

Tampico, Tamaulipas a 8 de **Diciembre** de 2022



VERDAD, BELLEZA, PROBIDAD

UAT



Facultad de Ingeniería
Arturo Narro Siller

Proyecto Integrador

Equipo 1

Nombres: Gonzalez Saldivar Luis Roberto
Martinez Reyes Fernando
Profesor: Dr. García Ruiz Alejandro Humberto
Asignatura: Programación de Microprocesadores
8vo. Semestre – Grupo “I”
2022-3

Proyecto Integrador Tabla de Multiplicar

Descripción: Realizar un programa híbrido entre ensamblador, C y C#, el cual sea capaz de calcular una tabla de multiplicar dependiendo cual número ingrese el usuario, además de cuantas veces el usuario lo desee, debe contar con interfaz gráfica.

Introducción:

Para realizar el programa se debe tener conocimiento de lo que es la programación híbrida

Programación Híbrida

La programación híbrida proporciona un mecanismo por medio del cual podemos aprovechar las ventajas del lenguaje ensamblador y los lenguajes de alto nivel, todo esto con el fin escribir programas más rápidos y eficientes.

Al trabajar con un lenguaje de alto nivel, en ocasiones nos encontramos con el problema de que necesitamos que haga determinada función o trabajo, pero desafortunadamente ésta solo existe en otro lenguaje que no es el que necesitamos utilizar, o simplemente, no encontramos esa función en ningún lenguaje de alto nivel.

Ventajas de la Programación Híbrida

- Para mejorar la escalabilidad
- Cuando muchas tareas producen desbalanceo
- Aplicaciones que combinan paralelismo de grano grueso y fino
- Reducción del tiempo de desarrollo de código
- Cuando el número de procesos MPI es fijo
- En caso de mezcla de paralelismo funcional y de datos

En este momento el lenguaje ensamblador constituye una herramienta no solo eficaz, sino simple para producir un parche para el compilador de nuestro lenguaje preferido.

Tal vez el mayor problema con el que nos enfrentemos sea el de cómo conectar ambos programas (el de alto y el de bajo niveles) y cómo pasar variables de un programa al otro. Para conseguir nuestro objetivo se utilizan pseudo-operadores, es decir, instrucciones que aparecen en el código fuente del ensamblador pero que no generan ninguna instrucción de máquina, pero proporcionan directivas para que el ensamblador pueda operar con datos, ramificaciones condicionales, generación de listados y con macros durante el proceso de ensamble.

Se requiere conocimiento de las siguientes instrucciones de lenguaje Ensamblador:

MOV:

Propósito: Transferencia de datos entre celdas de memoria, registros y acumulador.

Sintaxis:

MOV Destino,Fuente

Donde Destino es el lugar a donde se moverán los datos y fuente es el lugar donde se encuentran dichos datos.

Los diferentes movimientos de datos permitidos para esta instrucción son:

Destino: memoria. Fuente: acumulador

Destino: acumulador. Fuente: memoria

Destino: registro de segmento. Fuente: memoria/registro

Destino: memoria/registro. Fuente: registro de segmento

Destino: registro. Fuente: registro

Destino: registro. Fuente: memoria

Destino: memoria. Fuente: registro

Destino: registro. Fuente: dato inmediato

Destino: memoria. Fuente: dato inmediato

CALL: La instrucción CALL sirve para hacer llamadas a funciones en ensamblador.

