

## Registros del Microprocesador

### 1.- Tipos de Registros

Al usar ASM nos valemos directamente de los registro de nuestro microprocesador a diferencia de los lenguajes de alto nivel.

#### 1.1.-Registros de uso General :

**EAX** (Acumulador) : Utilizado para operaciones aritmetica (suma,resta,multiplicacion,divisi?n) , la mayoría de funciones despues de ser utilizadas devuelven un valor a EAX .

**EBX** (Base) : Se utiliza para direccionar el acceso a datos , situados en la memoria ; tambien se puede utiliza para operaciones arim?ticas .

**ECX** (Contador) : Se utiliza como contador por algunas instrucciones ; tambien se puede utiliza para operaciones arim?ticas .

**EDX** (Datos) : Algunas operaciones de entrada/salida requieren su uso ; las operaciones de multiplicaci?n y divisi?n con cifras grandes suponen al EDX y EAX trabajando juntos .

#### 1.2.-Registros de Indice :

**ESI** y **EDI** : Son requeridos por algunas operaciones con cadenas de caracteres .

#### 1.3.-Registros de banderas :

Se usan para registrar la informaci?n de estado y de control de las operaciones del microprocesador, y son 9 :

**CF, OF, ZF, SF, PF, AF, DF, IF, TF.**

#### 1.4.-Otros Registros Importantes :

**EIP** : Este registro contendr? la direcci?n de la siguiente instrucci?n que se ejecutar?.

**EBP** : Apunta a la Base de la pila.

**ESP** : Apunta a la parte Superior de la pila.

### 2.- Tama?os de los Registros

EAX (4 bytes = 32 bits)	
Sin nombre	AX (2 bytes = 16 bits)
	AH (1byte)   AL (1 byte)

Lo que representa la imagen expuesta no solo es aplicable al registro EAX sino que puede ser cambiado por los demas registros de uso general (EAX, EBX, ECX, EDX)

Ejemplo :

**12345678** => EAX

1234**5678** => AX

1234**56**78 => AH

123456**78** => AL

Referencias :

**DWORD** -> EAX

**WORD** -> AX

**BYTE** -> AH y AL

## La Pila

Seg?n la Wikipedia :

- T?nicamente es una estructura de datos, del tipo LIFO (del ingl?s Last In First Out, ?ltimo en entrar, primero en salir)

Enlace : <http://es.wikipedia.org/wiki/LIFO>

?Que esto de una estructura?. Bien, imaginad que formamos una ventana, o por ejemplo, el t?pico mensaje de texto. En ?ste caso es la API MessageBoxA .



Y claro, aqu? es donde entra el papel de la pila, antes de invocar a la API MessageBoxA tenemos que indicarle todos estos datos para que nos la forme tal y como nosotros le hemos indicado.

**PUSH T?tulo**

**PUSH Cuerpo**

**PUSH TipodeBotones**

**CALL MessageBoxA**

MessageBoxA es la API encargada de lanzar ese mensaje tipo "caja de texto" que puse en la imagen.

En Ensamblador o en C esto se visualizaria as? :

Llamada a MessageBoxA en C:  
MessageBoxA (0, "Hola mundo", "xD", 0);

Pila:

Código equivalente en ASM:

```
push 0
push Puntero_xD
push Puntero_HolaMundo
push 0
call[MessageBoxA]
```

## Practicando el Lenguaje Ensamblador

### 1.- Instrucciones Básicas del Microprocesador :

Comenzaremos por las instrucciones más básicas, con breves explicaciones y ejemplos:

#### **MOV destino, origen :**

Sirve para mover información del origen al destino. Por ejemplo de un registro a otro, de un registro a una variable, etc. Aunque no nos sirve para mover datos entre dos variables.

Ej: **MOV A, B** => El valor de A pasará a ser igual que el valor de B.

#### **ADD/SUB Destino, Cantidad**

**ADD** : Suma una determinada cantidad a un registro o variable (destino).

Ej: **ADD A, B** => Se le suma a A el valor de B y el resultado se guardará en A.

