


<u>UNIVERSIDAD AUTÓNOMA “TOMAS FRÍAS”</u> <u>CARRERA DE INGENIERÍA DE SISTEMAS</u>				
Materia:	Arquitectura de computadoras (SIS-522)			
Docente:	Ing. Gustavo A. Puita Choque			N° Práctica
Auxiliar:	Univ. Aldrin Roger Perez Miranda			9
Estudiante:	Univ. Lisbeth Cuenca Mamani			
20/11/2024	Fecha publicación			
06/12/2024	Fecha de entrega			
Grupo:	1	Sede	Potosí	

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

Un 'stack' es una estructura de datos que funciona con el principio de LIFO.

Se utiliza para almacenar información temporalmente durante la ejecución de un programa, como variables locales.

- **PUSH:** Inserta un valor. **POP:** Extrae el valor.
- **CALL:** Llama a una subrutina, guardando la dirección de retorno en la pila.
- **RET:** Retorna de una subrutina, recuperando la dirección de retorno de la pila.

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

El uso de ensamblador sería más ventajoso en situaciones donde se requiere un control muy preciso sobre el hardware, como en el desarrollo de controladores de dispositivos, sistemas operativos, y software embebido para dispositivos de tiempo real o microcontroladores.

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

Línea 1: Mueve el valor 5 al registro AX.

Línea 2: Mueve el valor 10 al registro BX.

Línea 3: Suma el valor en BX al valor en AX.

Línea 4: Mueve el valor en AX al registro CX.

```
MOV AX, 5 ; Línea 1
MOV BX, 10 ; Línea 2
ADD AX, BX ; Línea 3
MOV CX, AX ; Línea 4
```

4) Explique detalladamente cómo funcionan los compiladores

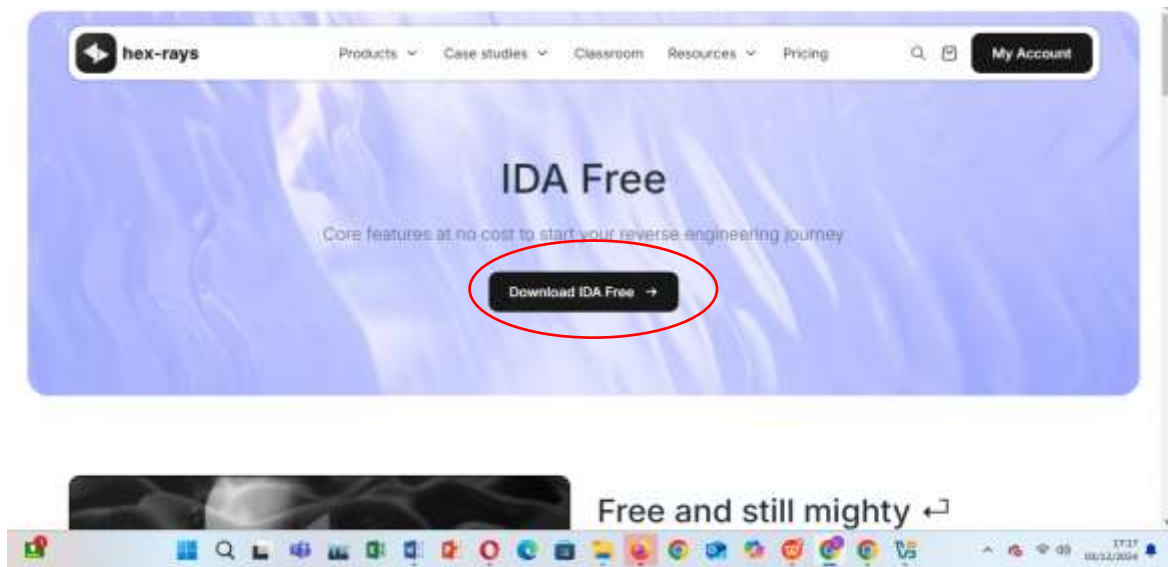
Los compiladores son programas que traducen código fuente escrito en un lenguaje de alto nivel a un lenguaje de bajo nivel o lenguaje máquina que la computadora puede ejecutar directamente.