MUL: Multiplicación sin signo

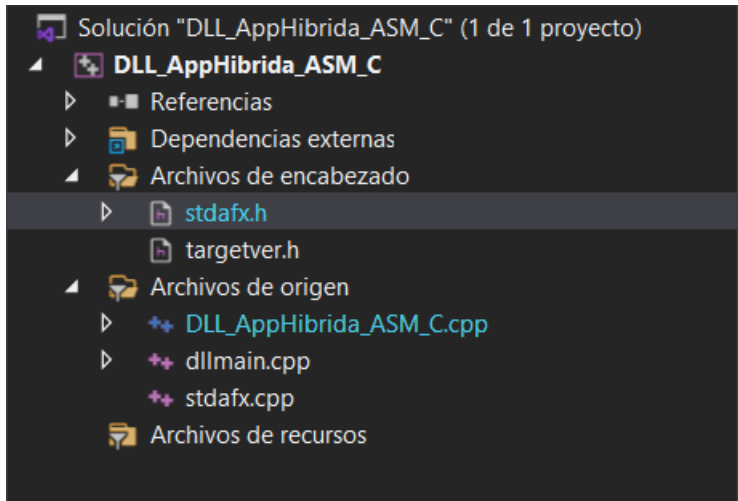
Sintaxis: MUL fuente

El ensamblador asume que el multiplicando será del mismo tamaño que el del multiplicador, por lo tanto, multiplica el valor almacenado en el registro que se le da como operando por el que se encuentre contenido en AH si el multiplicador es de 8 bits o por AX si el multiplicador es de 16 bits.

Cuando se realiza una multiplicación con valores de 8 bits el resultado se almacena en el registro AX y cuando la multiplicación es con valores de 16 bits el resultado se almacena en el registro par DX:AX.

Desarrollo:

En el proyecto de visual studio llamado DLL_AppHibrida_ASM_C.



En el archivo DLL_AppHibrida_ASM_C.cpp nos encontramos los códigos que se realizan de manera hibrida combinando lenguaje C con ensamblador.

Codigo en Ensamblador

```
330     [ ]
331     int __stdcall TablaMulti(int NUM, int MULTI) {
332         int RESU;
333
334         _asm {
335             MOV EAX, NUM
336             MUL MULTI
337
338             MOV RESU, EAX
339         }
340
341         return RESU;
342     }
343
```

En la línea 331 se declara el nombre del método además de ser tipo int con el tipo de llamada estándar (__stdcall) tendrá 2 parámetros de tipo int los cuales se nombraron como “NUM” y “MULTI”.

En la línea 332 se declara una variable int llamada Resu en lenguaje C.

En la línea 334 tenemos la instrucción `_asm` la cual nos permite escribir instrucciones en lenguaje ensamblador mientras se escriban dentro de los corchetes de la instrucción.

En la línea 335 se realiza una instrucción `Mov` la cual moverá la información almacenada en `Num`, guardándola en el registro de 32 bits `Eax`.

En la línea 336 se realiza la instrucción `Mul` se multiplicará el valor de `Multi` con el registro `Eax`, guardando el resultado de la multiplicación en el mismo registro `Eax`.

En la línea 338 se realiza una instrucción `Mov` la cual moverá la información almacenada en el registro `Eax`, guardándola en `Resu`.

