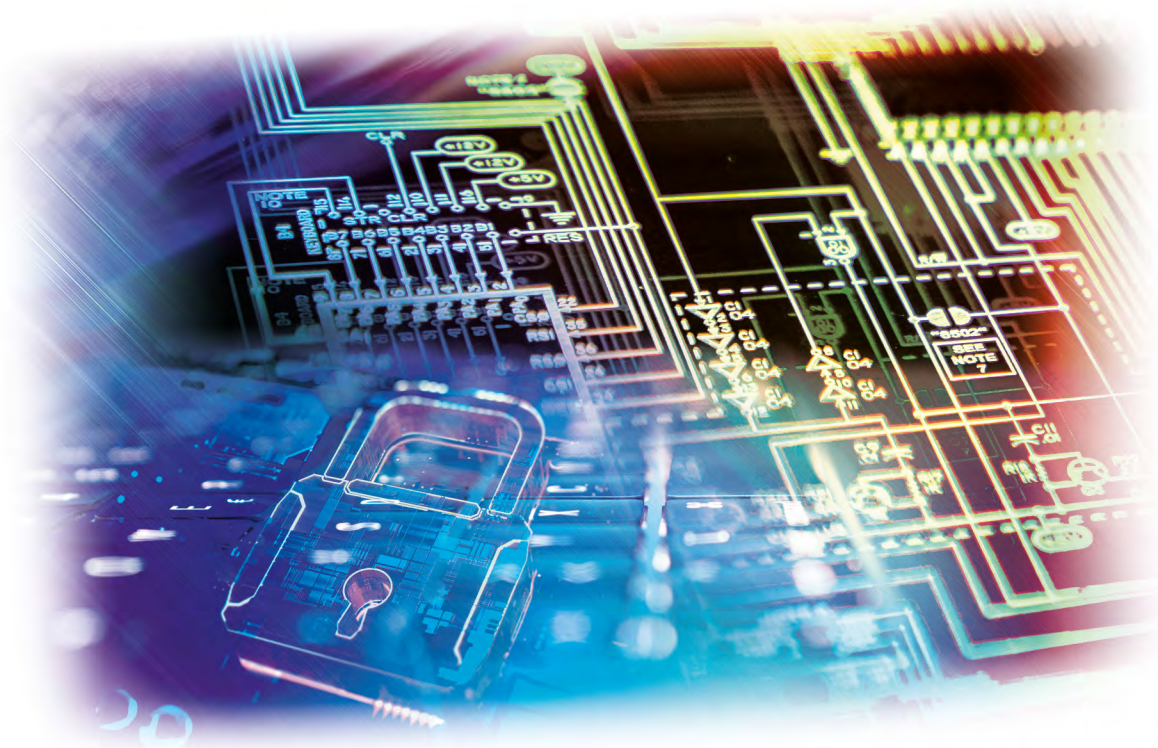
Ingeniería inversa

Curso práctico



Desde **www.ra-ma.es** podrá descargar material adicional.

Cayetano de Juan

Ra-Ma®

edU®

Ingeniería inversa

Curso práctico

Cayetano de Juan

Ra-Ma®
edU
Conocimiento a su alcance

De Juan, Cayetano

Ingeniería inversa / Cayetano de Juan --. Bogotá: Ediciones de la U, 2023
272 p. ; 24 cm
ISBN 978-958-792-521-0 e-ISBN 978-958-792-522-7
1. Informática 2. Lenguajes de programación 3. Lenguaje ensambladorl. Tít.
621.39 ed.

Edición original publicada por © Editorial Ra-ma (España)
Edición autorizada a Ediciones de la U para Colombia

Área: Informática
Primera edición: Bogotá, Colombia, abril de 2023
ISBN. 978-958-792-521-0

Ediciones de la U es una empresa editorial que, con una visión moderna y estratégica de las tecnologías, desarrolla, promueve, distribuye y comercializa contenidos, herramientas de formación, libros técnicos y profesionales, e-books, e-learning o aprendizaje en línea, realizados por autores con amplia experiencia en las diferentes áreas profesionales e investigativas, para brindar a nuestros usuarios soluciones útiles y prácticas que contribuyan al dominio de sus campos de trabajo y a su mejor desempeño en un mundo global, cambiante y cada vez más competitivo.

