
	UNIVERSIDAD AUTONOMA TOMAS FRIAS FACULTAD DE INGENIERIA CARRERA DE INGENIERIA DE SISTEMAS	Arquitectura de computadoras (SIS-522)	
DOCENTE: Ing. Gustavo A. Puita Choque	AUXILIAR :Univ. Aldrin Roger Perez Miranda	PRACTICA Nº 9	INICIAL DEL APELLIDO PATERNO:
ESTUDIANTE : Univ. Rodrigo Mauricio Ramos Carvajal	FECHA DE ENTREGA:		R

1) ¿Qué es el 'stack' en el contexto del lenguaje ensamblador y cómo se utiliza?

El 'stack' o pila es una estructura de datos fundamental en el contexto del lenguaje ensamblador, utilizada para almacenar información de manera temporal. Se maneja típicamente con los registros de puntero de pila, como el registro **SP** (Stack Pointer) y el registro **BP** (Base Pointer) en arquitecturas x86. El stack sigue un esquema LIFO (Last In, First Out), donde el último elemento en ser agregado es el primero en ser retirado.

Uso del Stack

1. **Almacenamiento Temporal:** El stack se utiliza para almacenar datos temporales, como variables locales y valores intermedios de cálculos.
2. **Gestión de Llamadas a Subrutinas:** Durante las llamadas a subrutinas (funciones), el stack guarda la dirección de retorno (para saber dónde continuar la ejecución después de que la subrutina termine) y los parámetros de la subrutina.
3. **Preservación de Registros:** Antes de usar registros para operaciones dentro de una subrutina, sus valores actuales se pueden guardar en el stack y restaurar después de la ejecución de la subrutina para no alterar el estado del programa.

2) Describe un escenario práctico donde el uso de ensamblador sería más ventajoso que el uso de un lenguaje de alto nivel

Escenario: Desarrollo de Controladores de Hardware (Drivers)

En el desarrollo de controladores de hardware (drivers), el uso de lenguaje ensamblador puede ser más ventajoso que el uso de un lenguaje de alto nivel por las siguientes razones:

1. **Control Preciso del Hardware:** Los controladores necesitan comunicarse directamente con el hardware, requiriendo un control muy preciso sobre los registros y las operaciones de entrada/salida. El ensamblador permite un acceso directo a estos recursos sin la abstracción que imponen los lenguajes de alto nivel.
2. **Optimización de Rendimiento:** En entornos donde el rendimiento es crítico, como en sistemas embebidos o dispositivos con recursos limitados, el ensamblador permite optimizar al máximo el uso del procesador y la memoria, ajustando las operaciones al nivel más bajo.
3. **Manejo de Interrupciones:** Los controladores de hardware suelen manejar interrupciones, que son señales enviadas por dispositivos para indicar que necesitan atención. La respuesta a interrupciones debe ser rápida y eficiente, algo que el ensamblador puede manejar mejor debido a su bajo nivel de abstracción.

4. **Minimización de Overhead:** Los lenguajes de alto nivel introducen overhead (sobrecarga) debido a la necesidad de gestionar la memoria y otros recursos de manera abstracta. El ensamblador permite escribir código más eficiente, minimizando este overhead.

3) Explique cada línea del siguiente código del lenguaje ensamblador y diga que es lo que se está haciendo

- 1 . • La primera línea inicializa el registro AX con el valor 5.
- 2 . • La segunda línea inicializa el registro BX con el valor 10.
- 3 . • La tercera línea suma el valor de BX al valor de AX y almacena el resultado en AX (AX ahora contiene 15).
- 4 . • La cuarta línea mueve el resultado de AX al registro CX, por lo que CX ahora también contiene 15.

4) Explique detalladamente cómo funcionan los compiladores

Análisis Léxico: Identifica tokens como `begin`, `writeln`, `if`, `else`, `readln`, `x`, `y`, `sqrt`, `abs`, operadores, números y cadenas de texto.

Análisis Sintáctico: Construye un AST donde cada `writeln`, `if-else` y `readln` se representan como nodos y sus operaciones y argumentos como subnodos.

Análisis Semántico: Verifica que `x` y `y` están correctamente declarados y utilizados, y que las operaciones son semánticamente correctas.

Generación de Código Intermedio: Transforma el AST en una representación de tres direcciones o similar, manejando cálculos de raíz cuadrada y condiciones.

