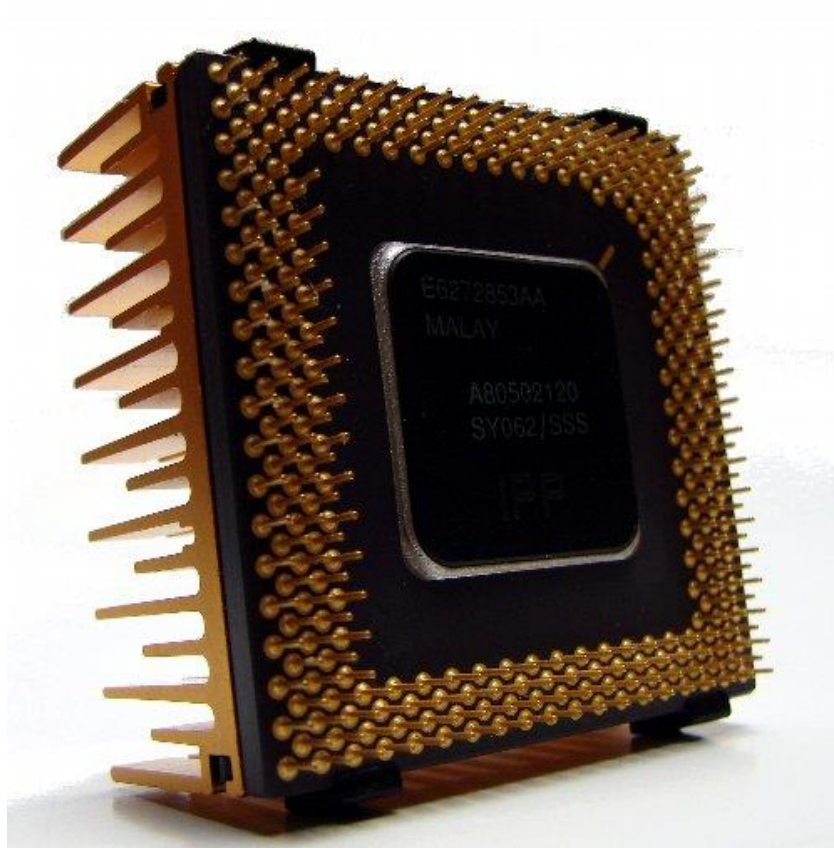


PROYECTO

MATERIAL ADICIONAL F-II

- Ensamblador inline en C++
- Ejercicios para desarrollar individualmente



2017

Proyecto de evaluación del rendimiento

F-II. Implementación de un benchmark reducido y evaluación del procesamiento de arquitecturas PC convencionales

Arquitectura de los Computadores

Grado en Ingeniería Informática

Dpto. Tecnología Informática y Computación

Universidad de Alicante

MATERIAL ADICIONAL F-II

PROYECTO DE EVALUACIÓN DEL RENDIMIENTO

I. CONTENIDOS, ENTREGA Y EVALUACIÓN (INDIVIDUAL)

El material adicional para el desarrollo de la F-II de la práctica 1 consta de:

- Información sobre la puesta a punto de programas de ensamblador inline con C++ empleando la herramienta de desarrollo de software Visual Studio .NET
- Ejercicios que deberán ser resueltos de manera individual. Los ejercicios forman parte de la guía de puesta a punto. Además, se propone el desarrollo de un programa en ensamblador inline en el apartado VIII.
- Información del benchmark SPEC, ejercicio individual y ejercicio de grupo.

