

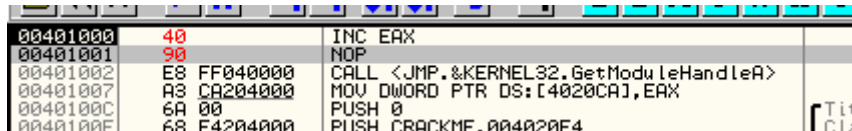
## INTRODUCCION AL CRACKING CON OLLYDBG PARTE 5

### INSTRUCCIONES MATEMATICAS

#### INC Y DEC

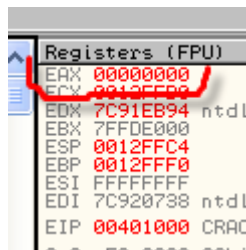
Estas instrucciones incrementan o decrementan respectivamente el operando, sumándole uno si es INC o restándole uno en el caso de DEC.

Veamos en el OLLY, como siempre abrimos el OLLYDBG y el crackme de cruehead y en la primera línea escribimos



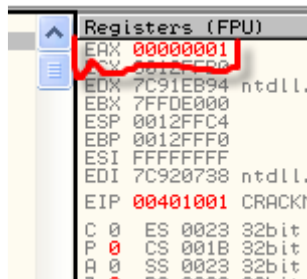
00401000	40	INC EAX
00401001	90	NOP
00401002	E8 FF040000	CALL <JMP.&KERNEL32.GetModuleHandleA>
00401007	A3 C8040000	MOV DWORD PTR DS:[4020CA],EAX
0040100C	6A 00	PUSH 0
0040100F	68 F4204000	PUSH CRACKME.004020F4

EAX en el estado inicial en mi maquina esta a cero, si no puedo cambiarlo a cero a mano



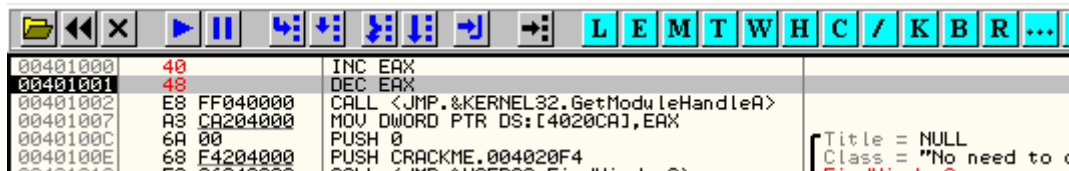
Registers (FPU)	
EAX	00000000
ECX	0012FFF0
EDX	7C91EB94 ntdll
EBX	7FFDE000
ESP	0012FFC4
EBP	0012FFF0
ESI	FFFFFFFF
EDI	7C920738 ntdll
EIP	00401000 CRACKME.00401000

Así que al apretar F7 y ejecutar la instrucción se INCREMENTARA EAX en uno veamos apretemos F7



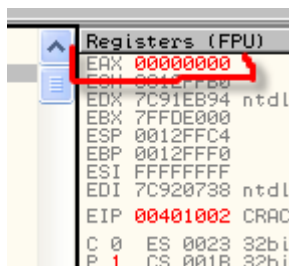
Registers (FPU)	
EAX	00000001
ECX	0012FFF0
EDX	7C91EB94 ntdll
EBX	7FFDE000
ESP	0012FFC4
EBP	0012FFF0
ESI	FFFFFFFF
EDI	7C920738 ntdll
EIP	00401001 CRACKME.00401001

Lo mismo ocurre con DEC podemos escribirlo debajo del anterior



00401000	40	INC EAX
00401001	48	DEC EAX
00401002	E8 FF040000	CALL <JMP.&KERNEL32.GetModuleHandleA>
00401007	A3 C8040000	MOV DWORD PTR DS:[4020CA],EAX
0040100C	6A 00	PUSH 0
0040100E	68 F4204000	PUSH CRACKME.004020F4

Al apretar F7 y ejecutar la instrucción se DECREMENTA EAX que estaba en 1 y volverá a 0.