Coordinación editorial: Adriana Gutiérrez M.
Carátula: Ediciones de la U
Impresión: DGP Editores SAS
Calle 63 #70D-34, Pbx (+57) 601 7217756

Impreso y hecho en Colombia
Printed and made in Colombia

AGRADECIMIENTOS

Cayetano Úbeda Granero, Diego De Juan Fernández, Juan Úbeda Granero, vosotros me marcasteis el camino a seguir y me disteis recursos para caminar sobre él, yo no me he desviado nunca de ese camino, ahora lo ando con mi mujer, Natalia Monfort, gracias infinitas.

ÍNDICE

ACERCA DEL AUTOR

CAYETANO DE JUAN UBEDA

Técnico Superior en Informática de Gestión. Auditor sénior en ciberseguridad, colegiado como perito judicial en informática forense. Diferentes certificaciones, como Cybersecurity Essentials por CISCO, CPHE (Certificado profesional de hacking ético). Cursos finalizados pendientes de certificación, PCAP (Programming Essentials In Python) por CISCO, CyberOps Associate por CISCO. EC Council Hacking y Ciberoperaciones por CISCO. Formador ocupacional por la Junta de Andalucía. Formador de formadores en la especialidad de Teleformación. Programación VB.NET, Python, lenguaje Ensamblador. Actualmente cursando el CEH.

Ha trabajado como consultor en The Security Sentinel, auditando empresas de nivel internacional como farmacéuticas y partidos políticos de gran relevancia, organismos oficiales como la Junta de Andalucía, Entidad Nacional de Acreditación. Ha trabajado como formador, impartiendo módulos al servicio Andaluz dSe Salud, Fundación O.N.C.E, etc. Su última formación fue por medio de la Fundación IL3 de La Universidad de Barcelona al PDI de Chile, edición 2022 (Estándares corporativos en protección de datos y detección y análisis de datos vulnerables).

En estos momentos dirige el departamento técnico auditor de The Security Sentinel, compaginándolo con proyectos formativos a destacar ARELANCE, impartiendo curso de ciberseguridad.

PRÓLOGO

En este libro encontrará, explicado de manera muy dinámica, la programación de aplicaciones, librerías, drivers y todo lo que imagine usando el lenguaje Ensamblador, el lenguaje más próximo al "hierro" e ingeniería inversa.

Podrá practicar, sin necesidad de conocimientos previos, cada capítulo mediante ejercicios prácticos que os ayudará a asentar los conocimientos adquiridos, de manera sencilla y explicados paso a paso.

Conocer el código máquina le abrirá un mundo nuevo en muchos campos de la informática, verá los programas de otra manera y deseará explorarlos.

En primer lugar, será capaz de crear aplicaciones más eficientes debido a que los lenguajes interpretados necesitan convertir el código creado en lenguaje máquina, haciendo que el código del programa crezca de manera innecesaria.

Además, le permitirá hacer ingeniería inversa en ejecutables ya creados y le permitirá entender el funcionamiento de un programa y detectar fallos, puertas traseras, desbordamientos de buffer, etc.

Sobre el autor, Cayetano De Juan, solo puedo decirle que es un apasionado de la tecnología y un gran profesional, experto en seguridad informática y programador con más años de experiencia que páginas tiene el libro.

Es un honor poder escribir este prólogo, como alumno del curso de ensamblador, en el que está basado este libro, os puedo decir que al comenzar no sabía si sería capaz de disfrutar programando en ensamblador ya que en la carrera de ingeniería informática había dado clases de ensamblador y el recuerdo no era muy agradable.

La manera de explicar, y que desde el primer momento ya te pones a practicar las lecciones aprendidas, hace que la curva de aprendizaje sea muy rápida y nada más acabar un tema ya tienes en la mente programas que podrías mejorar con lo aprendido.

Recomiendo encarecidamente este libro, no es uno de esos libros de informática que tenemos en la estantería y que le tenemos que quitar el polvo los sábados, es un libro que tendrá encima del escritorio para poder consultarlo a diario.

Raül Rivero.
Ingeniero Informático.

INTRODUCCIÓN

La Ingeniería Inversa, se refiere al estudio detallado de las funciones del malware, paso a paso, con el fin de descubrir cuál es el código responsable por su funcionamiento. Es una de las disciplinas más gratificantes dentro de la seguridad informática.