**SUB** : Resta una determinada cantidad a un registro o variable (destino).

Ej: **SUB A, B** => Se le resta a A el valor de B y el resultado se guardará en A.

- Recuerda : No está permitido sumar o restar datos directamente entre variables.

#### **INC/DEC Variable o registro**

**INC** : Incrementa el valor de una variable o registro en 1.

Ej: **INC A** => Se incrementará el valor de A en 1.

**DEC** : Decrementa el valor de una variable o registro en 1.

Ej: **DEC A** => Se decrementará el valor de A en 1.

### **CMP :**

Compara el valor de dos registros , en realidad es una instrucció'n SUB que no guarda el resultado en ningun lado sino que su funcion es cambiar los flags (Registros de banderas) , esta instrucció'n trabaja cuando a los saltos al igual que TEST.

Asumiendo A=B

Ej: **CMP A, B** => Se efectúa una comparaci3n entre A y B (la diferencia de A y B es cero por lo tanto el flag Z se activa)

### **TEST :**

Compara el valor de dos registros , a diferencia de CMP este en realidad es una instrucció'n AND que no guarda el resultado en ningun lado sino que su funcion es cambiar los flags (Registros de banderas)

Ej: **TEST A, B** => Se efectúa una comparaci3n entre A y B

### **PUSH DWORD**

PUSH mete un DWORD en la pila. La pila se utiliza por ejemplo para pasar parámetros a las funciones o para utilizar variables locales. Es importante que cada PUSH tenga un POP (ahora lo explico).

Ej: **push eax**

### **POP DWORD**

POP al contrario que PUSH saca un DWORD de la pila.

Ej: **push eax** => Guarda a eax en la pila

**pop ebx** => Sacar el ultimo valor metido en la pila (en este caso EAX) y lo guarda en EBX (en este caso)

### **RET**

RET se usa generalmente para regresar de una funci3n. Lo que hace realmente es "poppear" un DWORD de la pila y saltar a ?l. De ah? que en las funciones se meta la direcci3n de retorno en la pila y con el RET continúe la ejecuci3n del programa donde corresponde.

A los interesados en el Stack Overflow esto les puede interesar bastante

## **2.- Instrucciones L3gicas :**

Este tipo de operaciones se realizan a nivel de bits (binario) , los resultados son reemplazados en el operando Destino, si deseas realizar los ejemplos manualmente utiliza la calculadora de Windows en modo cientifica para convertir las cantidades a binario.

### **AND Destino, Fuente**

TABLA DE VERDAD AND		
A	B	Resultado
0	0	0
0	1	0
1	0	0
1	1	1

Ej: 1110 (14 en decimal o E hexadecimal).  
1101 (13 en decimal o D hexadecimal).  
 1100 (12 en decimal o C hexadecimal).

### OR Destino, Fuente

TABLA DE VERDAD OR		
A	B	Resultado
0	0	0
0	1	1
1	0	1
1	1	1

Ej: 1110 (14 en decimal o E hexadecimal).  
1101 (13 en decimal o D hexadecimal).  
 1111 (15 en decimal o F hexadecimal).

### XOR Destino, Fuente

TABLA DE VERDAD XOR		
A	B	Resultado
0	0	0
0	1	1
1	0	1
1	1	0

Ej: 1110 (14 en decimal o E hexadecimal).  
1101 (13 en decimal o D hexadecimal).  
 0011 (3 en decimal).

### NOT Destino

TABLA DE VERDAD NOT	
A	Resultado
1	0
0	1

Ej: 1101 (13 en decimal o D hexadecimal).  
 0010 (2 en decimal).

### 3.- Saltos Condicionales :

Simulan lo que sería la estructura IF de un lenguaje de alto nivel .

La forma de todo salto es :

**cmp** Valor1, Valor2

**Salto** Dirección

Hay muchos tipos, así que solo citaré los más importantes :

**JMP** - Salta siempre

**JE** ? Salta si los números comparados son iguales.

**JNE** ? Salta si no son iguales.

**JG** ? Salta si Valor1 es mayor que Valor2.

**JGE** ? Salta si Valor1 es mayor o igual que Valor2.

**JB** ? Salta si Valor1 es menor que Valor2.

**JBE** ? Salta si Valor1 es menor o igual que Valor2.

### 4.- Apis :

Son funciones que pueden ser utilizadas en cualquier programa , para tener que evitar repetir lo mismo en todos los programas.