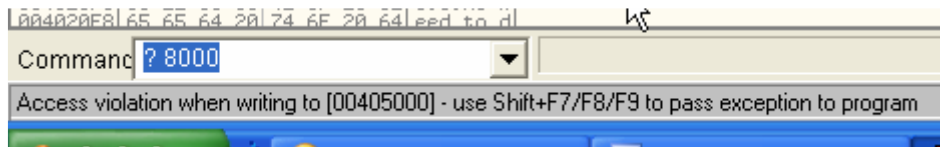


Registers (FPU)	
EAX	00000000
ECX	0012FFF0
EDX	7C91EB94 ntdll
EBX	7FFDE000
ESP	0012FFC4
EBP	0012FFF0
ESI	FFFFFFFF
EDI	7C920738 ntdll
EIP	00401002 CRACKME.00401002

También se pueden incrementar o decrementar el contenido de posiciones de memoria



Aunque en este caso la sección no tiene permiso de escritura y no dejara aumentar el contenido dando excepción.



Si la sección hubiera tenido permiso de escritura, vamos en el dump a la dirección 405000

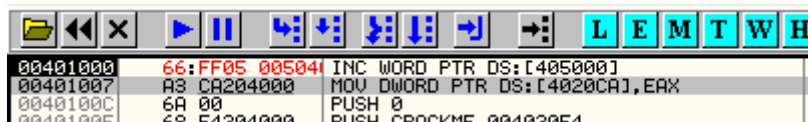
Address	Hex dump	ASCII
00405000	00 10 00 00 DC 00 00 00	.....
00405008	08 30 0F 30 1F 30 33 30	0*0030
00405010	3D 30 46 30 4B 30 58 30	=0F0K0X0

Ya que el contenido al leerlo al revés es 00001000, si lo incrementamos seria 00001001 o sea quedaría así

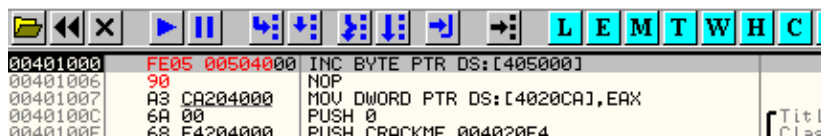
Address	Hex dump	ASCII
00405000	01 10 00 00 DC 00 00 00	.....
00405008	08 30 0F 30 1F 30 33 30	0*0030
00405010	3D 30 46 30 4B 30 58 30	=0F0K0X0

Este fue el caso para DWORD sumándole UNO a los 4 bytes del contenido.

En el caso de WORD el ejemplo sumaria solo a los últimos 2 bytes



Y en el caso de BYTE sumaria solo al último byte



## ADD

Add como ya vimos es la instrucción correspondiente a la suma, siempre suma ambos operandos y guarda el resultado en el primero.

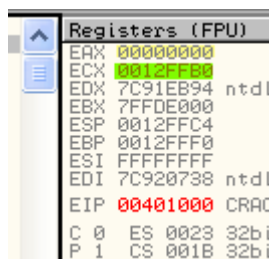
ADD EAX,1 es similar a INC EAX

También puede sumar registros

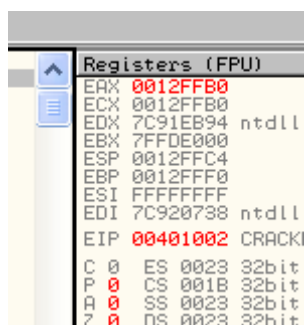
Veamos en OLLY



Antes de ejecutar la operación



En mi maquina EAX vale 00000000 y ECX vale 12FFB0 en sus maquina puede tener otros valores, pueden cambiarlos si quieren, pero al apretar F7 sumara ambos y guardara el resultado en EAX, veamos



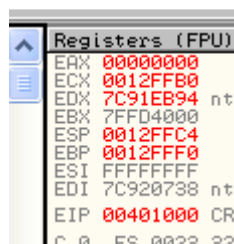
Allí esta EAX esta en rojo pues fue el que se modifico y tiene el resultado de la suma.

También podemos sumar a un registro el contenido de una memoria



En este caso no hay problema por el permiso de escritura ya que como EAX cambiara y guardara el resultado y el contenido de [405000] no cambiara ya que es el segundo operando, la operación no generara excepción.