Optimización de Código: Simplifica operaciones y elimina redundancias, si existen.

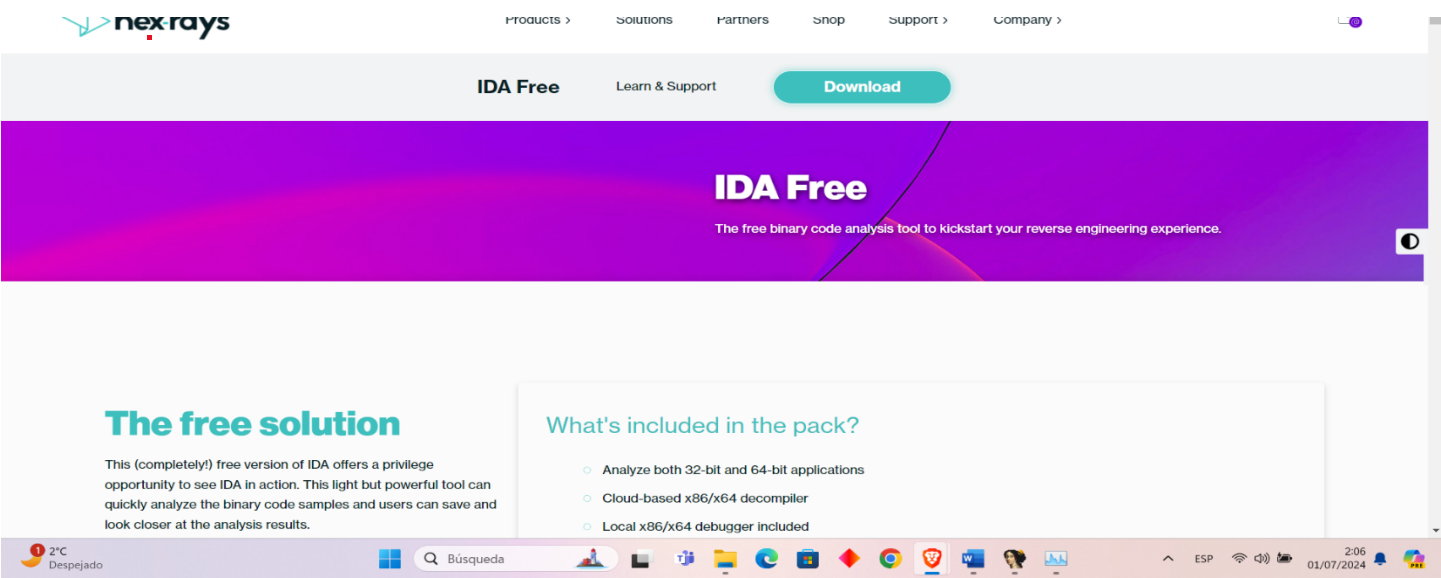
Generación de Código: Convierte el código intermedio en instrucciones de código máquina específicas para la CPU.

Enlazado y Carga: Combina todas las partes del programa y lo prepara para la ejecución.

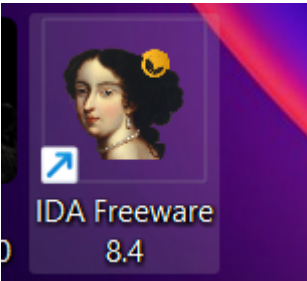
El resultado es un archivo ejecutable (`miPrimerPrograma.exe`) que la computadora puede ejecutar directamente, realizando las operaciones especificadas en el código fuente.

5)