Aquí pondré las más usadas en el mundo del cracking :

*En construcción*

## Aprendiendo con OllyDbg

### 1.- Introducción :

Habiendo una gran cantidad de debuggers en la actualidad, ¿Por qué vamos a hacer la introducción con OllyDBG?

Creo que la mayoría de expertos en el cracking que han utilizado cualquier debugger reconocen que es más sencillo empezar con OllyDBG, ya que muestra mayor información y es más cómodo para aprender, la idea es ingresar al mundo del cracking, por la puerta del OllyDBG, más adelante cuando uno ya conoce más sobre el tema del cracking, puede trasladar fácilmente a cualquier debugger lo aprendido pues cambian las formas de usar de los programas, pero no la esencia.

### 2.- Presentación de OllyDBG :

OllyDBG es principalmente un debugger para aplicaciones a 32 bits en Microsoft Windows. Es un debugger en Ring3.

Tiene muchas otras opciones e incluso es posible la utilización de plugins (o si eres programador hacerlos tú mismo) que hacen de él un programa verdaderamente potente.

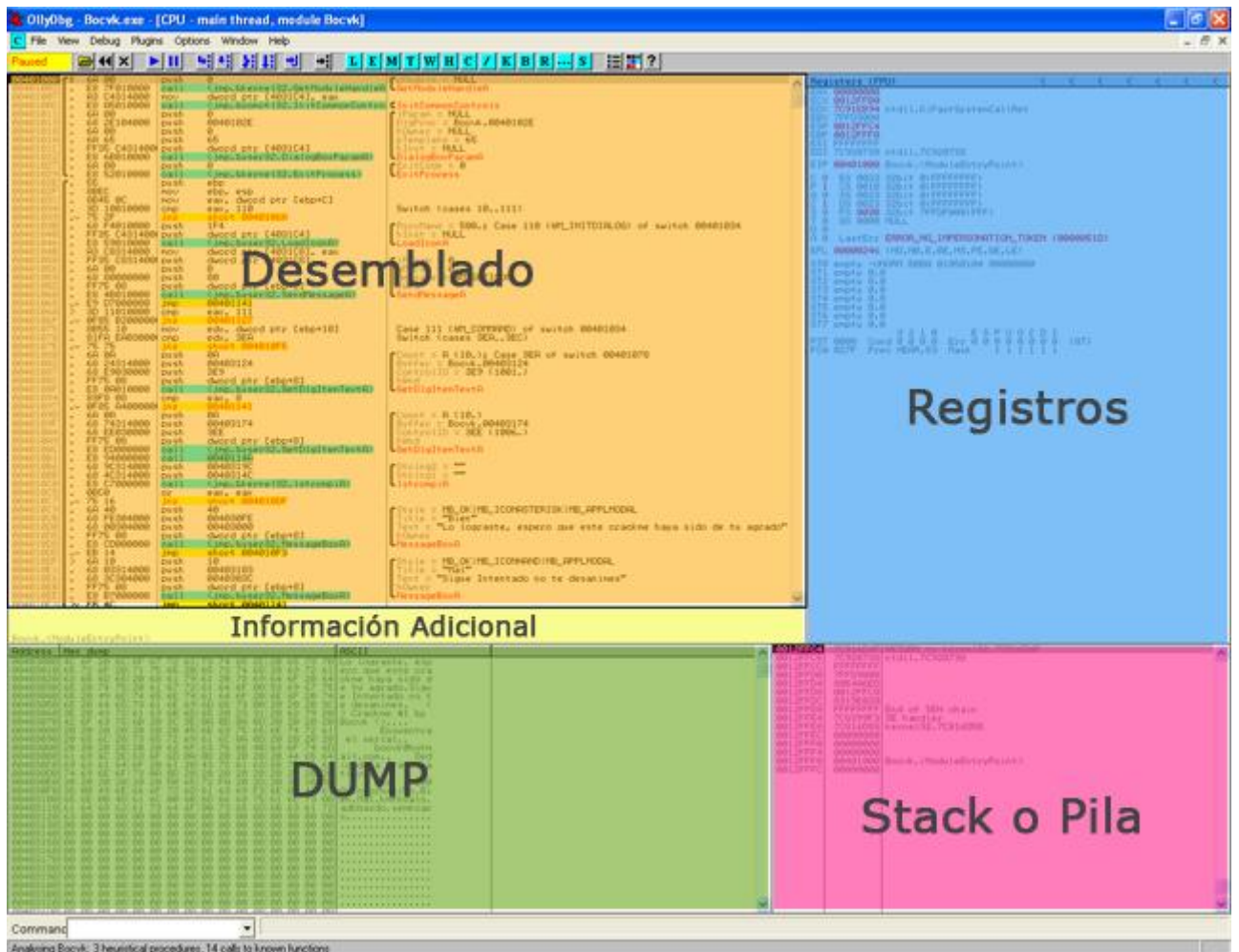
Para conocer sus características podemos visitar su página Web:

