UNIVERSIDAD AUTÓNOMA "TOMAS FRÍAS" CARRERA DE INGENIERÍA DE SISTEMAS				
Nombre:	Alvaro Rene Condori Quispe			
Materia:	Arquitectura de computadoras (SIS-522)			
Docente: **Auxiliar:**	Ing. Gustavo A. Puita Choque Univ. Aldrin Roger Perez Miranda			N° Práctica
16/06/2024	**Fecha publicación**			9
01/07/2024	**Fecha de entrega**			
Grupo:	**1**	**Sede**	**Potosí**	

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

En el contexto del lenguaje ensamblador, la pila o 'stack' es una estructura de datos que se rige por el principio LIFO (Last In, First Out), lo cual significa que el elemento más reciente en entrar será el primero en salir. Es un área de memoria que se emplea para guardar datos de forma temporal, especialmente en la gestión de subrutinas y el manejo de información local.

PUSH: Instrucción para añadir un valor a la pila. Se decrementa el puntero de la pila (SP - Stack Pointer) y se almacena el valor en la parte superior.

POP: Instrucción para quitar (remove) un valor de la pila. Se extrae el valor de la parte superior de la pila incrementando a continuación el puntero de la pila (SP).

CALL: Indicación para invocar una subrutina. Después de ejecutar la subrutina, se almacena en la pila la dirección de retorno para poder regresar al punto de llamada.

RET: Indicación para salir de una subrutina. Obtén la dirección de retorno almacenada en la pila y dirige el flujo de control hacia esa ubicación.

Además, el stack es utilizado para transmitir parámetros a subrutinas y guardar variables locales en su interior.

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

Cuando se requiere un mayor control sobre el hardware y la eficiencia del código, resulta más beneficioso utilizar el ensamblador. Un caso concreto se puede ver en el ámbito de los sistemas embebidos, como es el caso del firmware que opera un microcontrolador en dispositivos IoT (Internet of Things).

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

MOV AX, 5; Línea 1: Esta línea de código mueve el valor 5 al registro AX la acción que realiza es de AX = 5

MOV BX, 10; Línea 2: Esta línea de código mueve el valor 10 al registro BX la acción que realiza es de BX = 10

ADD AX, BX; Línea 3: Esta línea de código suma el valor en el registro BX al valor en el registro AX y almacena el resultado en AX y la acción que realiza es de AX = AX + BX, y un ejemplo seria del casos anterior ejemplo: AX = 5 + 10 = 15

MOV CX, AX; Línea 4: Esta línea de código mueve el valor en el registro AX al registro CX y la acción que realiza es de CX = AX y como resultado no da CX = 15

4) Explique detalladamente cómo funciona los compiladores.

Un programa que traduce código fuente escrito en un lenguaje de alto nivel, como C, C++ o Java, a un lenguaje de bajo nivel ejecutable por una computadora es conocido como compilador. Este proceso se realiza en varias fases: Este proceso se realiza en varias fases:

Análisis Léxico: El código fuente se transforma por el compilador en una sucesión de tokens, los cuales son conjuntos de caracteres con un significado conjunto (como términos clave, nombres y operadores). El analizador léxico o lexer es el encargado de realizar este paso. Un ejemplo sería descomponer la línea int x = 10; en los siguientes tokens: int, x, =, 10 y;

Análisis Sintáctico: El analizador de sintaxis organiza los tokens en una estructura jerárquica conocida como árbol sintáctico o árbol de análisis, asegurándose de que el código cumple con las reglas gramaticales del lenguaje. Un ejemplo sería que el árbol sintáctico de int x = 10 representa la declaración de int x y la asignación de x = 10.

Análisis Semántico: Durante esta etapa se realiza una verificación para asegurar que el código tiene coherencia en cuanto al significado del lenguaje, validando tanto las operaciones como las declaraciones realizadas. Un ejemplo sería asegurarse de que x, de tipo int, pueda recibir el valor 10.

Generación de Código Intermedio: La representación intermedia (IR) es una traducción del árbol sintáctico que no depende de la máquina. Un ejemplo sería que el código int x = 10; se transformara en Three-Address Code como t1 = 10.

Optimización de Código Intermedio: Se llevan a cabo mejoras en el código intermedio con el objetivo de incrementar la eficiencia del programa, tales como simplificar las expresiones y eliminar partes inútiles del código.