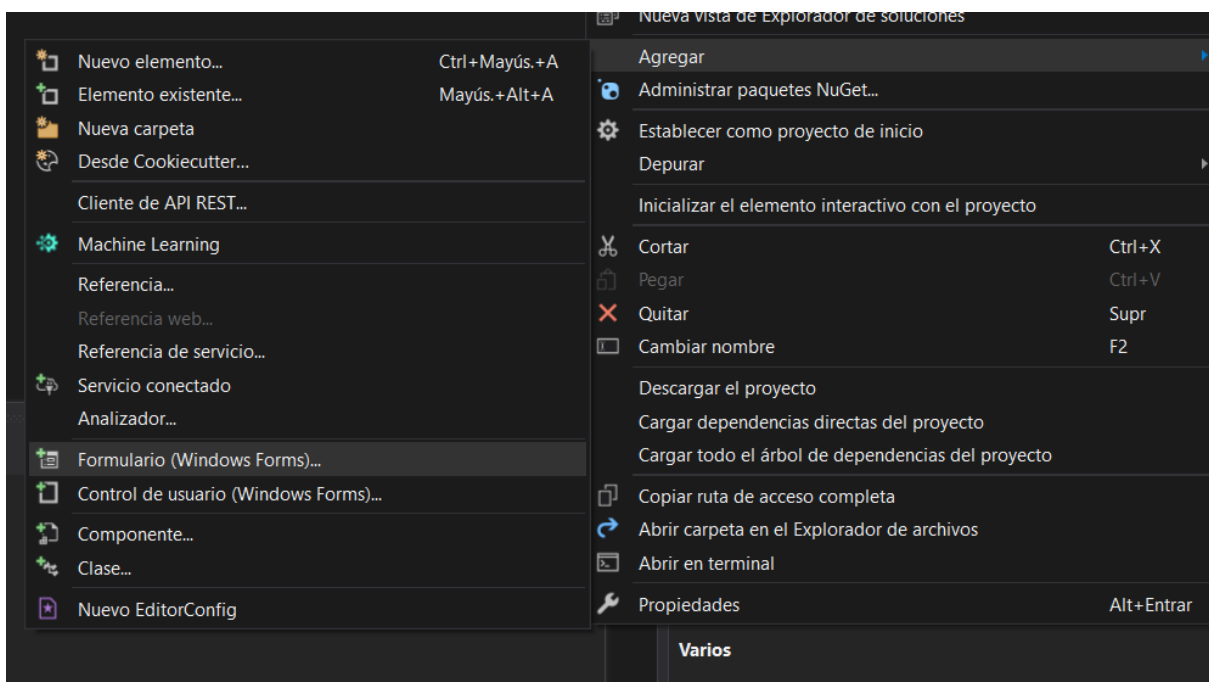
En la línea 339 se cierra la instrucción `_asm` y por consiguiente se terminará el método con lenguaje C.

En la línea 341 se realiza el retorno de la variable local `Resu`, para que sea el valor que nos devuelva al momento de llamar el método en un programa.

Después en el archivo `stdafx.h` en el cual se exportará el método del problema que creamos anteriormente con todos sus parámetros de la siguiente manera:

```
extern "C" __declspec(dllexport) int __stdcall TablaMulti(int NUM, int MULTI);
```

Posteriormente en el proyecto de visual studio `WindowFormAppConsumoDLL` para consultar el diseño del programa, además de poder ejecutarlo.



Se creo un Formulario de Windows para crear el diseño del programa y programar en C#

Diseño del Programa



The image shows a Windows application window titled "Proyecto_Final". The window has a light blue background and a title bar with standard Windows controls (minimize, maximize, close). The main content area contains the following elements:

- A large, bold, black title "TABLA DE MULTIPLICAR" centered at the top.
- A label "INGRESA EL NUMERO A MULTIPLICAR:" followed by a white text input box.
- A label "CUANTAS VECES DESEA MULTIPLICARLO:" followed by a white text input box.
- A button with the text "OBTENER RESULTADO" in the center.
- A large, empty white rectangular area at the bottom, intended for displaying the multiplication results.

Para el diseño del programa necesitamos de labels para expresarle al usuario que datos necesita ingresar en las diferentes casillas, para las casillas de texto se usaron Text Box y cuando el usuario llene las casillas correspondientes tendrá la posibilidad de obtener la tabla de multiplicar en un Text Box considerablemente mas grande para poder apreciar a detalle los resultados obtenidos, para obtener los resultados se usó un button.

Codigo en C#

```
7 using System.Text;
8 using System.Runtime.InteropServices;
9 using System.Threading.Tasks;
10 using System.Windows.Forms;
11
12 namespace WindowFormAppConsumoDLL
13 {
14     3 referencias
15     public partial class Proyecto_Final : Form
16     {
17         [DllImport(@"C:\Users\Lukio\source\repos\PM_2022\Unidad 5\PM_0_U5_ProgHibrida\DLL_AppHibrida_ASM_C\Debug\DLL_AppHibrida_ASM_C.dll")]
18         1 referencia
19         static extern int TablaMulti(int NUM, int MULTI);
```

Se importó System.Runtime.InteropServices para usar la ruta de exportación DLL, además de indicarla en el fragmento de código en la línea 17, en la línea 18 se escribe el método que se exportó en nuestro caso fue TablaMulti y se den marcar los parámetros correctamente.

```
39     1 referencia
40     private void btn_ExecuteTabla_Click(object sender, EventArgs e)
41     {
42         int Numero = Convert.ToInt32(txt_num.Text);
43         int Multiplicar = Convert.ToInt32(txt_multi.Text);
44         String Resultados = "";
45         for (int i = 0; i <= Multiplicar; i++)
46         {
47             Resultados += Numero + " x " + i + " = " + TablaMulti(Numero, i).ToString() + System.Environment.NewLine ;
48         }
49         list_Tabla.Text = Resultados;
50     }
51 }
```

Para que el programa funcione fue necesario crearle un método al botón y ahí se ingreso el código.