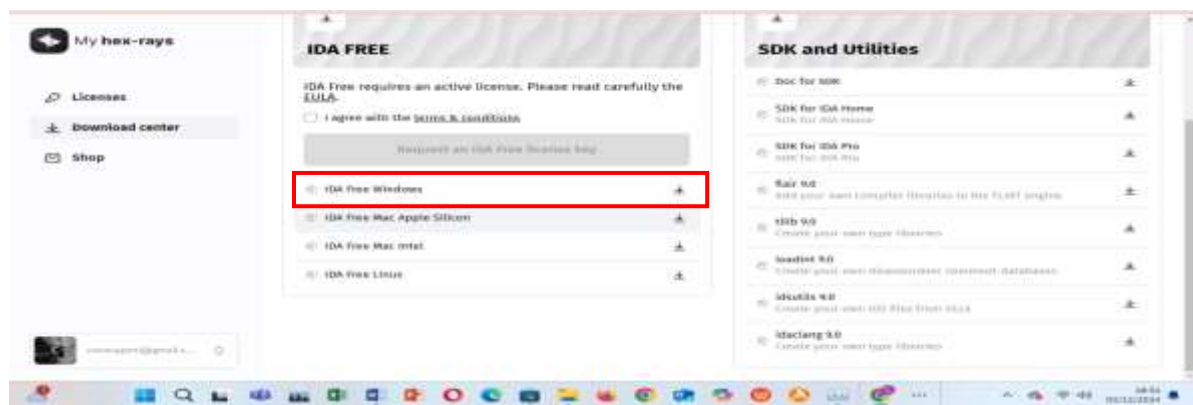
1. Convierte el código fuente en una serie de tokens, que son unidades básicas del lenguaje, como palabras clave, operadores, identificadores y literales.
2. Los tokens se organizan en una estructura que refleja la gramática del lenguaje, usualmente representada como un árbol de sintaxis
3. Verifica la semántica del programa, asegurándose de que las operaciones sean válidas y significativas.
4. Mejora el código intermedio para hacerlo más eficiente, sin cambiar su comportamiento.
5. Convierte el código intermedio optimizado en código máquina específico para la arquitectura del procesador de destino.
6. Traduce el código máquina a un archivo ejecutable y enlaza diferentes módulos del programa, incluyendo bibliotecas y funciones externas.

5) Realizar capturas de pantalla del siguiente procedimiento:

IDA: Es una de las herramientas más conocidas y potentes para el análisis de código binario y desensamblado. En este laboratorio se instalará IDA FREE pero también se tiene la versión de paga IDA PRO

Paso 1: Descargar el software IDA FREE el cual lo podrá a hacer del siguiente enlace: <https://hex-rays.com/ida-free/>