<http://www.ollydbg.de/>

Para descargar el programa podemos descargar la versión 1.10 básica, sin ningún plugin y ninguna modificación desde su página web, aquí:

<http://www.ollydbg.de/odbg110.zip>

### 3.- Interfaz :



#### 3.1.- Desamblado :

Address	dump	Disassembly	Comment
00401000	6A 00	push 0	pModule = NULL
00401002	E8 7F010000	call <jmp.&kernel32.GetModuleHandleA	GetModuleHandleA
00401007	A3 C4314000	mov dword ptr [4031C4], eax	
0040100C	E8 A5010000	call <jmp.&comctl32.InitCommonContro	InitCommonControls
00401011	6A 00	push 0	lParam = NULL
00401013	68 2E104000	push 0040102E	DlgProc = Bocvk.0040102E
00401018	6A 00	push 0	hOwner = NULL
0040101A	6A 65	push 65	pTemplate = 65
0040101C	FF35 C4314000	push dword ptr [4031C4]	hInst = NULL
00401022	E8 6B010000	call <jmp.&user32.DialogBoxParamA>	DialogBoxParamA
00401027	6A 00	push 0	ExitCode = 0
00401029	E8 52010000	call <jmp.&kernel32.ExitProcess>	ExitProcess
0040102E	55	push ebp	
0040102F	8BEC	mov ebp, esp	
00401031	8B45 0C	mov eax, dword ptr [ebp+C]	
00401034	3D 10010000	cmp eax, 110	Switch (cases 10..111)
00401039	75 2F	jnz short 0040106A	
0040103B	68 F4010000	push 1F4	RsrcName = 500.; Case 110 (WM_INITDIALOG) of
00401040	FF35 C4314000	push dword ptr [4031C4]	hInst = NULL
00401046	E8 59010000	call <jmp.&user32.LoadIconA>	LoadIconA
0040104B	A3 C8314000	mov dword ptr [4031C8], eax	
00401050	FF35 C8314000	push dword ptr [4031C8]	lParam = 0
00401056	6A 00	push 0	wParam = 0
00401058	68 80000000	push 80	Message = WM_SETICON
0040105D	FF75 08	push dword ptr [ebp+8]	hWnd
00401060	E8 4B010000	call <jmp.&user32.SendMessageA>	SendMessageA
00401065	E9 D7000000	jmp 00401141	
0040106A	3D 11010000	cmp eax, 111	
0040106F	0F85 B2000000	jnz 00401127	
00401075	8B55 10	mov edx, dword ptr [ebp+10]	Case 111 (WM_COMMAND) of switch 00401034
00401078	81FA EA030000	cmp edx, 3EA	Switch (cases 3EA..3EC)
0040107E	75 75	jnz short 004010F5	
00401080	6A 0A	push 0A	Count = A (10.); Case 3EA of switch 00401078
00401082	68 24314000	push 00403124	Buffer = Bocvk.00403124
00401087	68 E9030000	push 3E9	ControlID = 3E9 (1001.)
0040108C	FF75 08	push dword ptr [ebp+8]	hWnd
0040108F	E8 0A010000	call <jmp.&user32.GetDlgItemTextA>	GetDlgItemTextA