En la línea 41 se crea la variable local Numero de tipo int, a la cual se le asignara el valor de nuestro Text Box llamado txt_num convirtiendo el texto a int con la instrucción Convert.ToInt32.

En la línea 42 se crea la variable local Multiplicar de tipo int, a la cual se le asignara el valor de nuestro Text Box llamado txt_multi convirtiendo el texto a int con la instrucción Convert.ToInt32.

En la línea 43 se crea la variable local Resultado de tipo String, a la cual se le asigna una cadena vacia.

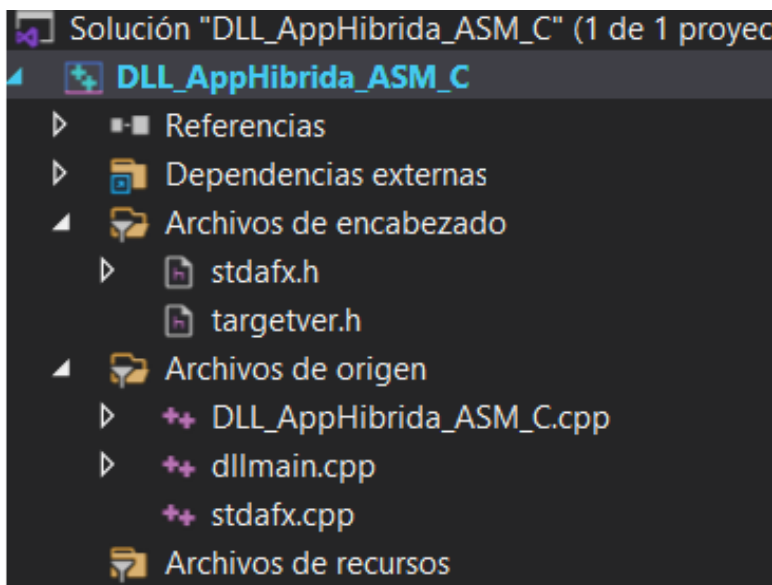
En la línea 45 se crea un for instanciando la variable *i* con valor 0, repitiendo el ciclo siempre y cuando *i* sea menor que Multiplicar incrementando *i* en uno cada que se terminen las instrucciones del ciclo.

En la línea 47 a Resultado se le asigna la suma de Resultado + la variable local (muestra el número que pidió el usuario que se multiplicara)Numero + “x” (para ver estéticamente cuales valores se multiplicaran) + *i* (el número de iteracion el que se encuentra) + “=” (Apreciar la igualdad de la ecuación) + TablaMulti(Numero, *i*).ToString() (con esta instrucción se llama el método tablaMulti exportado en el DLL y se pone como parámetro el número que desea multiplicar y el número de iteracion en que se encuentra el ciclo en ese momento) + System.Environment.NewLine (con esta instrucción se nos permite dar un salto de línea en c#).

En la línea 49 Después de que se termine el ciclo y se guarden los resultados es momento de imprimirlos en el Text Box, para esto es necesario poner el nombre del Text Box y usar la instrucción. Text y asignarle el valor de la variable Resultado, para que de esta manera se aprecie de una mejor manera los resultados.