La entrega constará de los archivos correspondientes al desarrollo de los ejercicios, una memoria explicativa del ejercicio propuesto en el apartado VII y de los ejercicios propuestos en el anexo de SPEC, F-II (3)

La evaluación individual de la práctica es la siguiente:

Aspecto a considerar	Calificación
Ejercicios guiados	10%
Desarrollo de los ejercicios propuestos	50 %
Memoria explicativa de los ejercicios a resolver	40 %

II. VISUAL STUDIO .NET

Visual Studio .NET implementa dos contenedores conceptuales que facilitan la administración de los elementos necesarios para el desarrollo de software, como pueden ser: referencias, conexiones de datos, carpetas y archivos. Estos contenedores se denominan soluciones y proyectos. Una solución incluye uno o varios proyectos más los archivos y metadatos que ayudan a definir la solución como un todo. Las carpetas de soluciones permiten organizar proyectos relacionados en grupos y gestionar esos grupos de proyectos. Un proyecto incluye un conjunto de archivos de código fuente más los metadatos relacionados,

como referencias de componentes e instrucciones de generación. Normalmente, los proyectos crean uno o más archivos de salida cuando se generan.

Cuando se crea un nuevo proyecto, Visual Studio genera una solución automáticamente, pudiendo agregar otros proyectos a la solución. El Explorador de soluciones proporciona una vista gráfica de toda la solución que ayuda a administrar los proyectos y archivos durante el desarrollo de la aplicación. También pueden crearse soluciones en blanco, es decir, sin proyectos, lo que permite modificar los proyectos, las carpetas y los archivos que la conforman de forma independiente. Dado que cada proyecto o solución comprende un directorio y su contenido, las soluciones y los proyectos se pueden mover, copiar o eliminar empleando el Explorador de Windows.

III. CREAR UNA NUEVA SOLUCIÓN Y UN NUEVO PROYECTO DE C++

En este primer ejercicio guiado se creará una nueva Solución Visual Studio .NET, denominada *EjerciciosEnsamblador*, que se utilizará para desarrollar los ejercicios de programación contenidos en esta práctica, y un nuevo Proyecto de Visual C++, denominado *Ejemplo1*. Para ello, realizar las siguientes acciones:

1. Abrir Visual Studio .NET.
2. Acceder al menú **Archivo** de Visual Studio .NET, expandir la opción **Nuevo** y seleccionar la opción **Proyecto...**
3. En el cuadro de diálogo **Nuevo Proyecto**, realizar las siguientes acciones:
 - a. En la sección **Plantillas instaladas**, expandir la opción **Visual C++**, elegir la opción **General** y seleccionar **Proyecto vacío**.
 - b. Comprobar que la casilla de verificación **Crear directorio para la solución** se encuentra activada.
 - c. En el cuadro de texto **Nombre** introducir *Ejemplo1*.
 - d. Seleccionar una carpeta como ubicación para la solución de Visual Studio mediante el botón **Examinar....**
 - e. En el cuadro de texto **Nombre de la solución** se mostrará el mismo nombre asignado al Proyecto de Visual C++. Modificar su contenido, introduciendo *EjerciciosEnsamblador* en el cuadro de texto **Nombre de la solución**.
 - f. Finalmente, comprobar que está activada la casilla de verificación **Crear directorio para la solución** y hacer clic en **Aceptar**.

Utilizando el Explorador de archivos de Windows, se puede comprobar que se ha creado la carpeta *EjerciciosEnsamblador* en la ubicación deseada. Y que en su contenido se encuentra el archivo de solución *EjerciciosEnsamblador.sln* y la carpeta *Ejemplo1*. A su vez, la carpeta *Ejemplo1* contiene los archivos que son propios de un Proyecto de Visual C++.

Una vez cerrado Visual Studio .NET, para acceder a la Solución *EjerciciosEnsamblador* y al proyecto *Ejemplo1*, bastará con hacer doble clic sobre el archivo *EjerciciosEnsamblador.sln* desde su ubicación, empleando para ello el Explorador de archivos de Windows.

IV. CREAR UN ARCHIVO DE CÓDIGO

Una vez creada la solución *EjerciciosEnsamblador* y el Proyecto de Visual C++ *Ejemplo1*, se creará un archivo de código denominado *Suma.cpp*. Para ello, hacer:

1. Sobre la ventana **Explorador de soluciones**, hacer clic con el botón derecho sobre el nombre del Proyecto de Visual C++ *Ejemplo1*, expandir la opción **Agregar** y seleccionar la opción **Nuevo elemento...**
2. En el cuadro de diálogo **Agregar nuevo elemento**, realizar las siguientes acciones:
 - a. En la sección **Plantillas instaladas**, expandir la opción **Visual C++**, elegir la opción **Código** y seleccionar **Archivo C++ (.cpp)**.
 - b. En el cuadro de texto **Nombre** introducir *Suma*.
 - c. Comprobar que el archivo se creará en la ubicación adecuada.
 - d. Hacer clic en **Agregar**.