## Información Adicional

Bocvk.<ModuleEntryPoint>

- La primera columna "**Address**" muestra la dirección donde se ejecutará el código.
- La segunda columna "**Hex dump**" muestra los bytes que forman la instrucción que hay a la derecha (columna "Disassembly").
- La tercera columna "**Disassembly**" indica el desensamblado propiamente dicho.
- La última columna "**Comment**" nos puede mostrar: comentarios propios, cadenas de texto, direcciones, nombres de funciones y direcciones, datos que vamos a insertar en la pila a cierta función o API etc... que nos pueden ayudar mucho.

### 3.2.- Registros :

Registers (FPU)			
EAX	00000000		
ECX	0012FFB0		
EDX	7C91EB94	ntdll.KiFastSystemCallRet	← Registros
EBX	7FFD5000		
ESP	0012FFC4		
EBP	0012FFF0		
ESI	FFFFFFFF		
EDI	7C920738	ntdll.7C920738	
EIP	00401000	Bocvk.<ModuleEntryPoint>	
C 0	ES 0023	32bit 0(FFFFFFFF)	
P 1	CS 001B	32bit 0(FFFFFFFF)	
A 0	SS 0023	32bit 0(FFFFFFFF)	
Z 1	DS 0023	32bit 0(FFFFFFFF)	
S 0	FS 003B	32bit 7FFDF000(FFF)	
T 0	GS 0000	NULL	
D 0			
O 0	LastErr	ERROR_NO_IMPERSONATION_TOKEN (0000051D)	
EFL	00000246	(NO, NB, E, BE, NS, PE, GE, LE)	
ST0	empty	-UNORM BBB0 01050104 00000000	
ST1	empty	0.0	
ST2	empty	0.0	
ST3	empty	0.0	
ST4	empty	0.0	
ST5	empty	0.0	
ST6	empty	0.0	
ST7	empty	0.0	
		3 2 1 0	E S P U O Z D I
FST	0000	Cond 0 0 0 0	Err 0 0 0 0 0 0 0 0 (GT)
FCW	027F	Prec NEAR, 53	Mask 1 1 1 1 1 1

### 3.3.- Dump :



Address	Hex dump	ASCII
00403000	4C 6F 20 6C 6F 67 72 61 73 74 65 20 20 65 73 70	Lo lograste, esp
00403010	65 72 6F 20 71 75 65 20 65 73 74 65 20 63 72 61	ero que este cra
00403020	63 6B 6D 65 20 68 61 79 61 20 73 69 64 6F 20 64	ckme haya sido d
00403030	65 20 74 75 20 61 67 72 61 64 6F 00 53 69 67 75	e tu agrado.Sigu
00403040	65 20 49 6E 74 65 6E 74 61 64 6F 20 6E 6F 20 74	e Intentado no t
00403050	65 20 64 65 73 61 6E 69 6D 65 73 00 20 20 20 3C	e desanimés. <
00403060	3E 20 43 72 61 63 6B 6D 65 20 23 31 20 62 79 20	> Crackme #1 by
00403070	42 6F 63 76 6B 20 3C 3E 0A 0D 0A 0D 20 20 20 20	Bocvk <>....
00403080	20 20 20 20 20 20 20 45 6E 63 75 65 6E 74 72 61	Encuentra
00403090	20 65 6C 20 73 65 72 69 61 6C 0A 0D 20 20 20 20	el serial..
004030A0	20 20 20 20 20 20 20 62 6F 63 76 6B 40 68 6F 74 6D	bocvk@hotmail
004030B0	61 69 6C 2E 63 6F 6D 0A 0D 20 20 20 20 44 65 64	ail.com.. Ded
004030C0	69 63 61 64 6F 20 61 20 43 72 61 63 6B 73 4C 61	icado a CracksLa
004030D0	74 69 6E 6F 73 0A 0D 20 20 20 20 20 20 20 20 20	tinós..
004030E0	20 20 20 20 20 20 50 65 72 FA 20 20 20 32 30 30	Per. - 200
004030F0	38 00 49 6E 66 6F 72 6D 61 63 69 F3 6E 00 42 69	8.Informaci%n.Bi
00403100	65 6E 00 4D 61 6C 00 6B 6D 66 68 75 61 6C 73 00	en.Mal.kmfhuals.
00403110	61 64 68 62 61 73 64 6F 00 73 65 6D 6B 63 61 72	adhasbasdo.senkar