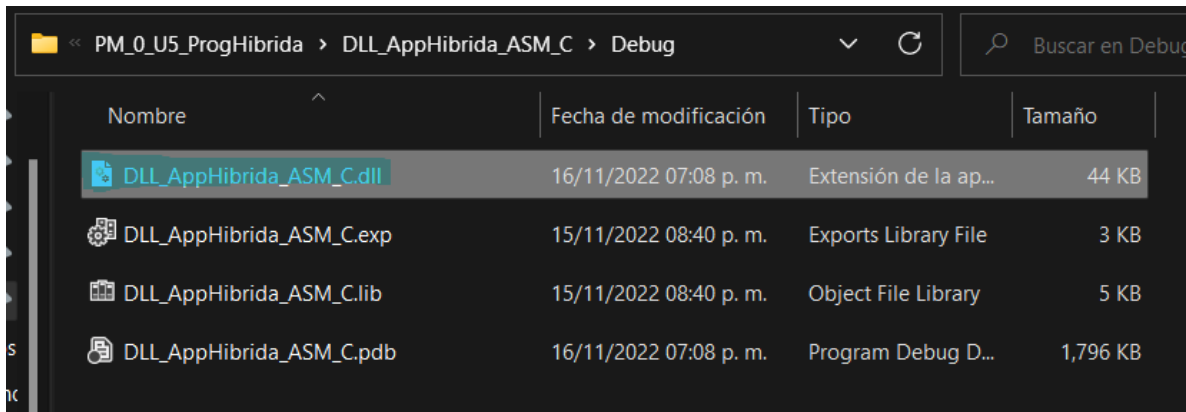
Obtener Ruta DLL

Para obtener la ruta que necesitamos tendremos que volver al proyecto DLL_AppHibrida y seleccionamos el proyecto y damos click derecho y seleccionamos la opción “Limpiar” Posteriormente repetimos el proceso pero ahora, seleccionamos la opción “Compilar”

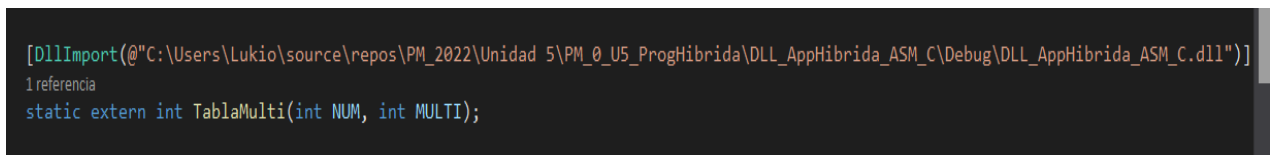


```
1>dllmain.cpp
1>Generando código...
1> Creando biblioteca C:\Users\Lukio\source\repos\PM_2022\Unidad 5\PM_0_U5_ProgHibrida\DLL_AppHibrida_ASM_C\Debug\DLL_AppHibr
1>DLL_AppHibrida_ASM_C.vcxproj -> C:\Users\Lukio\source\repos\PM_2022\Unidad 5\PM_0_U5_ProgHibrida\DLL_AppHibrida_ASM_C\Debug\DLL_AppHibr
===== Compilar: 1 correctos, 0 incorrectos, 0 actualizados, 0 omitidos =====
```


Se tiene que copiar la ruta marcada en la imagen, y pegarla en el explorador de archivos para tener acceso al DLL.