Esta obra le explica de forma secuencial como poner en práctica esta materia a través de explicaciones claras y didácticas, acompañados de ejemplos y ejercicios de autoevaluación.

Para facilitar la asimilación de los contenidos se ha dividido en dos partes:

En la primera parte aprenderá el lenguaje de más bajo nivel legible que existe, el lenguaje **Ensamblador**, y lo hará comenzando desde cero con este orden:

- A moverse por el mundo de las API de Windows.
- A enlazar Ensamblador con lenguajes de alto nivel como Python y VB.Net.
- A crear su propia Shell Inversa en Ensamblador y conectarla con Python.
- A crear sus propias DLL.

En la segunda parte asimilará a interpretar los programas compilados y aprenderá:

- A crear ficheros Binarios PE.
- A poner puntos de ruptura (memoria, anular objetivos por API, por cadenas, etc.).
- A crear sus propios parches o cambios en un binario.
- A cifrar texto por XOR.
- A reconstrucción de código intermedio.
- A analizar un binario contaminado por Malware real.

Con este libro obtendrá los conocimientos necesarios para poder usar desensambladores y depuradores como IDA Pro, OllyDBG, Inmmunity Debugger, WinDBG, etc.

Además, con esta obra tendrá acceso a 40 videos y supuestos prácticos descargables desde la web del libro que complementan el contenido y que están indicados en el libro con los iconos  y  .

Parte 1

LENGUAJE ENSAMBLADOR

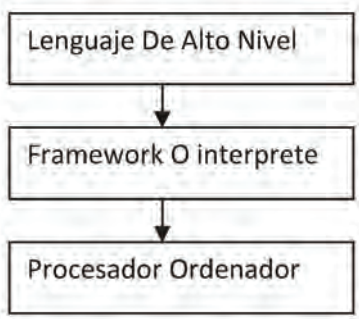
1

CONCEPTOS BÁSICOS Y EXPECTATIVAS DEL CURSO

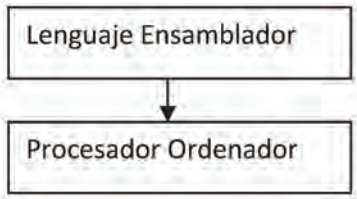
Video 1

El lenguaje ensamblador es un lenguaje de bajo nivel, se compone de una serie de instrucciones básicas para los microprocesadores. Este lenguaje no necesita un Framework para poder ejecutar los programas realizados con él, como por ejemplo Vb.net. Esto quiere decir que nuestros ejecutables podrán cargarse o ejecutarse sin necesidad de una plataforma detrás que haga la interpretación.

Lenguajes de Alto Nivel:

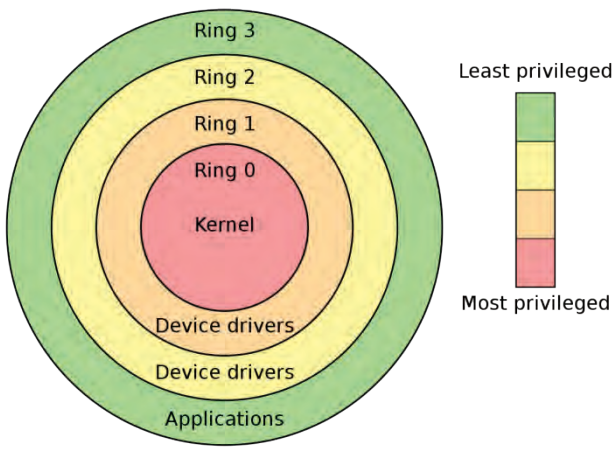


Lenguaje Ensamblador (Lenguaje de bajo nivel)



Desde el comienzo de los sistemas operativos, empezando por el Ms-Dos. El lenguaje ensamblador ha estado presente, con este lenguaje se podría controlar absolutamente todo el ordenador, memoria, disco duro, periféricos. De hecho otros lenguajes como C, realmente provienen del lenguaje ensamblador. Antiguamente en un sistema Ms-Dos con el lenguaje ensamblador se podía programar usando las interrupciones del sistema, en los sistemas operativos actuales como Windows, que es en lo que se apoya este curso, el lenguaje ensamblador usaría las denominadas **API** de Windows que vienen a ser esas interrupciones de Ms-Dos.