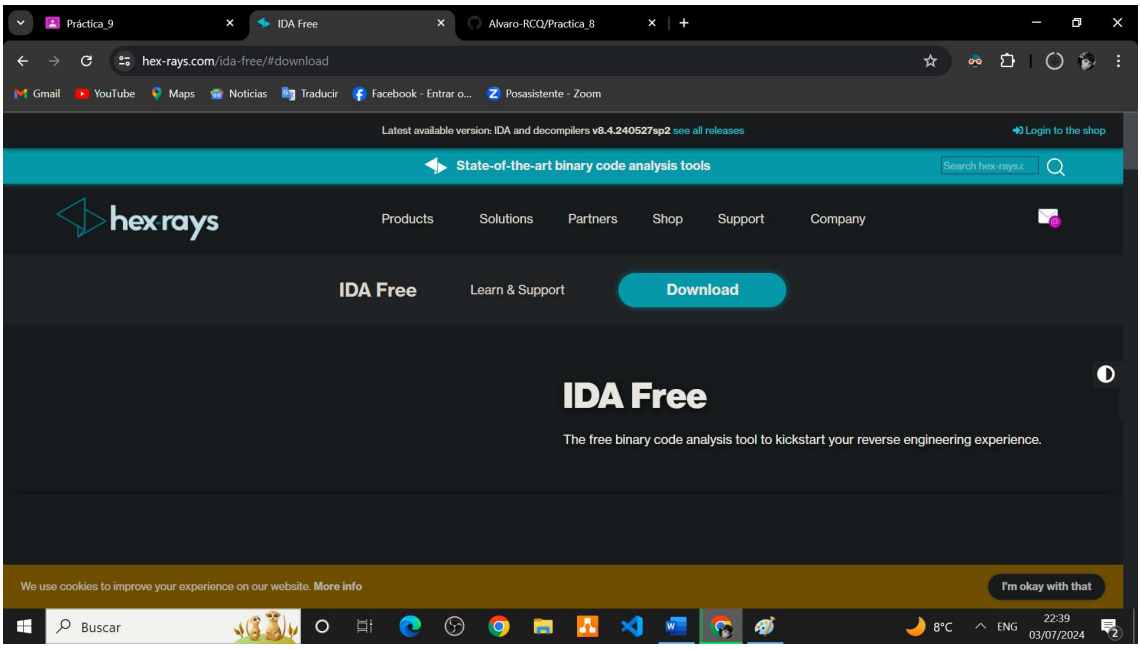
Generación de Código de Máquina: Se traduce a código de máquina específico para el hardware, una vez que se optimiza el código intermedio. Un ejemplo es cuando las operaciones aritméticas se transforman en instrucciones de ensamblador que son específicas para la CPU.

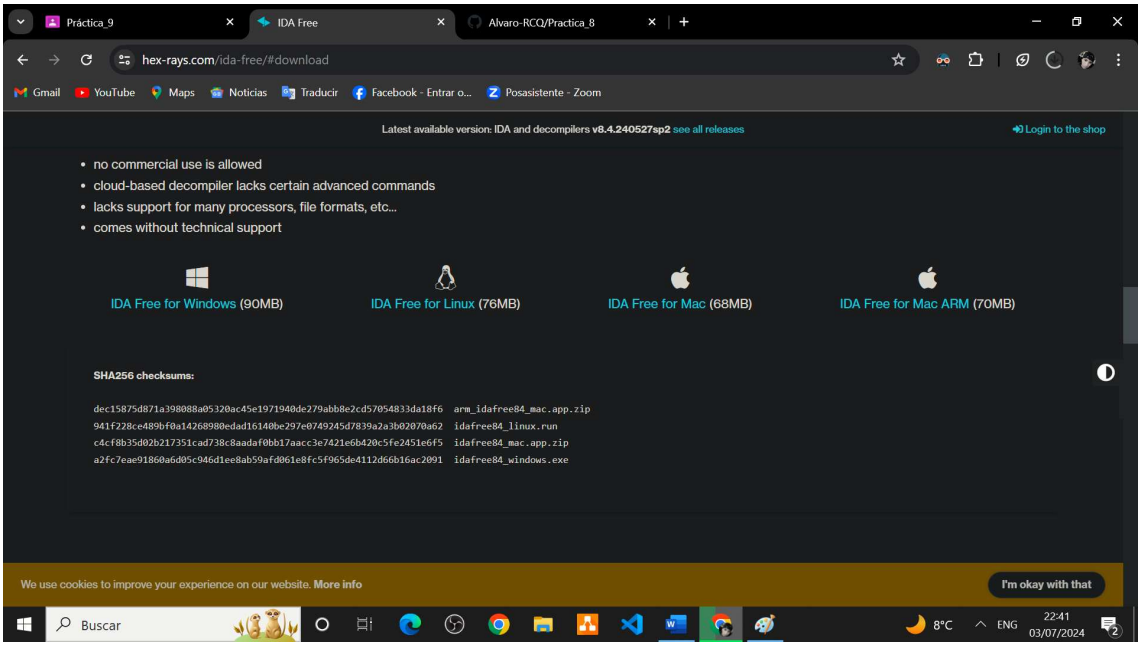
Optimización de Código de Máquina: Se aplican técnicas de optimización que aprovechan las características del hardware, ajustando el uso de registros para minimizar accesos a memoria.

Enlazado: Finalmente, el enlazador combina diferentes módulos de código y bibliotecas en un solo archivo ejecutable, resolviendo referencias entre módulos y ajustando direcciones de memoria.