Empleando el **Explorador de soluciones** de Visual Studio .NET y el **Explorador de archivos** de Windows, se puede comprobar que el archivo *Suma.cpp* ha sido creado.

3. Agregar el siguiente código sobre el archivo de código *Suma.cpp*.

```
// Ensamblador en línea. Procesador: x86 (32 bits)

#include <stdio.h>

int suma(int a, int b);

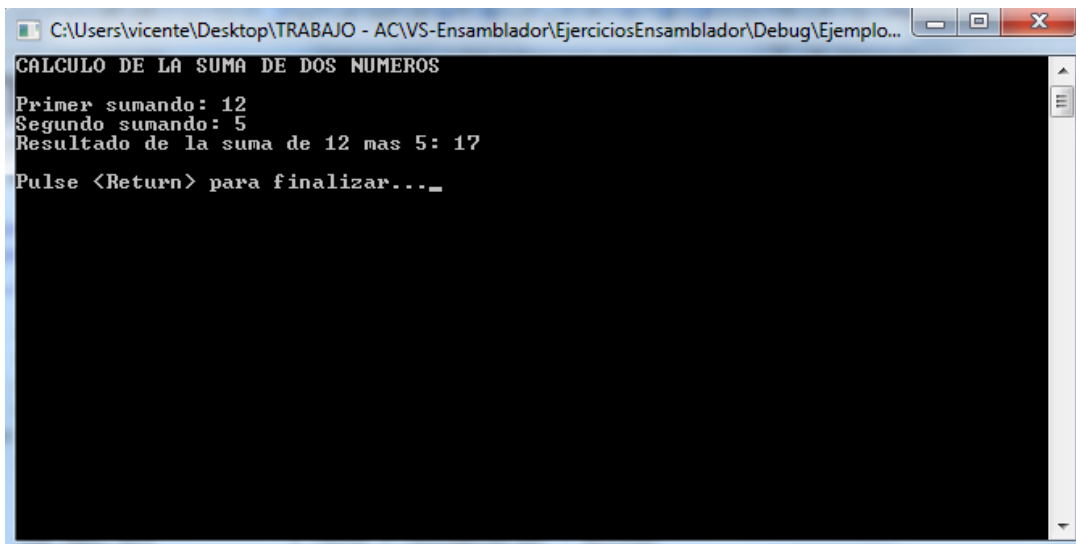
int main( void )
{
    int x = 0;
    int y = 0;

    printf( "CALCULO DE LA SUMA DE DOS NUMEROS\n");
    printf_s("\n");
    printf_s("Primer sumando: ");
    scanf_s("%d",&x);
    printf_s("Segundo sumando: ");
    scanf_s("%d",&y);
    getchar();
    printf_s("Resultado de la suma de %d mas %d: %d\n", x, y, suma(x, y));
    printf_s("\n");
    printf_s("Pulse <Return> para finalizar...");
    getchar();
}

int suma(int a, int b)
{
    __asm
```

```
{  
    mov eax, a      ; Almacena en EAX el primer argumento  
    mov ecx, b      ; Almacena en ECX el segundo argumento  
    add eax, ecx    ; Suma de los operandos, deja el resultado en eax  
    // Devuelve el resultado de EAX  
}  
}
```

4. Una vez editado el código, seleccionar la opción **Guardar Suma.cpp** del menú **Archivo**, o, hacer clic en el botón **Guardar** de la **Barra de herramientas** de Visual Studio .NET.
5. Seleccionar la opción **Generar solución** del menú **Generar**. Si se produjeran errores durante el proceso de compilación deberán resolverse antes de ejecutar el código.
6. Para comprobar el funcionamiento del programa, seleccionar la opción **Iniciar la depuración** del menú **Depurar**, o bien, hacer clic sobre el botón **Iniciar depuración** de la **Barra de herramientas** de Visual Studio.