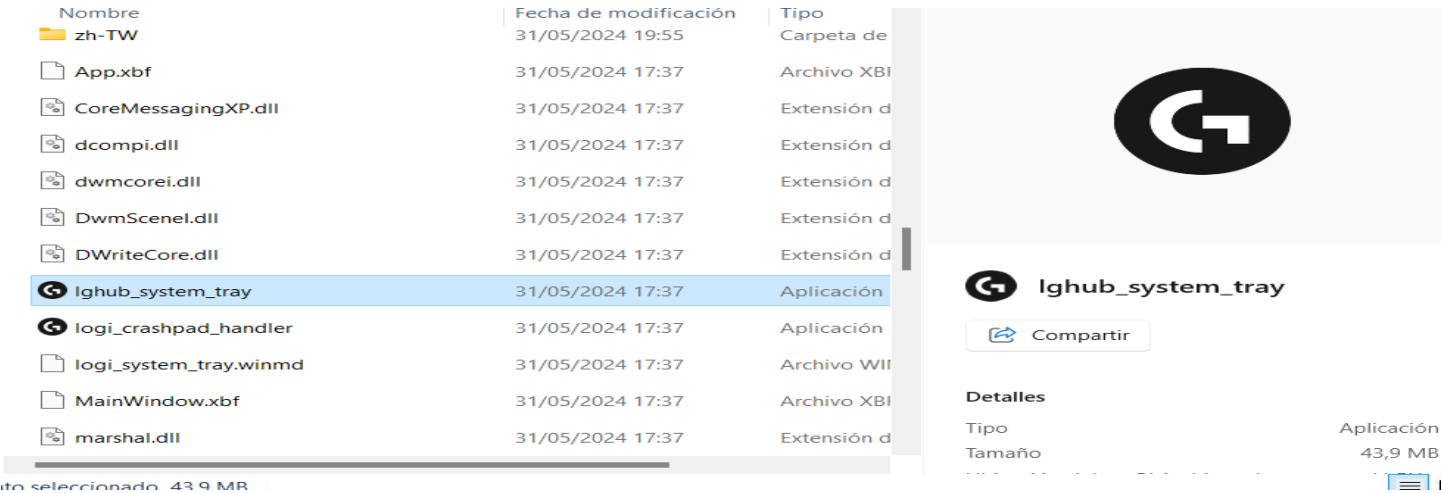
Se ingreso al link para descargar el programa



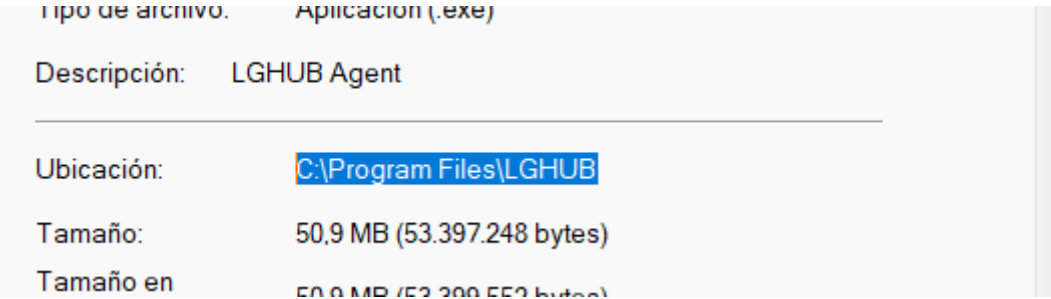
Ya instalado el programa



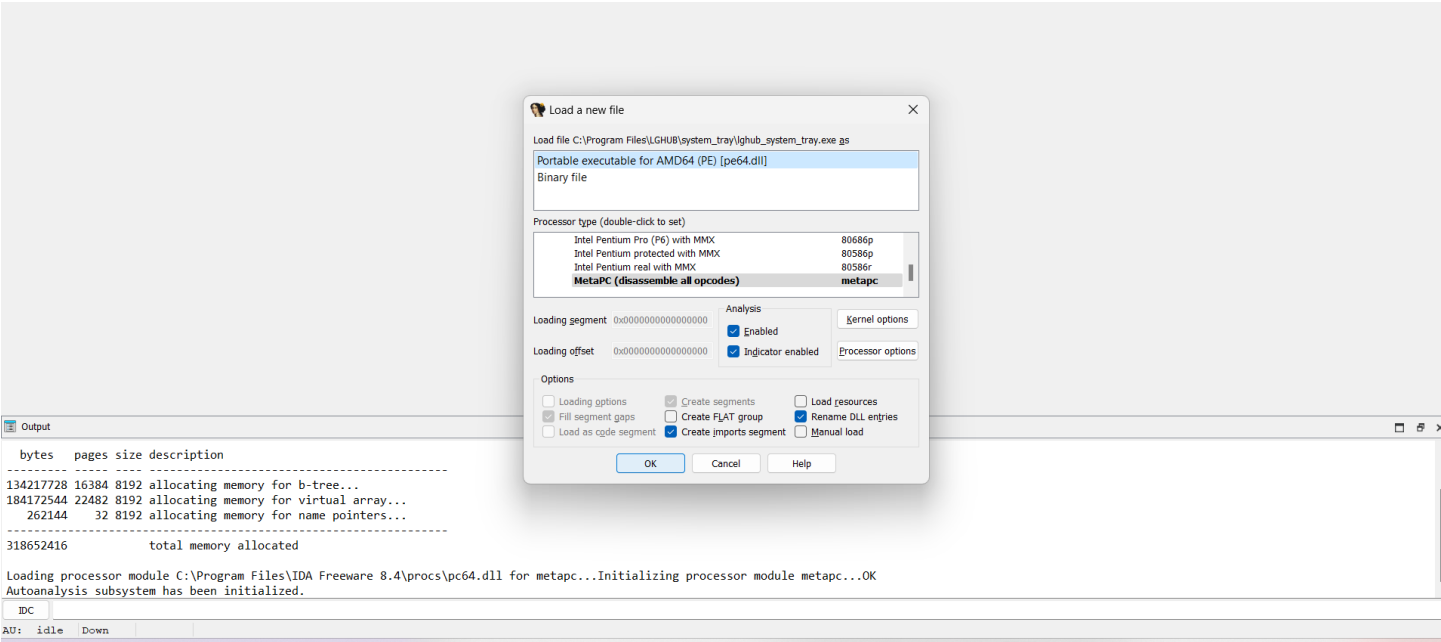
En este paso entre a administrador de tareas y elegi la aplicación de lghub