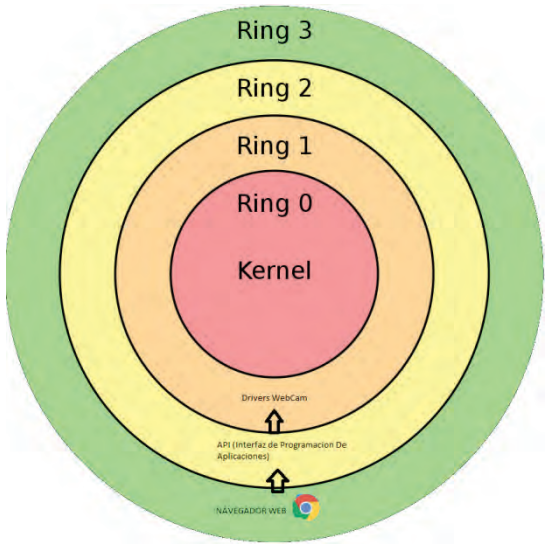
El esquema de desarrollo de Windows quedaría algo parecido a esto:



Windows dispone de un sistema de protección del **Kernel**, basado en anillos, de esta forma se asegura de que ningún programa mal intencionado pueda dañar su núcleo. Los anillos se pueden comunicar entre sí mediante “puertas” que controlan el propio sistema operativo.

Supongamos el siguiente ejemplo:

Tenemos un ordenador con una webcam, la webcam tiene instalado en el sistema sus drivers necesarios para poder funcionar. Desde el navegador queremos realizar una video conferencia, usando esa webcam. Bien, pues el navegador estaría en el anillo tres, los drivers de control de la cámara estarían en el anillo/dos o uno. Algo así:



- **Nivel 1:** En este nivel está la unidad central de proceso, que resumiendo se componen de pequeñas funciones en lenguaje máquina que introduce el propio fabricante, sumar, restar.
- **Nivel 2:** El sistema operativo se compone de diferentes funciones ya más adecuadas al programador o usuario del mismo, con las que se pueden crear carpetas, archivos, ejecutar programas. Estas funciones se traducen a código máquina las cuales pasarían al **Nivel 1**.
- **Nivel 3:** Los lenguajes de programación, se encuentran siempre por encima del sistema operativo, estos lenguajes como ya hemos comentado, anteriormente, pueden ser lenguajes de Alto o Bajo nivel. En el caso del Lenguaje Ensamblador es de bajo nivel, esto quiere decir que este lenguaje usa instrucciones básicas que pueden ser interpretadas directamente por el **Nivel 1**, y al mismo tiempo puede

usar funciones propias del sistema operativo del **Nivel 2**, en nuestro caso estas funciones propias del sistema operativo serán las **APIS**. Los programas realizados en ensamblador, una vez compilados se traducen en su totalidad a lenguaje de código máquina.

- **Nivel 4:** Los lenguajes de Alto nivel, como C++, Vb.Net, Java, etc. Se componen de framework o poderosas librerías que agilizan y nos simplifican el tratamiento por ejemplo de acceder a un disco duro, cámara web. Los compiladores de estos lenguajes traducen los programas realizados con ellos, a programas de **Nivel 3**, y a su vez traducen en código de **Nivel 3**.

En este curso empezaremos desde cero, para conocer la base del lenguaje ensamblador bajo Windows 32Bits. Se presentará el **EASY CODE**, un Software con entorno visual, para poder desarrollar en lenguaje ensamblador, este Software creado por Ramón Salas, nos ayudará a crear nuestros programas en ensamblador de una forma más sencilla y amena.

Una vez terminado el curso, debería haber obtenido los siguientes conocimientos:

- Base y desarrollo de proyectos bajo lenguaje ensamblador.
- Conocimientos sobre el funcionamiento interno de Windows.
- Conocimientos sobre aplicaciones de Windows.
- Conocimientos sobre las APIS de Windows.