Antes de apretar F7 vemos que EAX vale 0 y el contenido de 405000 vale 00001000



Address	Hex dump	ASCII
00405000	00 10 00 00 DC 00 00 00	. . . . .
00405008	08 30 0F 30 1F 30 33 30	*00030
00405010	3D 30 46 30 4B 30 58 30	=0F0K0X0
00405018	69 30 6F 30 79 30 7D 30	!0o0y0j0
00405020	85 30 87 30 8C 30 9A 30	5000T0K0

Apreto F7 y se ejecuta la suma

Registers (FPU)	
EAX	00001000
ECX	0012FFB0
EDX	7C91EB94 ntdll.
EBX	7FFD4000
ESP	0012FFC4
EBP	0012FFF0
ESI	FFFFFFFF
EDI	7C920738 ntdll.
EIP	00401006 CRACK
C 0	ES 0023 32bit
P 1	CS 001B 32bit
A 0	SS 0023 32bit

Entonces EAX=0 mas 1000 se modifica EAX quedando el resultado allí que es 1000.

Si hacemos al revés y escribimos

00401000	0105 00504000	ADD DWORD PTR DS:[405000],EAX
00401006	90	NOP
00401007	A3 C8204000	MOV DWORD PTR DS:[4020CA],EAX

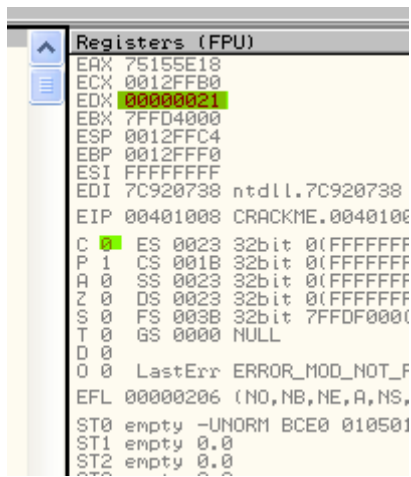
En este caso el resultado se guardara en el contenido de 405000 y esto modificara el mismo por lo cual al apretar F7 generara una excepción al no tener permiso de escritura.

Command	7 8000
Access violation when writing to [00405000] - use Shift+F7/F8/F9 to pass exception to program	

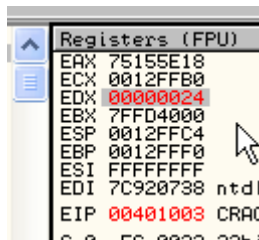
### [ADC \(ADD WITH CARRY\)](#)

En este caso se suman ambos operandos y se le suma el valor del CARRY FLAG O FLAG C y se guarda en el primer operando.

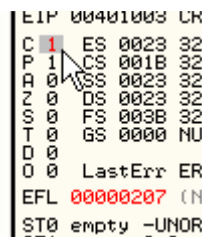
00401000	83D2 03	ADC EDX,3
00401003	90	NOP
00401004	90	NOP
00401005	90	NOP
00401006	90	NOP
00401007	A3 C8204000	MOV DWORD PTR DS:[4020CA],EAX



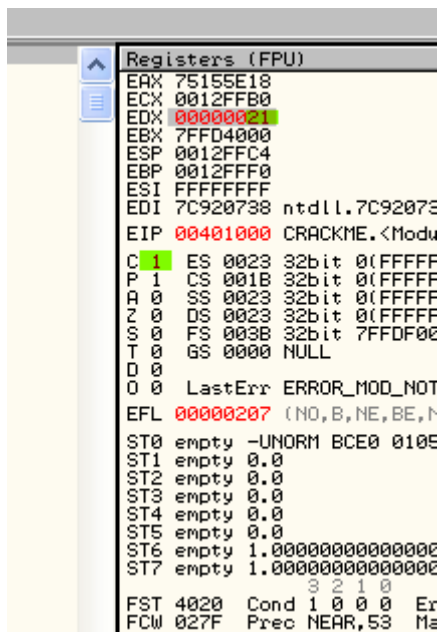
Allí vemos que sumara EDX que vale 21 mas 3 mas el valor del flag C, que en este caso es cero, si apreto F7.