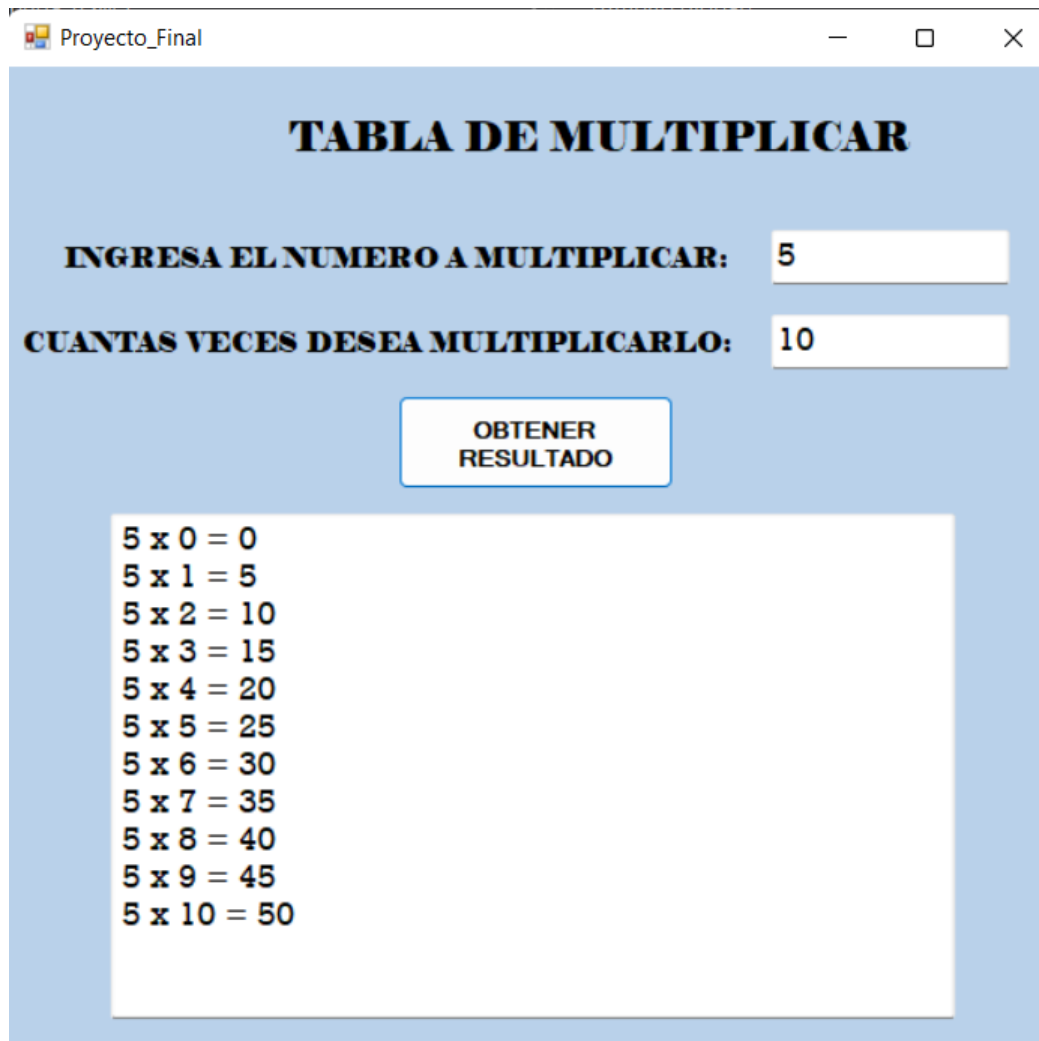


Seleccionamos archivo .dll y damos click derecho, seleccionamos la opción de copiar ruta de archivo.



Y de esta forma se obtiene la ruta que pegaremos en el código del programa.

Resultados:



The screenshot shows a web browser window titled 'Proyecto_Final'. The main heading is 'TABLA DE MULTIPLICAR'. Below it, there are two input fields: 'INGRESA EL NUMERO A MULTIPLICAR:' with the value '5' and 'CUANTAS VECES DESEA MULTIPLICARLO:' with the value '10'. A button labeled 'OBTENER RESULTADO' is positioned below the input fields. The results are displayed in a white box, showing the multiplication table for 5 multiplied by numbers from 0 to 10.

5 x 0 =	0
5 x 1 =	5
5 x 2 =	10
5 x 3 =	15
5 x 4 =	20
5 x 5 =	25
5 x 6 =	30
5 x 7 =	35
5 x 8 =	40
5 x 9 =	45
5 x 10 =	50

Conclusiones

En conclusión, acerca de este proyecto, al realizar un programa en diferentes lenguajes de programación al mismo tiempo aprendimos a usar la programación híbrida aplicando todos los beneficios de los lenguajes de programación tanto de alto como de bajo nivel, creando un programa que es capaz de realizar las tareas designadas de una manera mas eficiente y eficaz.