1.1 FUNCIONAMIENTO DE WINDOWS, MENSAJES Y EVENTOS

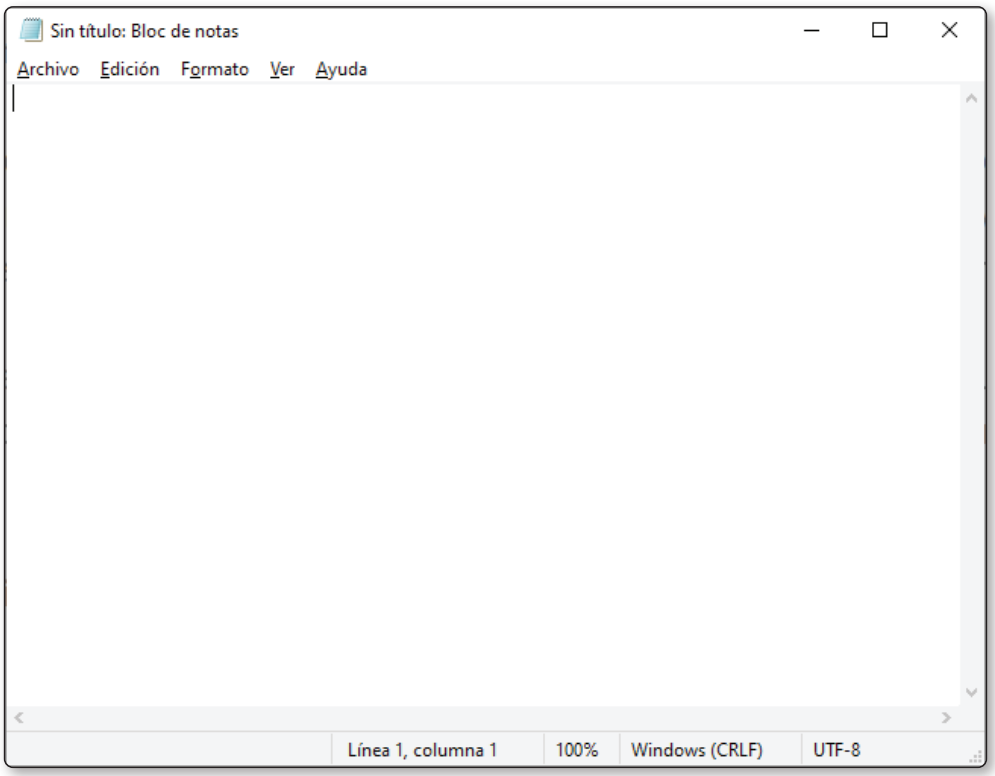
Si alguna vez ha desarrollado un programa, por muy pequeño que sea, con algún lenguaje de alto nivel, como Vb.net, Java, etc. Y si este lenguaje de alto nivel, es un lenguaje visual, donde se parte de un formulario, al que le vamos agregando controles personalizados como botones, cajas de texto. Pues toda esta integración al final se traduce en las **API de Microsoft Windows**.

Básicamente el funcionamiento interno de Windows se traduce en dos principios:

- Para el sistema, todo parte de una ventana (un simple botón, para Windows es una ventana, en general cualquier objeto, es tratado como una ventana).
- Las ventanas y objetos que la componen se comunican por medio de mensajes.

Todos los objetos, o ventanas de Windows se identifican mediante un **handle**. Y a su vez, de un procedimiento que es capaz de interceptar mediante mensajes como se tiene que comportar.

Por ejemplo si abrimos el Bloc De Notas, veremos seguramente una pantalla parecida a esta:



Si pudiéramos ver el código del núcleo de esta aplicación, sería algo parecido a esto.

El control de mensajes, se comportaría como un bucle, donde está siempre recibiendo mensajes por parte de la aplicación, si movemos el ratón dentro de nuestro programa Bloc De Notas, ese mensaje será recibido en el control de mensajes, acciones de teclado, como la pulsación de una tecla, apertura del menú. El control de mensajes a su vez los procesa en el segundo cuadro Tratado De Mensajes.

1.2 ¿PARA QUÉ PODEMOS USAR EL LENGUAJE ENSAMBLADOR?

El lenguaje ensamblador, al ser un lenguaje de bajo nivel, nos va a permitir en resumen dos cosas:

- Rapidez.
- Poder crear pequeños procesos o utilidades que los lenguajes de alto nivel, no nos dejan o es demasiado engorroso hacerlo.

Otra de las ventajas del lenguaje ensamblador es que no tiene que pasar por ningún framework para poder ejecutar su código, una vez compilado, el lenguaje ensamblador esta convertido en lenguaje máquina.

Entender este lenguaje, nos abre también una puerta a la ingeniería inversa o a desensamblar programas compilados con lenguajes de alto nivel.