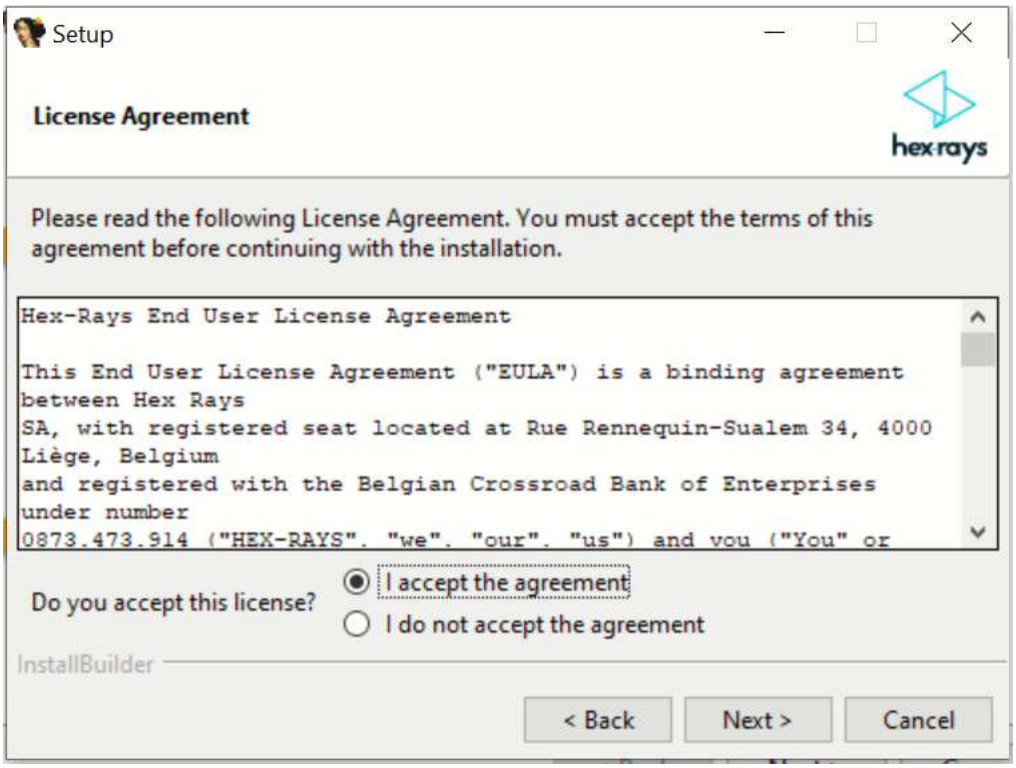
5) Realizar capturas de pantalla del siguiente procedimiento.

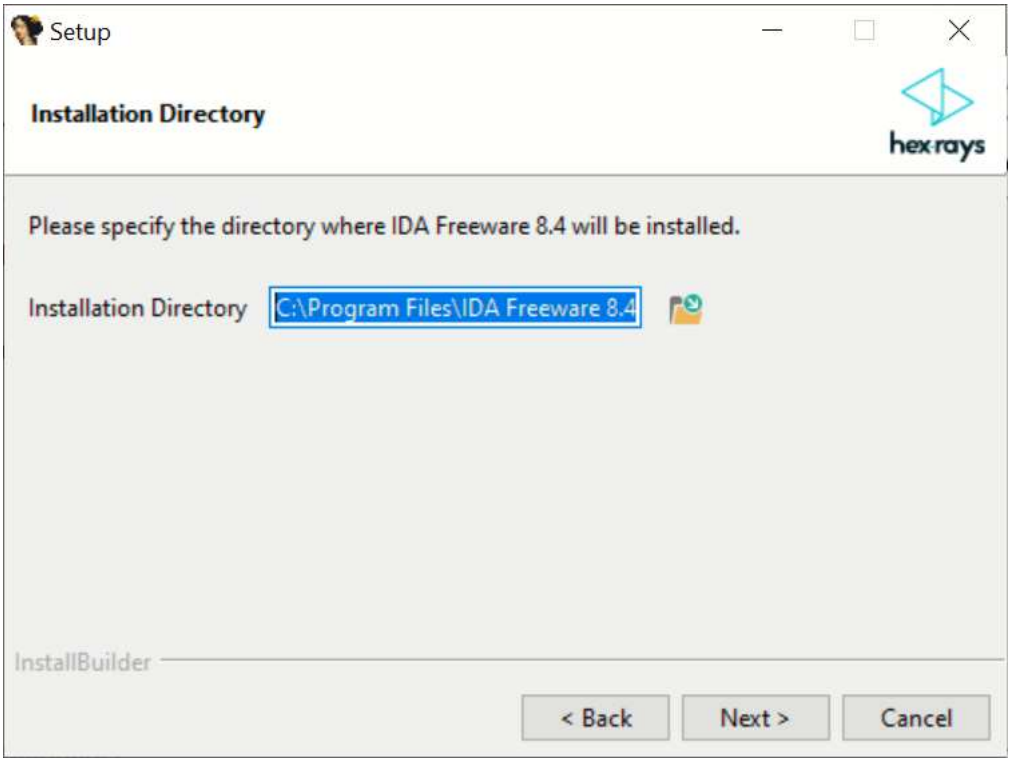
Paso 1: Como paso numero uno entramos al link de y empezamos a descargar la IDA Free