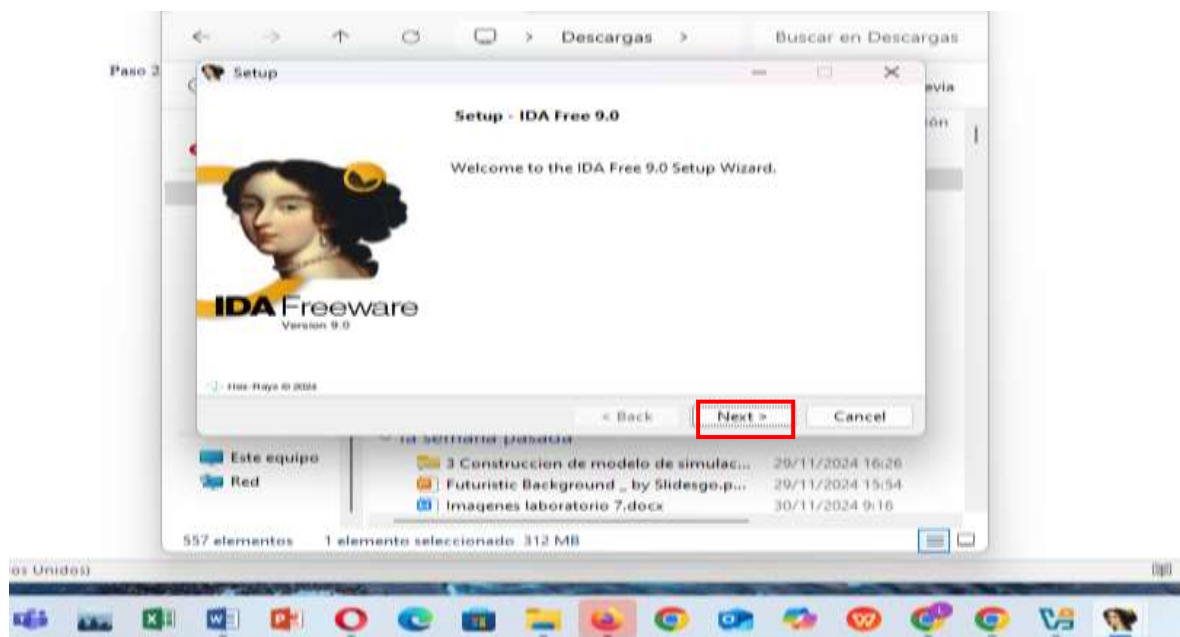


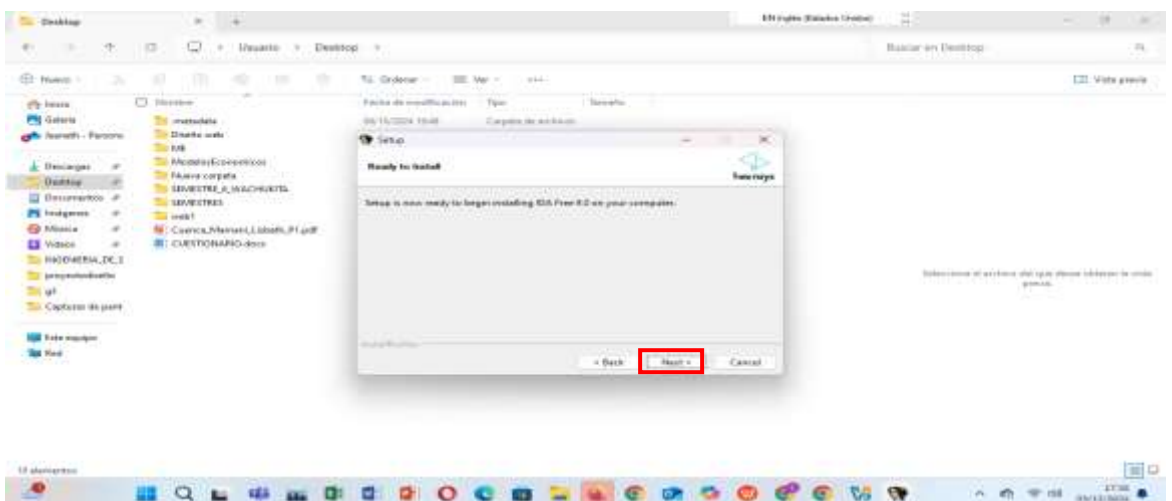