1.3 NUMERACIÓN Y CÁLCULO ARITMÉTICO

Antes de meternos de lleno con el lenguaje ensamblador, explicare unas pequeñas nociones de cómo cuentan los ordenadores. Esto debe parecer bastante simple: 1, 2 ,3, 4, 5, 6, 7, 8, 9, 10, 11. Esta numeración la entenderíamos bien 1 más 1 es 2, mas 1, 3. Bien pues los ordenadores no cuentan así, lo hacen en formato binario (base 2), para contar hasta 3 lo haría de esta forma: 1, 10, 11. El sistema binario es un sistema numérico que solo tiene dos dígitos, el 0 y el 1. Al contrario de los 10 dígitos que tenemos en nuestro sistema decimal (base 10).

El microprocesador de nuestro ordenador, cuenta en este modo binario, pero como para nosotros sería un poco complejo el tratar con ciertos números y traducirlos a números binarios, vamos a usar algo intermedio, que serían los números hexadecimales. Los números hexadecimales, al menos nos van a ayudar a tratar con los números binarios recortando su longitud.

1.3.1 Números hexadecimales

Para intentar hacer más ameno, el aprendizaje de los números hexadecimales, vamos a usar un viejo programa, que viene de MS-DOS **“Debug**”. La evolución de este programa, es “**WinDBG**”.

El nombre de “**Debug**” seguro que ya nos dice algo, “Bugs” informáticamente, son bichos… que serían los fallos de un programa.

Debug, nos ayudara a ejecutar programa instrucción por instrucción, esto seguro que para los que os gusta la ingeniería inversa, os llamara un poco la atención, pues pudiendo ver cada una de las líneas de ejecución de un programa podríamos estar analizando o depurando dicho programa.

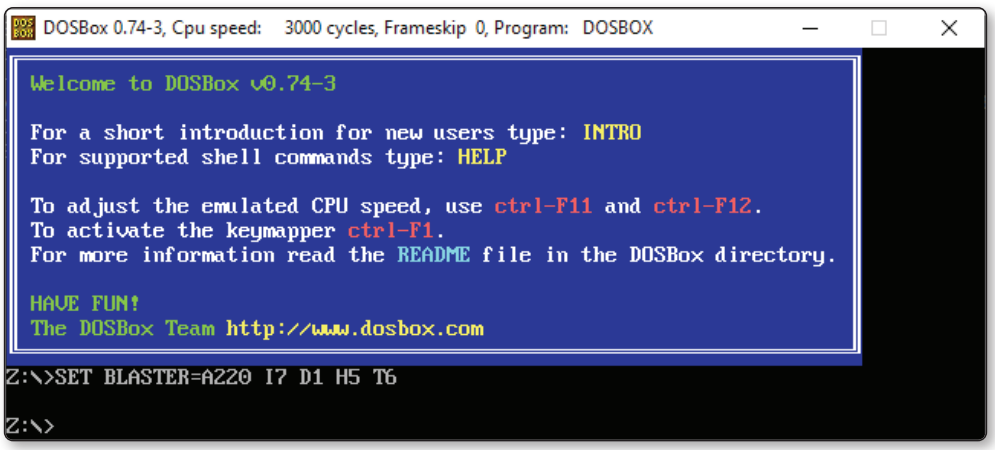
Para poder usar **Debug**, en nuestro sistema operativo actual, Windows 8.1, Windows 10 o Windows 11. Tendremos que descargar dos programas.

- El primero una especie de simulador de Ms-Dos.
- El segundo el propio Debug.exe.