En esta subventana podemos representar los valores hexadecimales, de muchas formas diferentes:

- Podemos ver el carácter ASCII que representa al valor hex.
- Podemos ver el código UNICODE que representa a los valores hex.
- Podemos ver la representación de punto flotante.
- Podemos ver las direcciones que pueden representar.
- Podemos ver incluso el código desensamblado, aunque para esto ya tenemos nuestra ventana de desensamblado.
- Podemos ver un PE header con todos sus datos.

Y muchas opciones más. Una ventana muy útil, que nos va a ayudar enormemente en nuestra labor.

Normalmente se suele utilizar la apariencia de la imagen anterior, que está dividida en 3 columnas.

**Address** --> dirección donde se encuentran los bytes

**Hex dump** --> los valores hexadecimales

**ASCII** --> el carácter ASCII del valor hexadecimal

### 3.4.- Pila :

0012FFC4	7C816D4F	RETURN to kernel32.7C816D4F
0012FFC8	7C920738	ntdll.7C920738
0012FFCC	FFFFFFFF	
0012FFD0	7FFDF000	
0012FFD4	8054A6ED	
0012FFD8	0012FFC8	
0012FFDC	849748E8	
0012FFE0	FFFFFFFF	End of SEH chain
0012FFE4	7C8399F3	SE handler
0012FFE8	7C816D58	kernel32.7C816D58
0012FFEC	00000000	
0012FFF0	00000000	
0012FFF4	00000000	
0012FFF8	00401000	Bocvk.<ModuleEntryPoint>
0012FFFC	00000000	

## 4.- Lo mas usado en el OllyDbg:

### 4.1.- Conceptos :

**Tracear** : Ejecutar línea por línea de su código del programa ahí dos formas con F7 y F8

### 4.2.- Teclas :

**F7** : Ejecuta una sola línea de código (si estas en un CALL entra al mismo a ejecutarlo por dentro)

**F8** : Ejecuta una sola línea de código (si estas en un CALL no entra al mismo lo ejecuta completo sin entrar y sigue en la siguiente línea luego del CALL)

Esas dos formas de tracear manualmente son verdaderamente diferentes y según cada caso usaremos F7 o F8 .

**F2** : Coloca un Breakpoint COMUN en la línea que marcas con el Mouse o esta grisada en el listado, para quitar el BP apretas nuevamente F2.

**F9** : Arrancar el programa (Run)

**CTRL+G** : Ir hacia un dirección determinada por nosotros

**CTRL+N** : Listado de Apis (funciones) utilizadas

*En Construcción*

---

**Autores : karmany y Shaddy**

**Bibliografía :**

**Introducción al Cracking con OllyDBG Desde Cero por Ricardo Narvaja , Team CrackLatinos**

**Clases de MASM - RadASM por ^A|An M0r3N0^ y RedH@wk , RVLCN**

**Modificaciones : Bocvk**

PD :

Con esto he terminado una parte de la Introducción al Cracking espero que les haya servido , pues me cansado de escribir nunca he escrito tutoriales (Papers), así que espero que me ayuden a seguir con esto ,me gustaria saber quien esta interesado en escribir o apoyarme con temas de Cracking, y tambien quisiera saber quienes lo leyeron o mejor a quienes le importa porque no me gustaria escribir tanto si nadie le va a prestar la atención ...

Por favor espero comentarios ,criticas y/o sugerencias ...