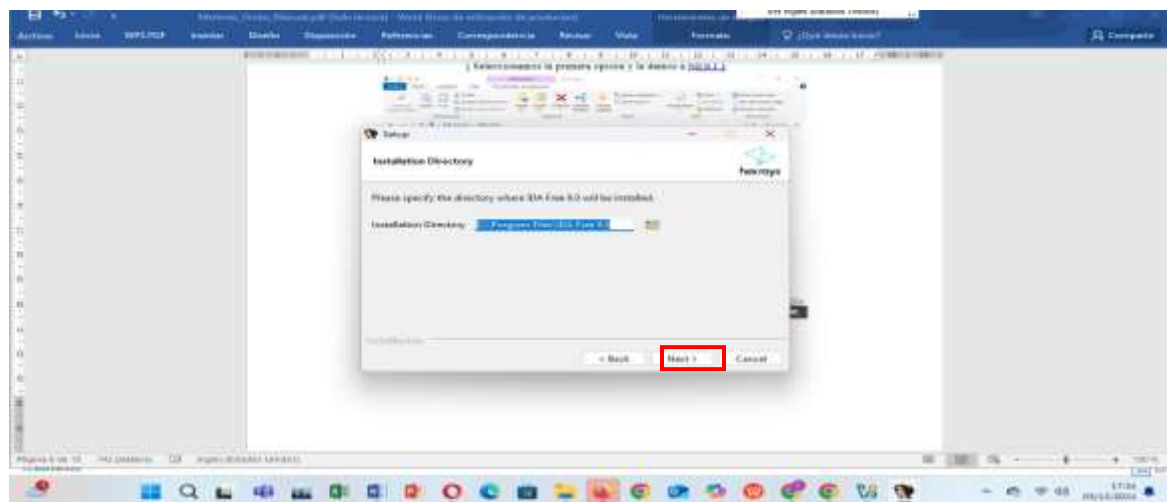
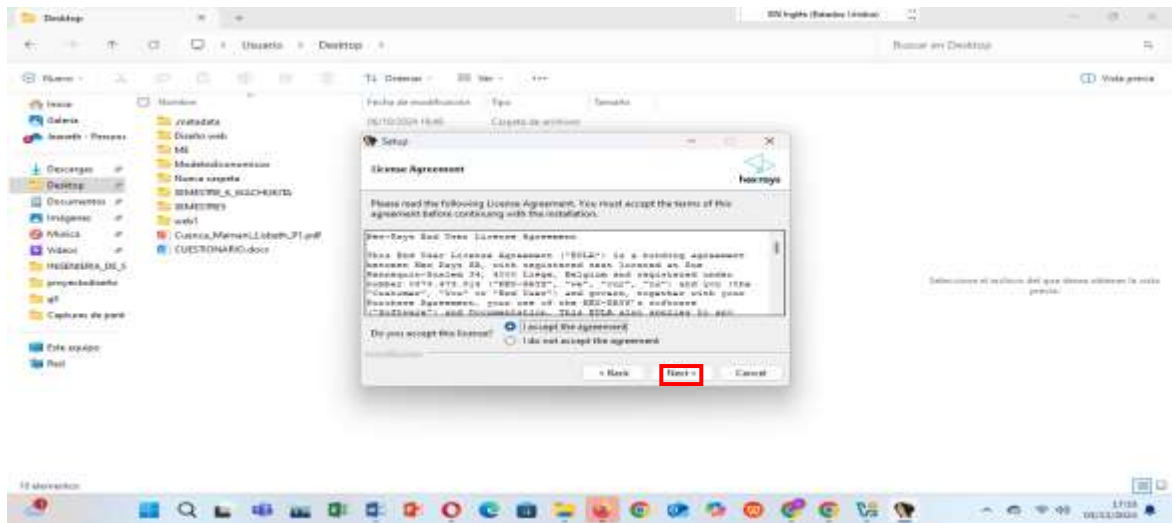
Ordenar