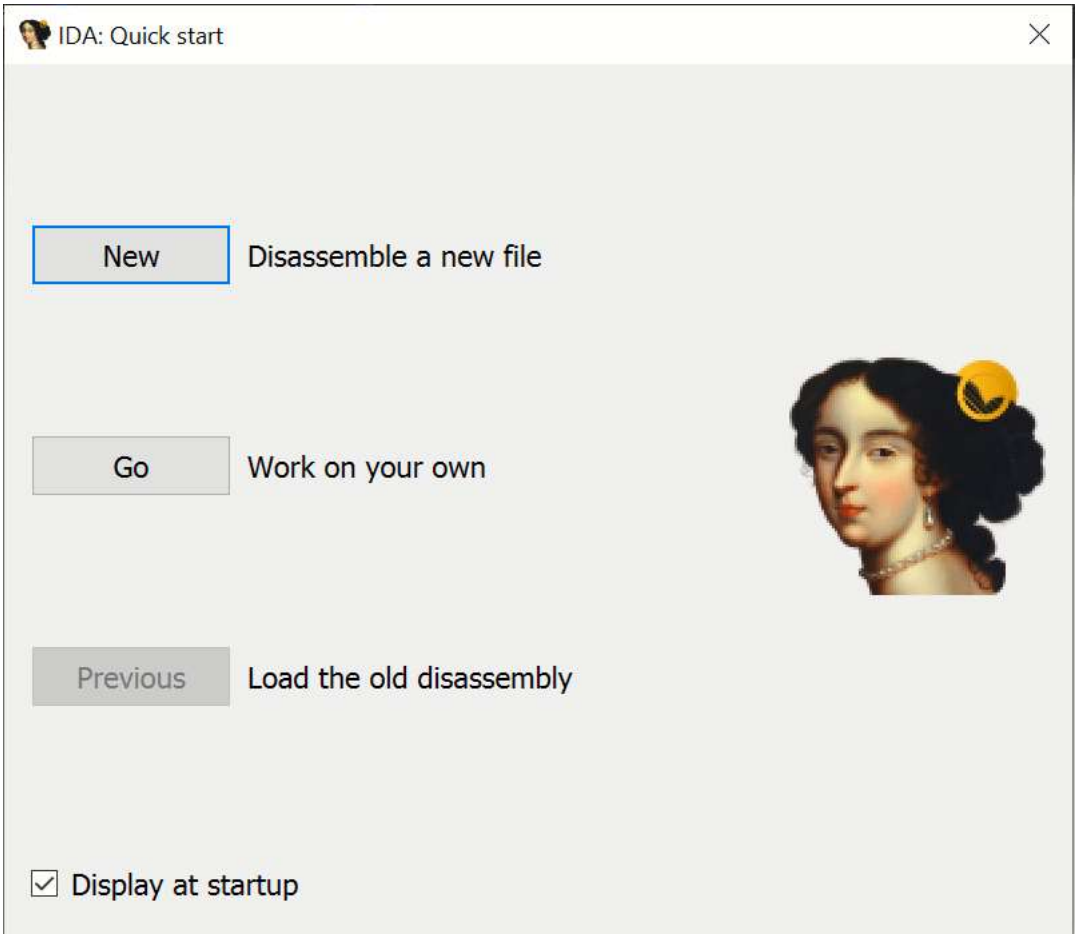
Paso 2: Una ves descargado la herramienta se procede a la instalación de la IDA Free siguiendo los pasos de la guía que se nos proporcionó.


Setup
License Agreement
hex-rays
Please read the following License Agreement. You must accept the terms of this agreement before continuing with the installation.
Hex-Rays End User License Agreement
This End User License Agreement ("EULA") is a binding agreement between Hex Rays
SA, with registered seat located at Rue Rennequin-Sualem 34, 4000 Liège, Belgium
and registered with the Belgian Crossroad Bank of Enterprises under number
0873.473.914 ("HEX-RAYS", "we", "our", "us") and you ("You" or
Do you accept this license?
I accept the agreement
I do not accept the agreement
InstallBuilder
< Back
Next >
Cancel

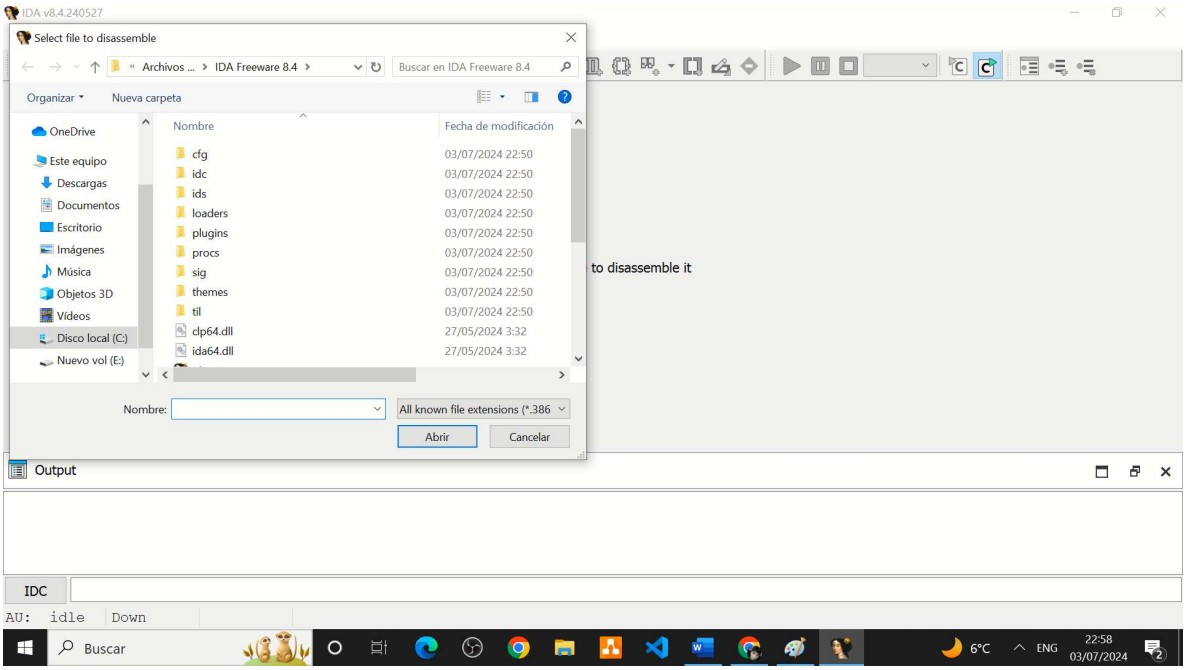

Setup
Installation Directory
hex-rays
Please specify the directory where IDA Freeware 8.4 will be installed.
Installation Directory
C:\Program Files\IDA Freeware 8.4
InstallBuilder
< Back
Next >
Cancel