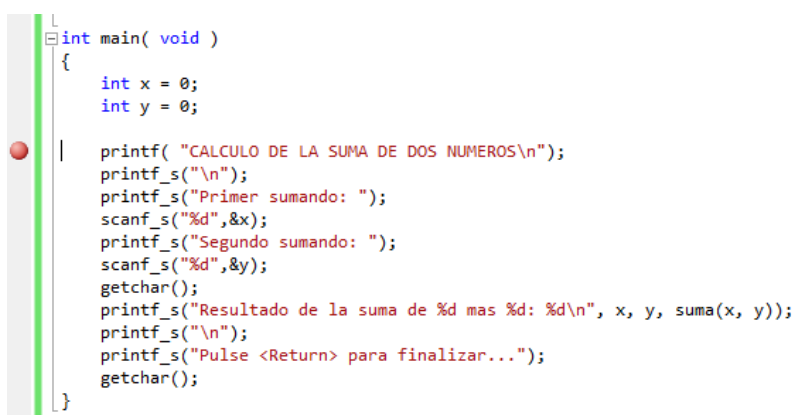
V. DEPURACIÓN PASO A PASO

El código de los programas informáticos puede contener diversos tipos de errores. La forma de detectar los errores lógicos es probar el programa para comprobar que el resultado es el esperado. Las pruebas son una parte integrante del proceso de desarrollo de software. En algunos casos de error, las pruebas pueden indicar que el resultado de un programa es incorrecto, aunque pueden no proporcionar ninguna pista acerca de qué parte del código ha causado el problema en la ejecución. En estos casos es útil emplear la depuración paso a paso.

La depuración paso a paso consiste en ejecutar línea a línea el código para inspeccionar los valores que van adoptando las variables a lo largo del flujo de la ejecución del código. El menú **Depurar** contiene dos comandos para avanzar paso a paso por el código: **Paso a paso por instrucciones** y **Paso a paso por procedimientos**. Es posible iniciar estos métodos de depuración paso a paso pulsando las teclas **<F11>** y **<F10>**, respectivamente.

Los métodos de depuración **Paso a paso por instrucciones** y **Paso a paso por procedimientos** sólo se diferencian en la forma en que tratan las llamadas a funciones. Ambos comandos indican al depurador que ejecute la siguiente línea de código. Si la línea contiene una llamada a una función, **Paso a paso por instrucciones** sólo ejecuta la llamada en sí y, a continuación, se detiene en la primera línea de código incluida en la función. Sin embargo, **Paso a paso por procedimientos** ejecuta toda la función y después se detiene en la primera línea que está fuera de ella.

En algunas ocasiones, durante la depuración, es posible que se desee ejecutar el código hasta un punto determinado y después interrumpir la ejecución. En estos casos, se introducirá un Punto de interrupción en ese lugar, pudiendo continuar la ejecución en el modo de depuración paso a paso. Para introducir un Punto de interrupción, basta con seleccionar la línea de código deseada haciendo clic a la izquierda de la línea sobre la barra vertical de selección de líneas de código. Los puntos de interrupción se representan mediante un círculo de color rojo. Para eliminar un Punto de interrupción basta con hacer clic sobre él.



```

int main( void )
{
    int x = 0;
    int y = 0;

    printf( "CALCULO DE LA SUMA DE DOS NUMEROS\n");
    printf_s("\n");
    printf_s("Primer sumando: ");
    scanf_s("%d",&x);
    printf_s("Segundo sumando: ");
    scanf_s("%d",&y);
    getchar();
    printf_s("Resultado de la suma de %d mas %d: %d\n", x, y, suma(x, y));
    printf_s("\n");
    printf_s("Pulse <Return> para finalizar...");
    getchar();
}

```

En cualquier momento puede detenerse la depuración seleccionando la opción **Detener depuración** del menú **Depurar**, o bien, hacer clic sobre el botón **Detener depuración** de la **Barra de herramientas** de Visual Studio.