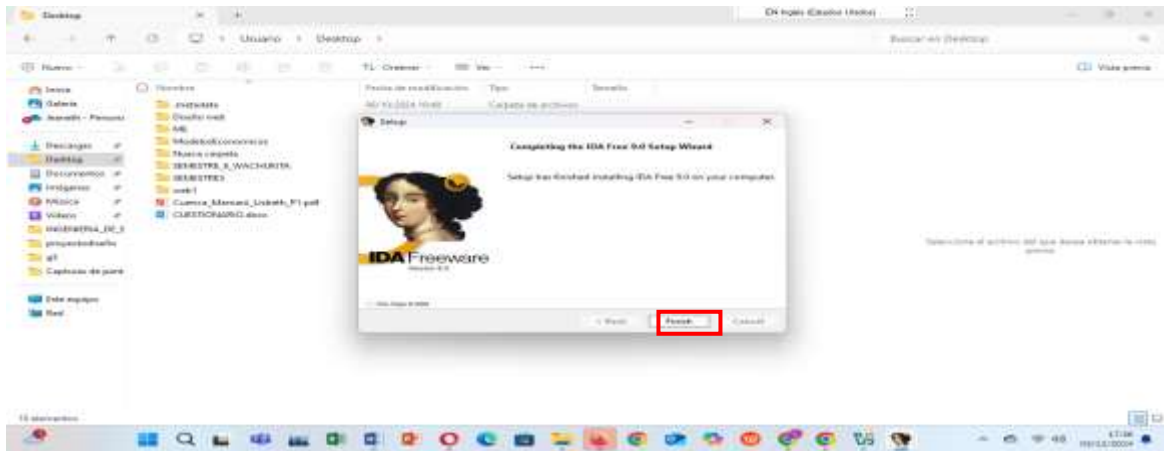
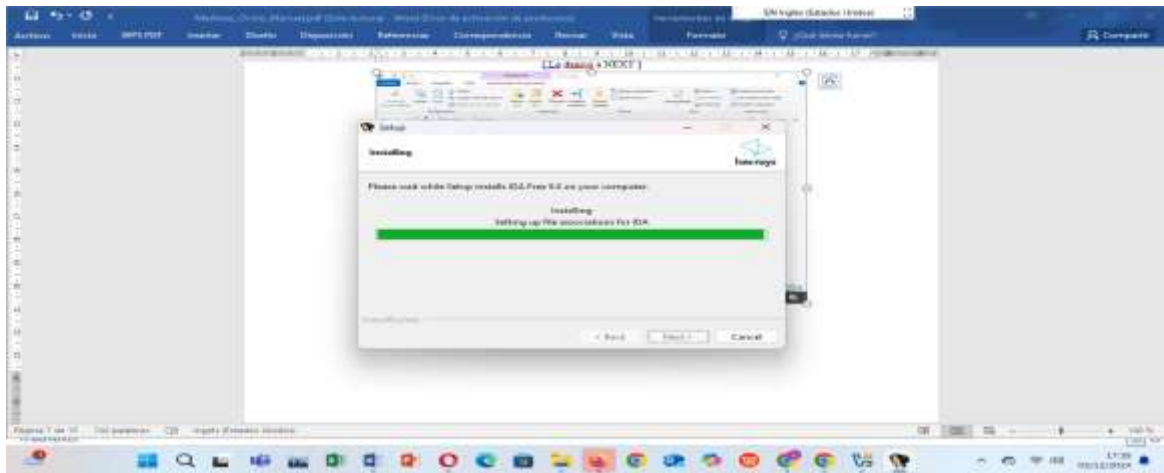
Ver

