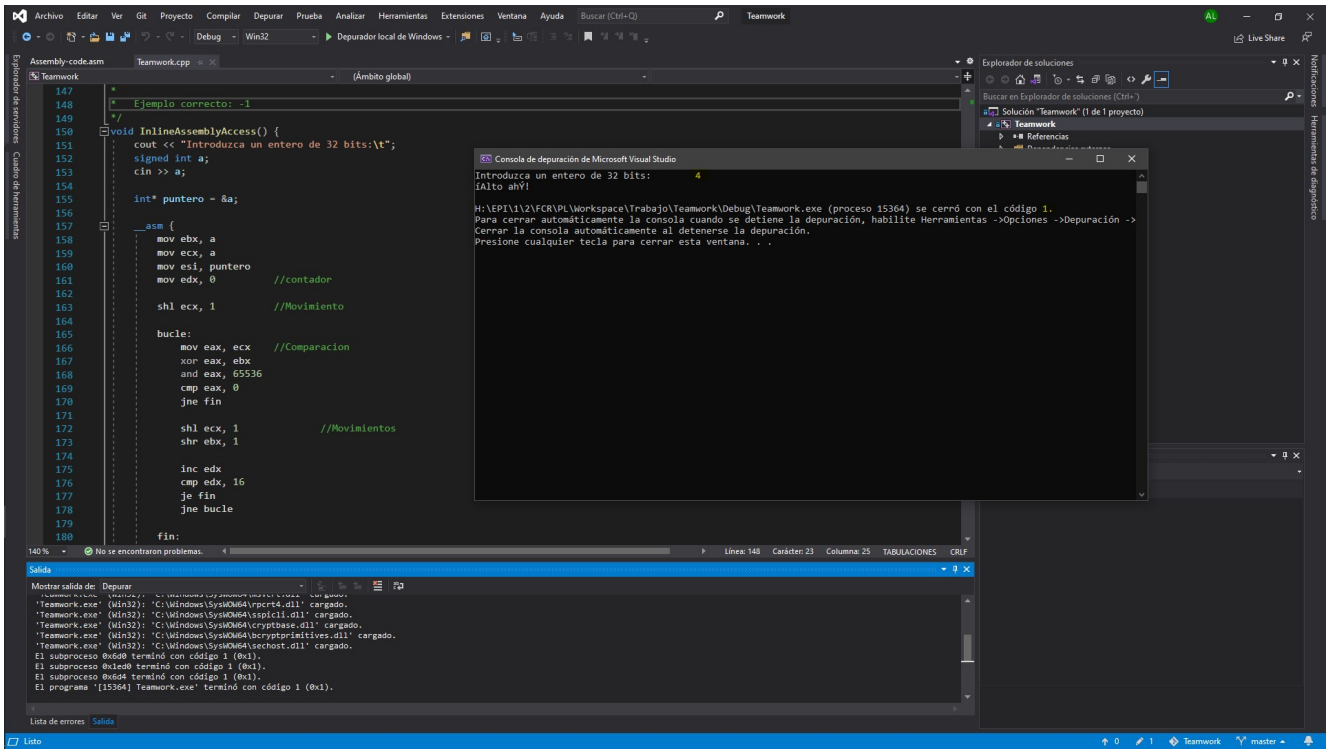

Ilustración 12 - Ejemplo 7 de entrada inválida.

División del trabajo

Durante las cuatro semanas en las que hemos llevado a cabo la primera parte del proyecto, hemos repartido el trabajo en función de los UO. De forma que el UO más bajo se encargaría de la primera función, y el más alto la cuarta. Al pasar la semana transferiríamos lo hecho al compañero/a siguiente. De esta forma, podríamos ver todas las funciones y aportar diferentes puntos de vista y formas de optimizar el código.

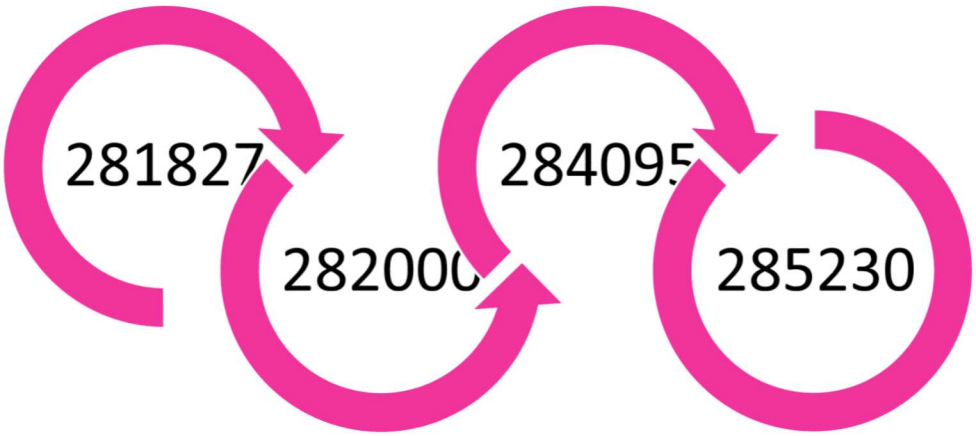

Ilustración 13 - Iteración de las subrutinas.

La mayoría de las subrutinas se terminaron durante la segunda o tercera semana. Por lo que la última semana se dedicó especialmente a redactar este documento.

Respecto a las horas concretas, cada alumno invirtió de media el mismo número de horas en cada subrutina. Concluyendo en las siguientes horas invertidas por alumno en cada subrutina:

Subrutina	Tiempo / Alumno	Tiempo total	Tiempo total	Tiempo total
1	1	4	12	48
2	2	8		
3	5	20	36	
4	4	16		

Tabla 3 - Representación de las horas invertidas.

Como se puede observar, las dos primeras subrutinas no supusieron mayor problema, sin embargo, las dos segundas fueron más problemáticas debido a su dependencia de la programación en Assembly, a la que nos tuvimos que hacer.