Una vez descargado e instalado deberán abrir el ejecutable .exe

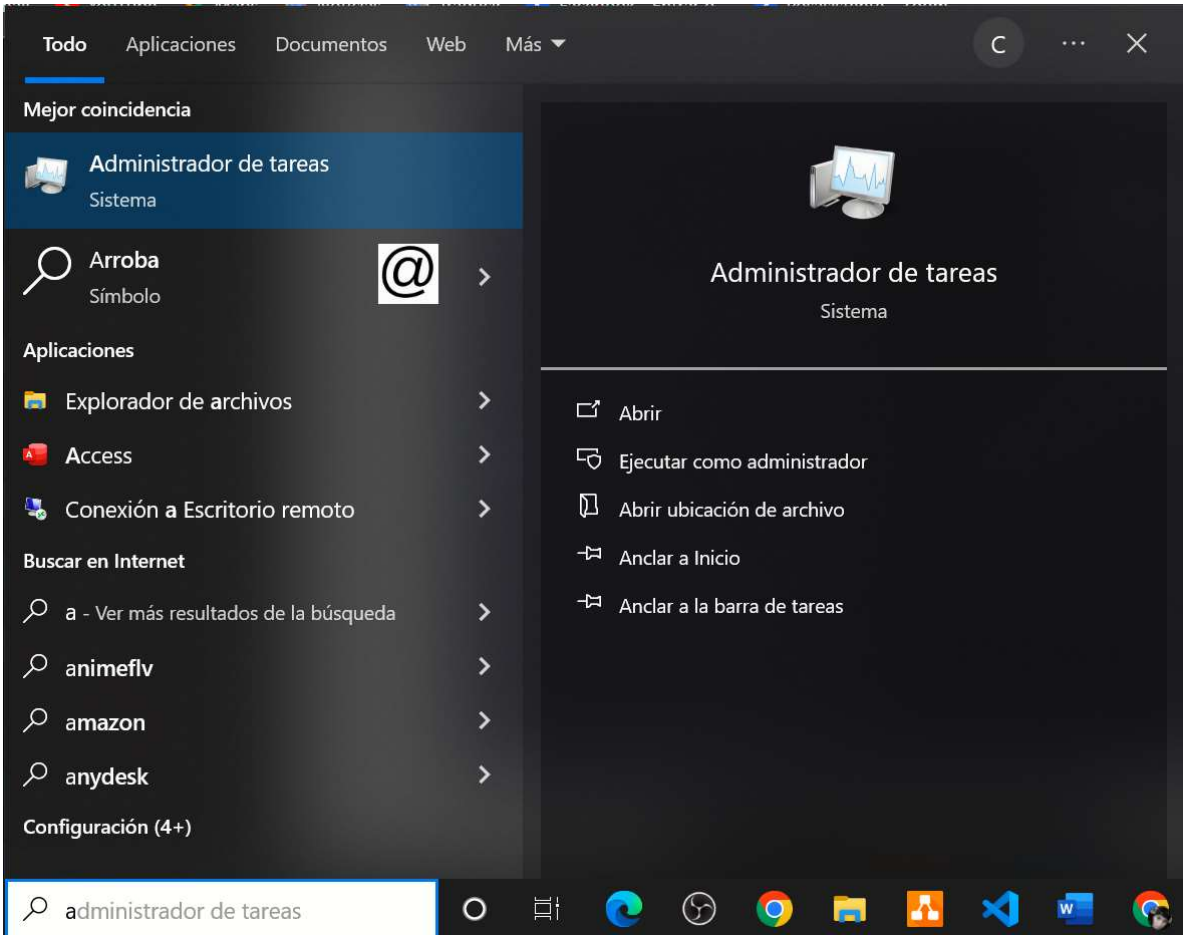
Paso 3: Procederemos a abrir un servicio en Windows.