<input type="checkbox"/> Nombre	Fecha de modificación	Tipo	Tamaño
▼ hoy			
ida-free-pc_90_x64win.exe	03/12/2024 17:22	Aplicación	319.511 KB
VirtualBox-7.1.4-165100-Win.exe	03/12/2024 16:02	Aplicación	108.437 KB
Oracle_VirtualBox_Extension_Pack-7.1....	03/12/2024 15:55	VirtualBox Extensi...	22.429 KB
Oracle_VirtualBox_Extension_Pack-7.1....	03/12/2024 15:56	Archivo WinRAR	22.022 KB

Paso 2: Instalación

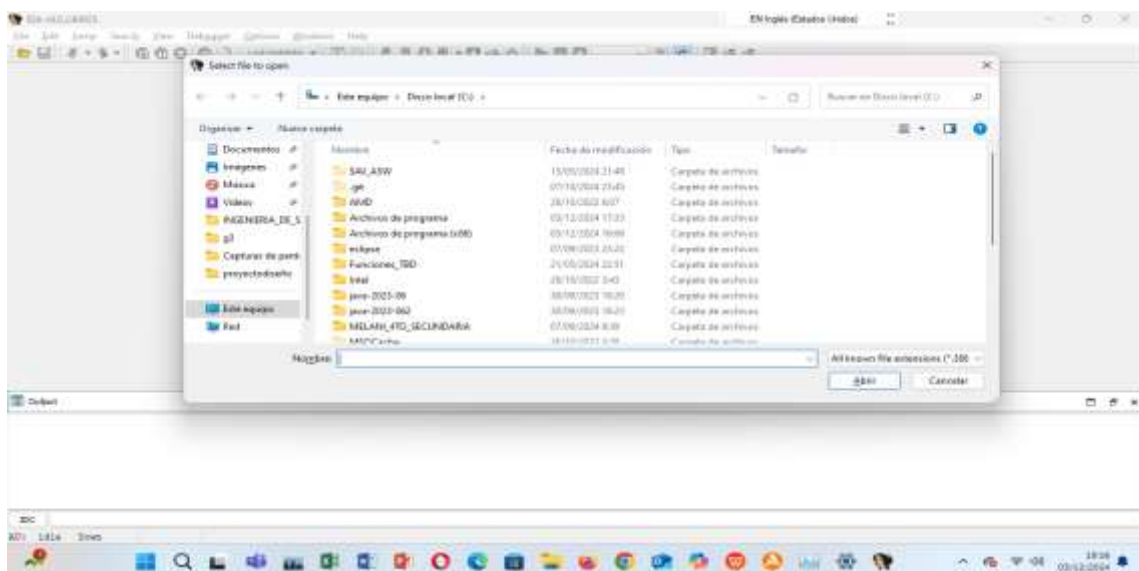
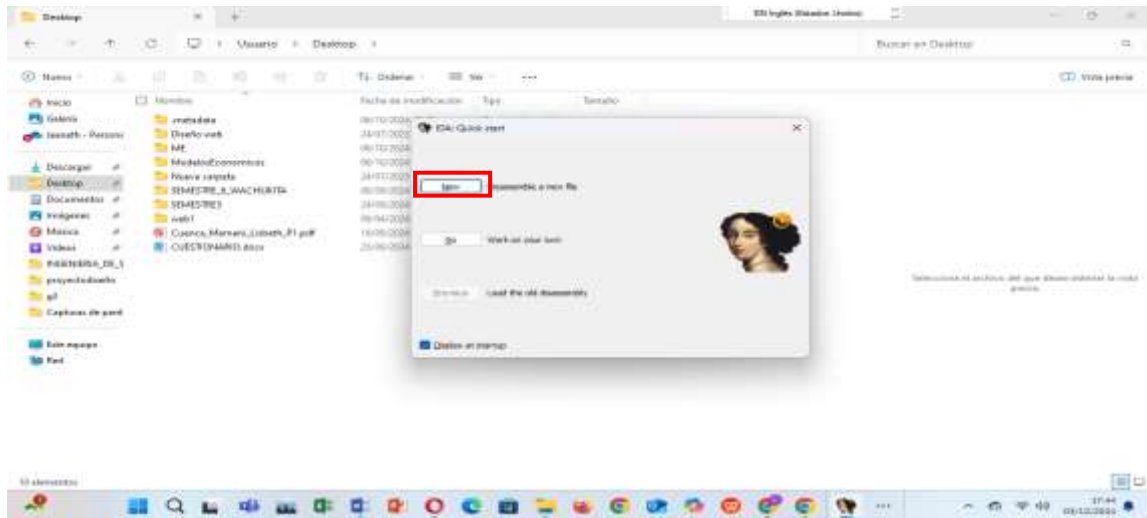
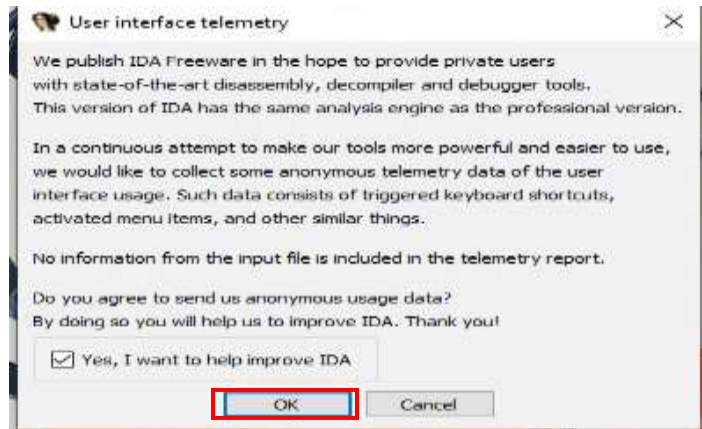