Como ejercicio, se propone que se realice la depuración del archivo de código *Suma.cpp*, introduciendo un punto de interrupción al principio del programa y ejecutando paso a paso por procedimientos hasta la línea de código que incluye el punto de llamada a la función *suma*.

VI. CREAR UNA APLICACIÓN DE WINDOWS FORMS DE VISUAL C++ Y AGREGAR CÓDIGO (VERSIONES 2012-2013-2015)

Incluir código ensamblador inline sobre aplicaciones de escritorio de Windows consiste sencillamente en agregar código asociado a los eventos de los controles incluidos en un Windows Form. Para ello realizar las siguientes acciones:

1. Sobre la ventana **Explorador de soluciones**, hacer clic con el botón derecho sobre el nombre de la Solución *EjerciciosEnsamblador*, expandir la opción **Agregar** y seleccionar la opción **Nuevo Proyecto...**
2. En el panel **Tipos de proyecto** del cuadro de diálogo **Agregar nuevo proyecto**, seleccionar **CLR** en el nodo **Visual C++** y, a continuación, seleccionar **Proyecto vacío de CLR** en el panel **Plantillas instaladas de Visual Studio**. En el cuadro de texto **Nombre** introducir *Ejemplo2* y

- hacer clic en **Aceptar**. Se habrá creado del proyecto de aplicación de Windows Forms de Visual C++ denominado *Ejemplo2*.
3. Comprobar mediante el **Explorador de soluciones** de Visual Studio .NET y el **Explorador de archivos** de Windows que se ha creado el proyecto *Ejercicio2* que contiene un conjunto de archivos propios de una aplicación de Windows Forms de Visual C++ en la carpeta de solución *EjerciciosEnsamblador*.
 4. Con el botón derecho sobre el proyecto añadiremos un nuevo elemento: **Añadir → Nuevo Elemento...**
 5. Seleccionaremos el elemento **Windows Form** dentro de las UI bajo Visual C++ y cambiaremos el nombre por defecto (MyForm.h) por Form1.h. Puede comprobarse que se abre el archivo de diseño del Window Form *Form1.h* en el **Diseñador de Windows Forms**, apareciendo un área en la que puede agregar los controles que desee sobre la superficie de diseño de controles.
 6. En la ventana del proyecto, modificaremos el fichero *Form1.cpp* de la siguiente forma:

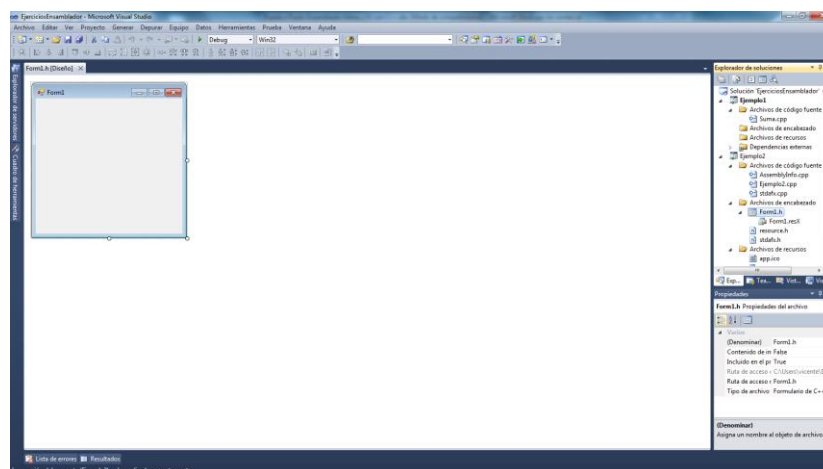
```
#include "Form1.h"

using namespace System;
using namespace System::Windows::Forms;

[STAThread]
void Main(array<String^>^ args)
{
    Application::EnableVisualStyles();
    Application::SetCompatibleTextRenderingDefault(false);

    Ejemplo2::Form1 form;
    Application::Run(%form);
}
```