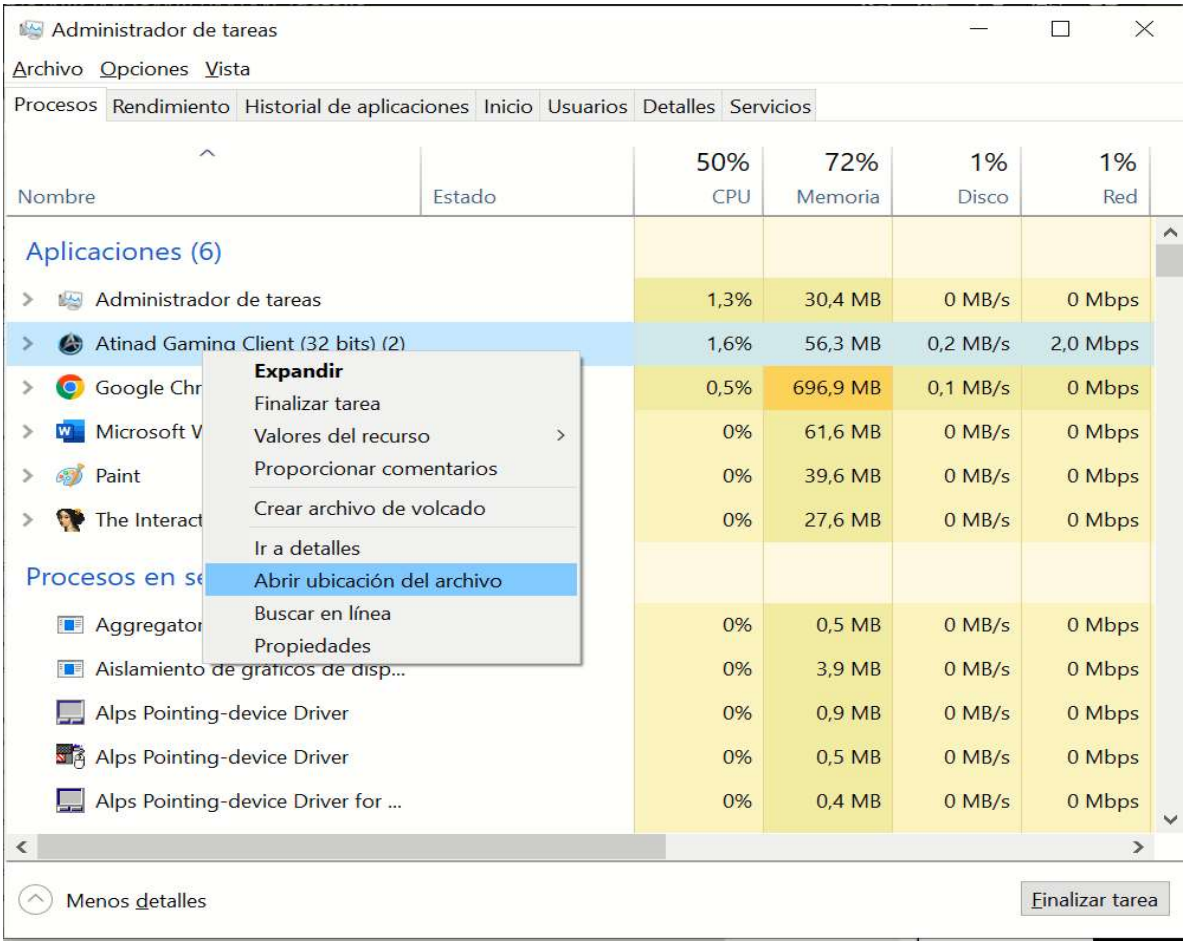
Como nos indica la guía presionamos en New para poder elegir un servicio que se ejecuta e tiempo real.

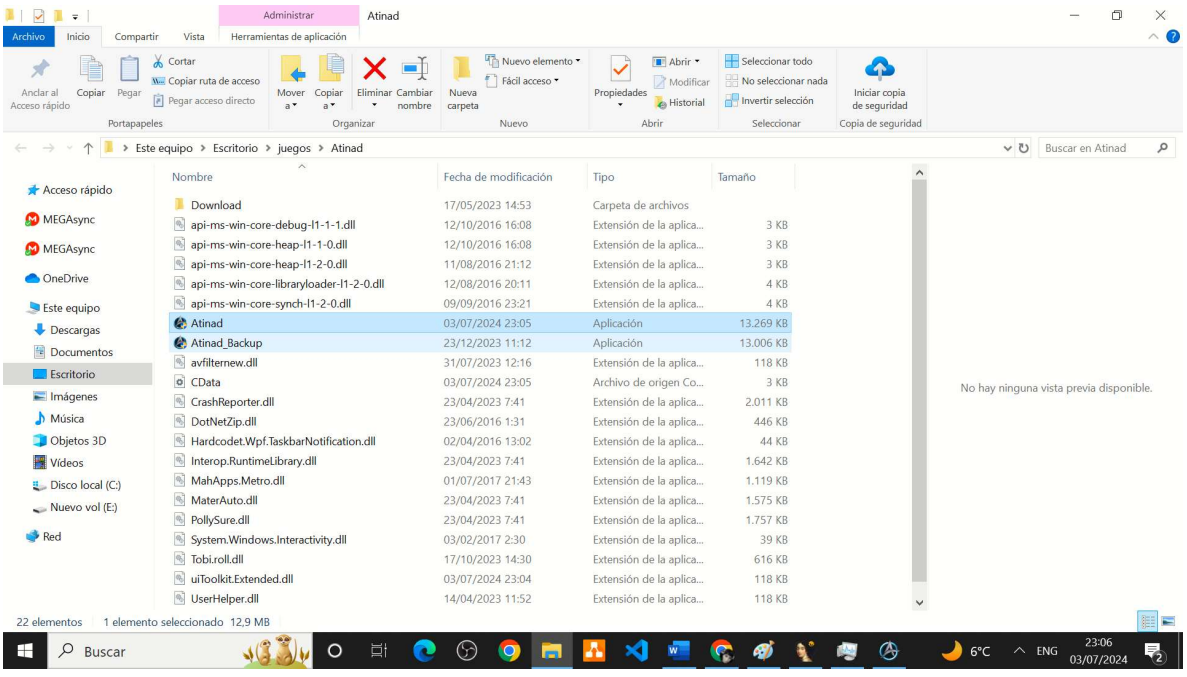


Ahora deberá seleccionar algún servicio de su administrador de tareas, primeramente, vamos a abrir el administrador de tareas



Ahora, en la pestaña de procesos, deberá buscar cualquier servicio que se esté ejecutando en tiempo real. Luego, haga clic izquierdo sobre el servicio que le interese ver el código ensamblador. Después, haga clic derecho y seleccione “Abrir ubicación del archivo”.