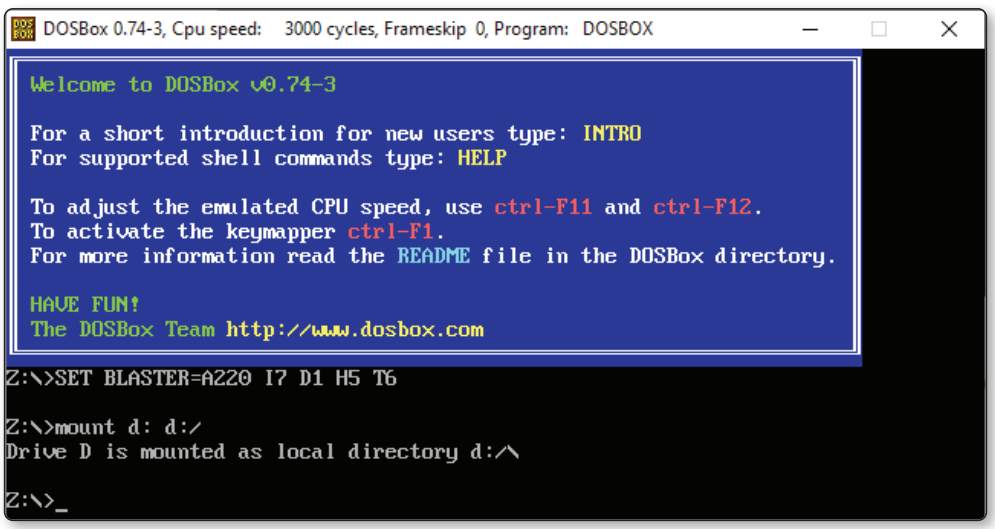
https://www.dosbox.com/download.php?main=1
http://www.mediafire.com/file/bopdmu16tav8thg/debug.zip/file

Una vez instalado el primer programa, vamos a descomprimir nuestro "Debug.exe" en la unidad D: (en su caso la unidad que corresponda).

Una vez descomprimido, ejecutamos el primer programa "**DosBox**".

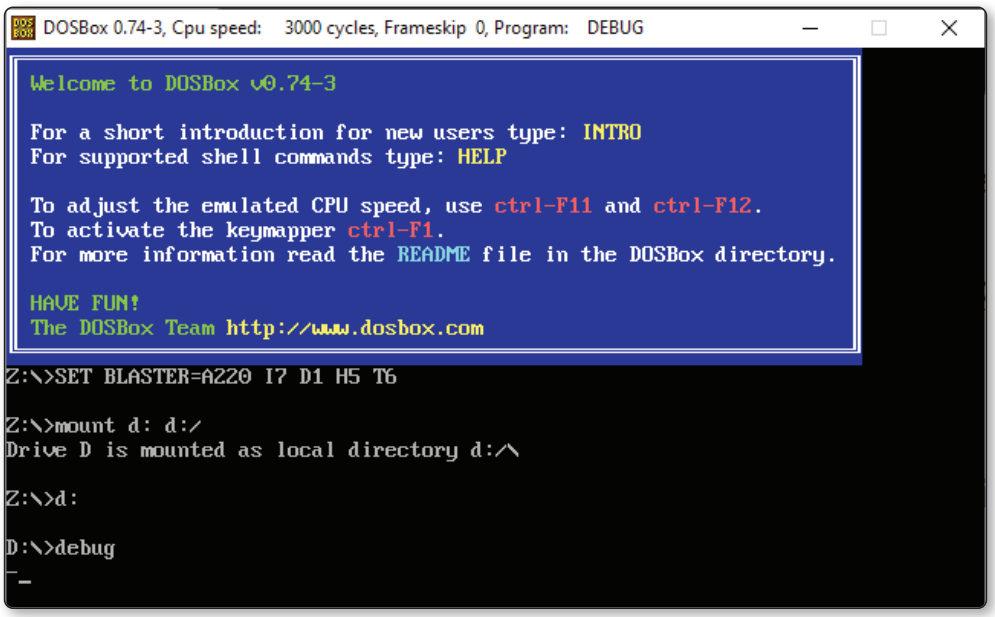


Y vamos a montar la unidad D:, que es donde está nuestro debug (en su caso la unidad que corresponda).



De esta forma, ya tendríamos acceso a nuestro programa “Debug.exe”

Pasándonos a la unidad D: y tecleando Debug. Ya estaríamos dentro del mismo.



Si se fija en la foto superior, al ejecutar el **Debug**, solo aparece un guion. Bien esto sería correcto, el programa estaría esperando a que empecemos a meter comandos. Para salir de **Debug**, y regresar al Ms-Dos, vamos a usar la letra “**Q**”.

Veamos otro comando de **Debug**, pero ya aplicado a lo que nos interesa aprender, la suma de dos números hexadecimales.

Vamos a decirle a **Debug**, que sume 2 + 3 = 5.

Pondríamos lo siguiente en nuestro **Debug**.

```
-H 3 2
```

Debug, nos devolvería:

```
0005  0001
```

En general se vería algo así:

```
D:\>debug
-H 3 2
0005  0001
-
```