7. Finalmente, con el botón derecho sobre el proyecto *Ejemplo2*, seleccionaremos las **Propiedades** del mismo. En la opción **Propiedades de configuración → Linkado → Sistema** seleccionaremos **Windows (/SUBSYSTEM:WINDOWS)** para el **Subsistema**. Además para **Propiedades de configuración → Linkado → Avanzado**, teclearemos **Main** para la opción **Punto de entrada**.
8. Si ejecutamos el proyecto con F5 deberíamos ver el formulario vacío.



9. Sobre la ventana **Explorador de soluciones**, hacer clic con el botón derecho sobre el nombre del proyecto de Aplicación de Windows Forms de Visual C++ denominado *Ejemplo2* y seleccionar la opción **Establecer como proyecto de inicio**.
10. Desde el **Cuadro de herramientas**, arrastrar sobre la vista diseño del archivo *Suma1.h* los controles y modificar sus propiedades tal como se indica en la siguiente tabla:

Control	(Name)	Text	Font - Bold
Label	label1	Dividendo	
TextBox	textBox1		
Label	label2	Divisor	
TextBox	textBox2		
Button	buton1	Calcular	
Label	label3	Cociente	
TextBox	textBox3		
Label	label4	Resto	
TextBox	textBox4		
Label	label5	DIVISIÓN ENTERA DE DOS NÚMEROS	True

La apariencia del Windows Form *Form1.h* puede quedar de la siguiente forma:

11. En la vista diseño del archivo de Windows Form denominado *Form1.h*, hacer doble clic sobre el control de tipo *Button* denominado *button1* para acceder al código asociado al procedimiento de evento *Click* y, agregar el siguiente código.

```
private: System::Void button1_Click(System::Object^ sender,
    System::EventArgs^ e) {

    int a = 0;
    int b = 0;
    int c = 0;
    int r = 0;

    a = Convert::ToInt32(this->textBox1->Text);
    b = Convert::ToInt32(this->textBox2->Text);
```



```

        division_entera(a,b,&c,&r);
        this->textBox3->Text = Convert::ToString(c);
        this->textBox4->Text = Convert::ToString(r);
    }

```

12. Al principio del archivo de código, tras de la directiva `#pragma once` incluida, agregar siguiente código que describe el prototipo de la función `división_entera`:

```
void division_entera(int dividendo, int divisor, int *cociente, int *resto);
```

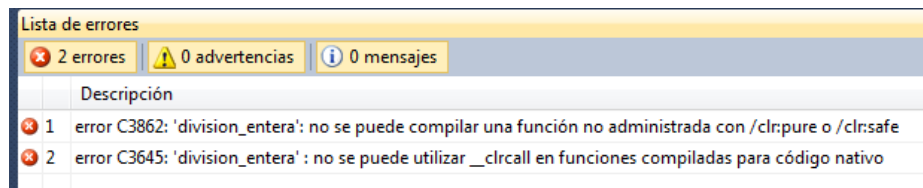
13. Finalmente, agregar el código de la función `división_entera` al final del archivo de código, fuera de cualquier bloque lógico de código.

```

void division_entera(int dividendo, int divisor, int *cociente, int *resto)
{
    int lcociente = 0;
    int lresto = 0;
    __asm
    {
        mov ebx, dividendo    ; Almacena en EBX el dividendo
        mov ecx, divisor      ; Almacena en ECX el divisor
        mov edx, lcociente    ; Inicializa EDX a cero
        mov eax, ebx          ; Inicializa EAX con el dividendo
    bucle:
        cmp eax, ecx          ; Si resto < divisor, finaliza el bucle
        js fin
        inc edx               ; Incrementa el cociente
        sub eax, ecx          ; resto = restoanterior - divisor
        jmp bucle
    fin:
        mov lcociente, edx
        mov lresto, eax
    }
    *cociente = lcociente;
    *resto = lresto;
}

```

14. Guardar los cambios y generar la solución. Al generar la solución aparecen los errores de compilación. Estos errores de compilación están relacionados con las opciones de compilador establecidas por defecto en Visual Studio .NET.