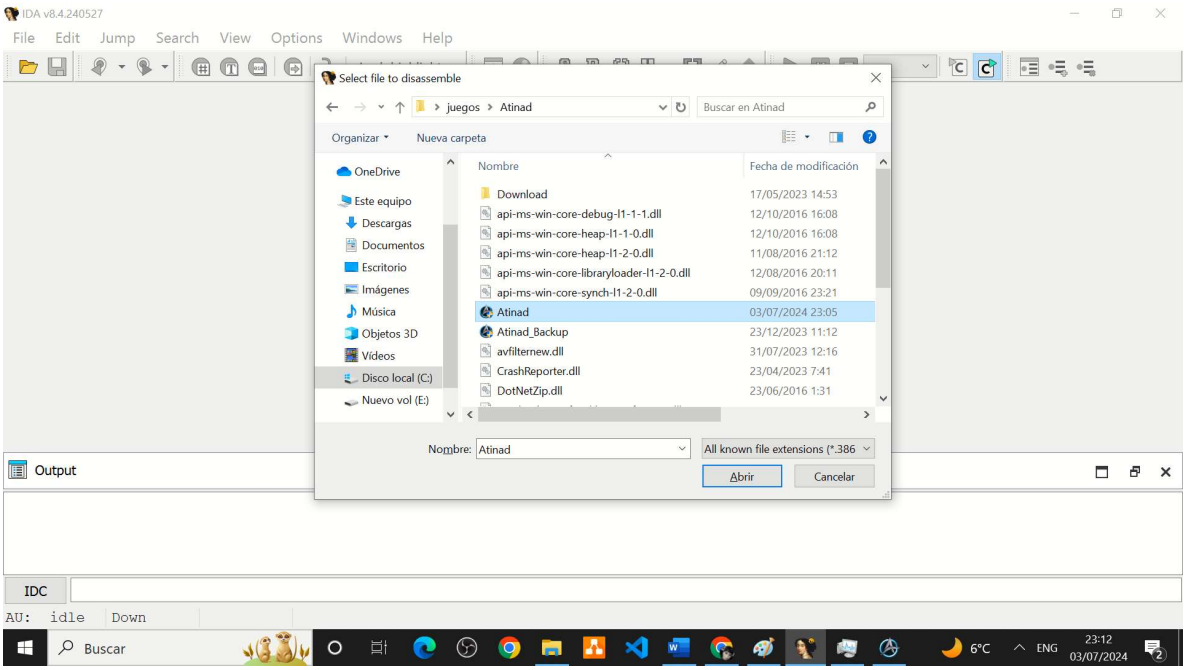




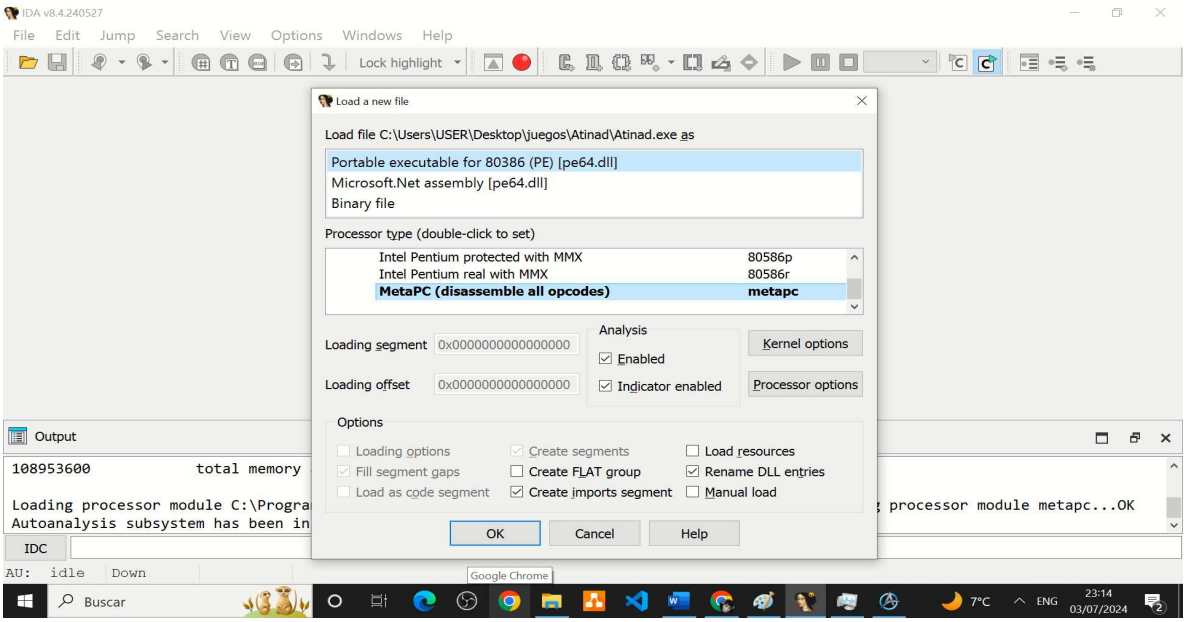
Una vez hecho esto, se abrirá la ubicación del archivo del servicio.