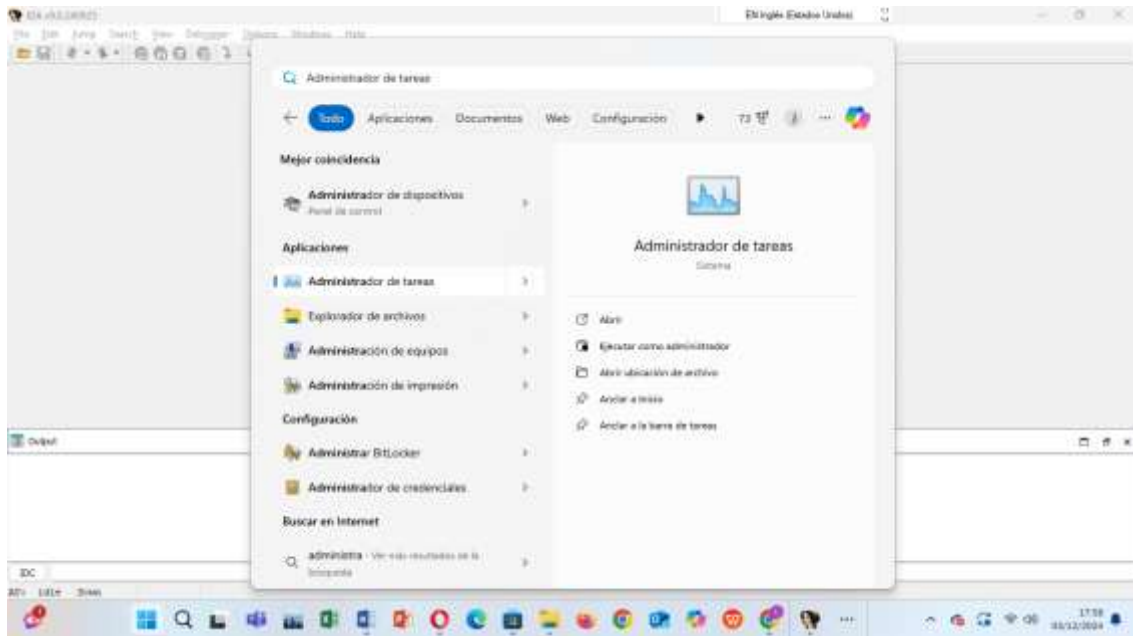


Paso 3: Procederemos a abrir un servicio en Windows

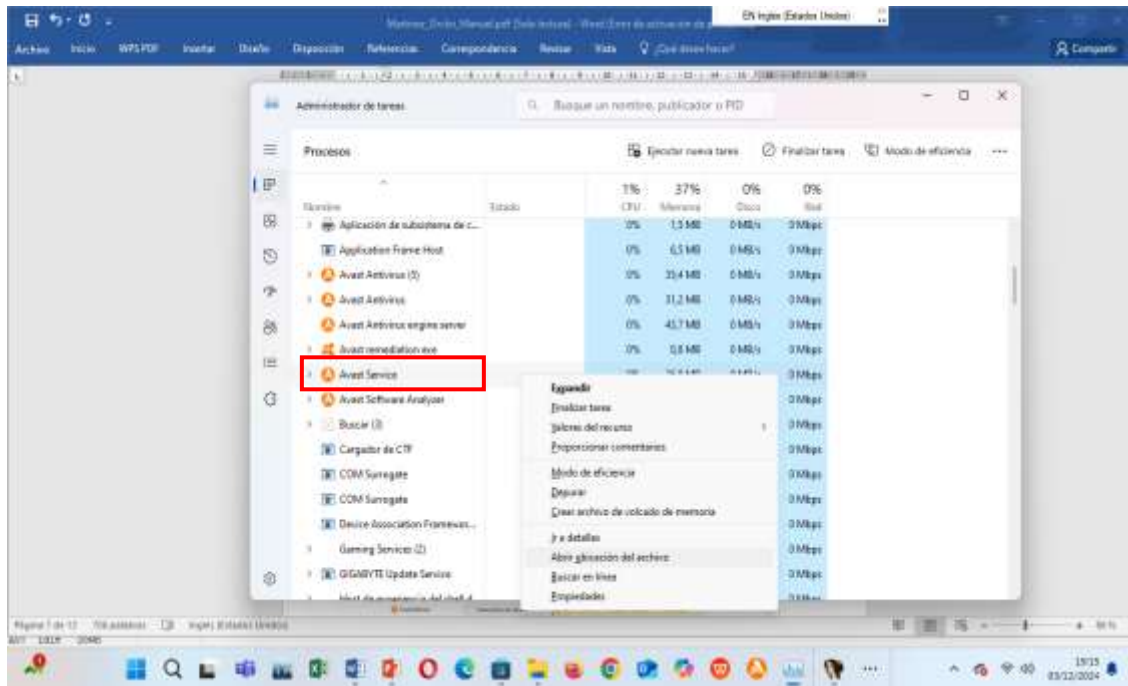




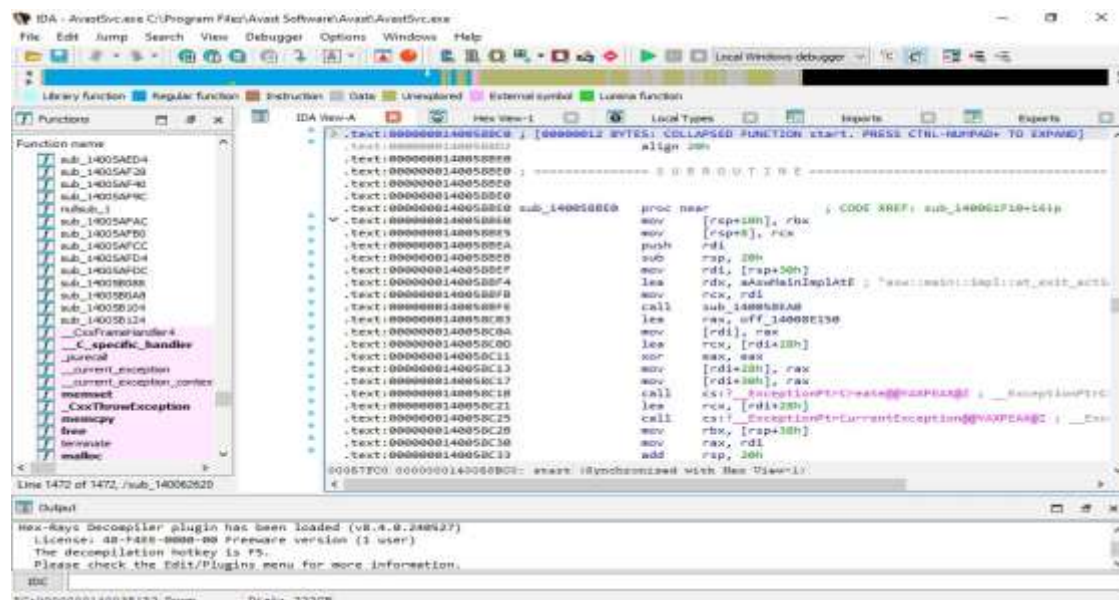
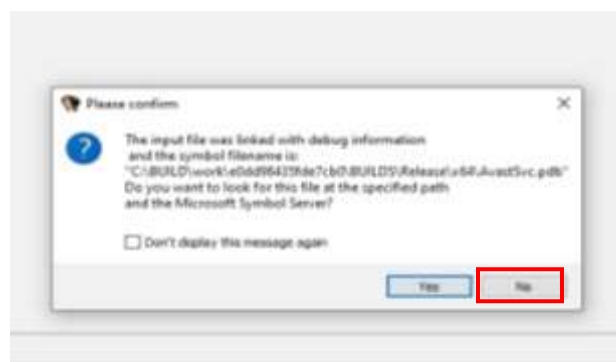
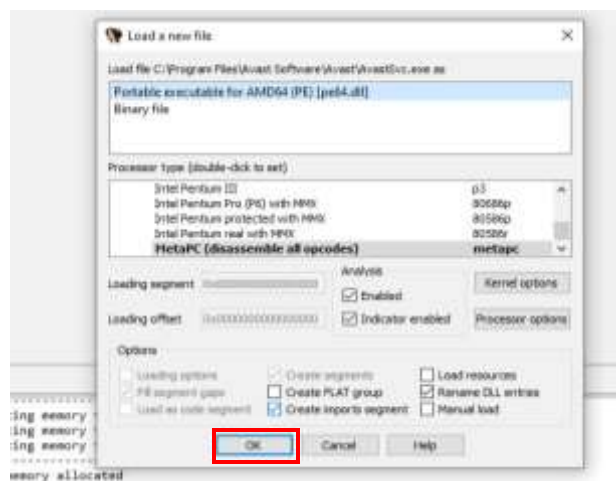
[En este seleccionamos un servicio para analizar]



[Antes nos dirigimos al administrador de tareas]



[Escogemos el servicio que queremos y abrimos en la ubicación del archivo]



Paso 4: Finalmente, se podrá ver código Assembler del servicio que hemos desensamblado

The screenshot shows the IDA Pro interface with the assembly view selected. The function list on the left includes sub_14005AD0, sub_14005AD8, sub_14005AE0, j_UserMathErrorFunction, sub_14005AED4, sub_14005AF28, sub_14005AF40, sub_14005AF9C, nullsub_1, sub_14005AFAC, sub_14005AFB0, sub_14005AFDC, _GSHandlerCheck, _GSHandlerCheckCommon, sub_14005B104, sub_14005B124, _CoframeHandler, _C_specific_handler, _purecall, _current_exception, _current_exception_context, memset, and _CoxThrowException. The assembly view shows the following code:

```
.text:00000000140058B0 sub_140058B0 proc near  
: CODE XREF: sub_140058C40+92F64p  
: DATA XREF: .pdata:000000001400AC25C1  
arg_0 = qword ptr 0  
arg_8 = qword ptr 10h  
mov [rsp+arg_8], rbx  
mov [rsp+arg_0], rcx  
push rdi  
sub rsp, 20h  
rdi, [rsp+28h+arg_0]  
lea rcx, aNewWinImpLate ; "new::win::impl::exit_acti  
mov rcx, rdi  
call sub_140058E40  
lea rcx, off_14008E150  
mov [rdi], rcx  
lea rcx, [rdi+28h]  
xor eax, eax  
mov [rdi+28h], rcx  
call cs:_ExceptionPtrCreate@VAXPE40 ; __ExceptionPtrC  
lea rcx, [rdi+28h]  
call cs:_ExceptionPtrCurrentException@VAXPE40 ; __Exc  
mov rcx, [rsp+28h+arg_0]  
mov rcx, rdi  
add rsp, 20h  
000577E0 00000000140058B0: sub_140058B0 (Synchronised with Hex View-1)
```