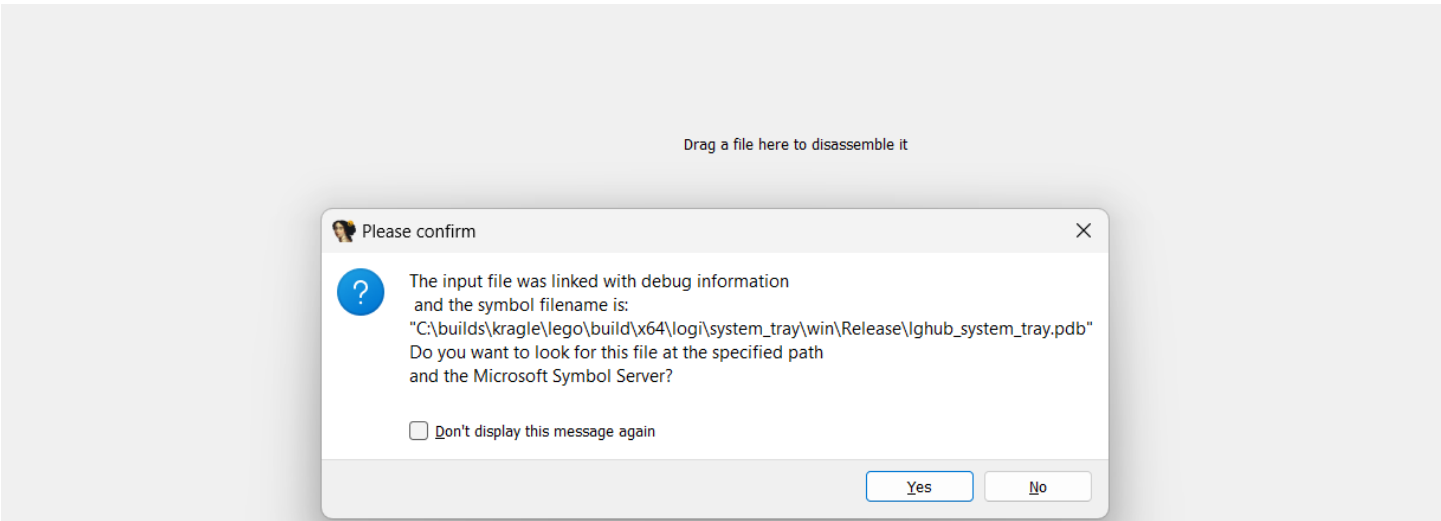
Ya copiada la ruta del archivo



Una vez que coloque en guardar procedi a desensamblar e lservicio en este caso el “lghub” e hice clk en “ok”



Después “no”



```
000000142010000 00000000142000000) ... .. OK
000000142B6B000-0000000142B6C000) ... .. OK

data)...
83 functions...

0000001404DA018-0000000142AA3000) ... .. OK
dll is used for module WS2_32
```

Como se puede ver aquí se tiene como una estructura de tablas