15. Para hacer desaparecer los errores, acceder a la opción **Propiedades de Ejemplo2...** del menú **Proyecto** de Visual Studio .NET. En el cuadro de diálogo **Páginas de propiedades de Ejemplo2** realizar las siguientes acciones:

- Expandir la opción **Propiedades de Configuración**.
- A continuación, expandir la opción **C/C++** y seleccionar la opción **General**.
- Modificar el valor de la siguiente propiedad:

Propiedad	Compatible con Common Language Runtime
Nuevo valor	Compatible con Common Language Runtime (/clr)

- Hacer clic en **Aceptar**.

16. Nuevamente, generar la solución y comprobar que ahora no se devuelve ningún error.

17. Iniciar la depuración para comprobar los resultados obtenidos.

VII. OPCIONES DE COMPILACIÓN

Para acceder a las opciones de compilación, seleccionar **Propiedades de Ejemplo2...** del menú **Proyecto** de Visual Studio .NET. En el cuadro de diálogo **Páginas de propiedades de Ejemplo2**, expandir la opción **Propiedades de Configuración**. Finalmente, al expandir la opción **C/C++** aparecen las opciones de compilación agrupadas por categorías. Recorrer cada una de las categorías para conocer sus opciones. Algunas de las opciones más relevantes son:

- Optimización. Deberá quedar **Deshabilitada (/Od)** al desarrollar programas de prueba de rendimiento.
- Archivos de salida. Para obtener el archivo de ensamblador, seleccionar **Ensamblado con código fuente (/FAs)** en la opción **Resultados de Ensamblado**. Al generar, se crea el archivo .asm en la carpeta *Debug* de la carpeta del Proyecto de Visual C++. Así, al ajustar esta opción y generar nuevamente la solución *Ejemplo2*, puede comprobarse que en la carpeta *Debug* de la

carpeta del proyecto se habrá creado el archivo *Ejemplo2.asm*. Este archivo contiene el código ensamblador correspondiente al programa. Conviene comprobar cómo cada instrucción de ensamblador incluida en el bloque `__asm{}` corresponde con una única instrucción después de la compilación.

Línea de comandos. Se visualiza las opciones de la línea de comandos del compilador.

VIII. EJERCICIO

Se propone desarrollar en lenguaje ensamblador del x86 un programa ensamblador que realice la ordenación de un vector mediante el algoritmo de la burbuja.

A continuación se expresa el algoritmo de la burbuja para ordenación de vectores:

```
para i=0 hasta n-1 hacer
  para j=n hasta i+1 hacer
    si (vector[j-1]<vector[j])
      intercambiar(vector[j-1],vector[j])
    fin_si
  fin_para
fin_para
```

Realizar el programa inicializando el vector al declararlo en C++ con una longitud de 20 componentes. La lógica del algoritmo de la burbuja debe ser expresamente implementada en ensamblador.

ANEXO. TIPOS DE DATOS EN C/C++

La ocupación en memoria y el rango de valores de los tipos de datos más comunes en C++ se muestran en la siguiente tabla.

Tipo de Datos	Ocupación en memoria	Rango
unsigned char	8 bits	0 a 255
char	8 bits	-128 a 127
short int	16 bits	-32,768 a 32,767
unsigned int	32 bits	0 a 4,294,967,295
int	32 bits	-2,147,483,648 a 2,147,483,647
unsigned long	32 bits	0 a 4,294,967,295
enum	16 bits	-2,147,483,648 a 2,147,483,647
long	32 bits	-2,147,483,648 a 2,147,483,647
float	32 bits	3.4 x 10 ⁻³⁸ a 3.4 x 10 ⁺³⁸ (6 decimales)
double	64 bits	1.7 x 10 ⁻³⁰⁸ a 1.7 x 10 ⁺³⁰⁸ (15 decimales)