Y se copiara la ruta de donde está la aplicación en este caso es:

Y una vez copiada la ruta se deberá introducir en la IDE Free donde nos pidió que abriéramos un servicio a analizar.

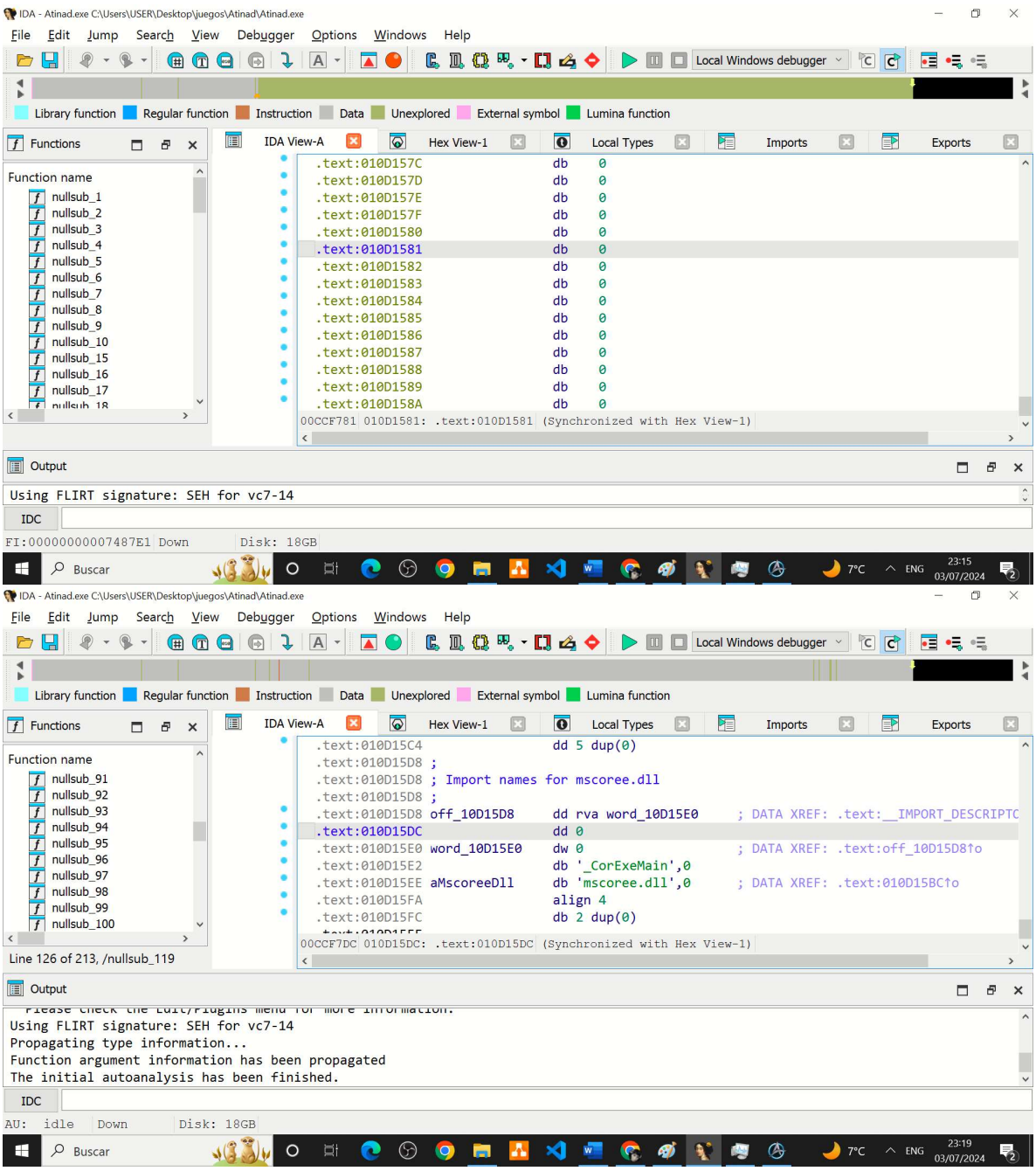


Una vez que seleccionemos la opción de guardar, procederemos a desensamblar el servicio, en este caso, “steam”. El tiempo que tomará este proceso dependerá del tamaño del servicio a analizar. Dejaremos todas las configuraciones por defecto y haremos clic en “OK”.



Paso 4:

Finalmente, se podrá ver código Assembler del servicio que hemos desensamblado



Principalmente se utiliza a IDA Free como programa desensamblador y depurador interactivo para el análisis de código de máquina y la ingeniería inversa de programas binarios. Facilita a los analistas de seguridad, investigadores de malware y desarrolladores de software la exploración y comprensión del código ejecutable en lenguaje ensamblador. Con IDA Free, los usuarios tienen la capacidad de analizar rutinas de código, reconocer funciones, comprender la lógica del programa y buscar posibles vulnerabilidades o comportamientos indeseados en aplicaciones binarias.