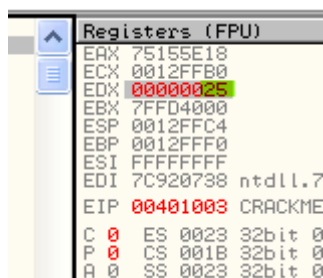
Veo que el resultado es 24, pero si repito la operación con el FLAG C puesto a 1 el cual se puede cambiar haciendo doble click en el mismo.



Allí lo cambie a uno y pongo todo como antes para repetir la operación solo cambiando el flag C.



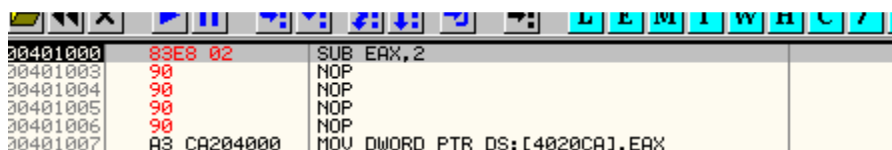
Apreto F7 para que se realice la suma y ahora el resultado es 25



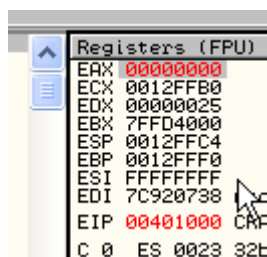
Ya que suma  $EDX = 21$  mas 3 mas el FLAG C que en este caso vale 1.

## SUB

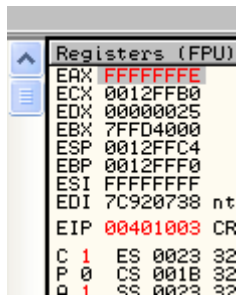
Es la resta o subtracción o sea la operación contraria a ADD, lo que hace es restar el segundo operando al primero y guardar el resultado en el primer operando.



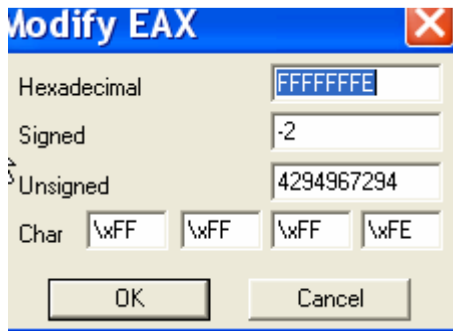
En mi maquina los registros valen antes de ejecutar



Al apretar F7 le restara a EAX que vale cero el valor 2



El resultado es -2 que en hexadecimal se representa FFFFFFFE si hacemos doble click en dicho valor



Vemos que corresponde al decimal -2.

También se pueden restar registros, y posiciones de memoria en la misma forma que lo hicimos con ADD.

SUB EAX,ECX

Por ejemplo hará EAX-ECX y guardara el resultado en EAX

Y

SUB EAX,DWORD PTR DS:[405000]

Restará a EAX el contenido de la posición de memoria 405000, guardando el resultado en EAX.

En el caso inverso

SUB DWORD PTR DS:[405000],EAX

Ya que el resultado se guarda en el primer operando, si no tenemos permiso de escritura en la sección, nos dará una excepción.

## SBB

Es la operación contraria a ADC, es este caso se restan ambos operandos y se le resta el valor del CARRY FLAG O FLAG C y se guarda en el primer operando.



Antes de ejecutar la operación ponemos EDX a 21 y el carry flag a 0

Registers (FPU)				
EAX	00000000			
ECX	0012FFB0			
EDX	00000021			
EBX	7FFD4000			
ESP	0012FFC4			
EBP	0012FFF0			
ESI	FFFFFFFF			
EDI	7C920738	ntdl		
EIP	00401000	CRACK		
C	0	ES	0023	32bit
P	0	CS	001B	32bit
A	1	SS	0023	32bit
Z	0	DS	0023	32bit
S	1	FS	003B	32bit
T	0	GS	0000	NULL
O	0	LastError	ERROR	
EFL	00000292	(NO, I		
ST0	empty	-UNORM		
ST1	empty	0.0		

Al apretar F7 hará EDX-3 y le restara cero del FLAG C

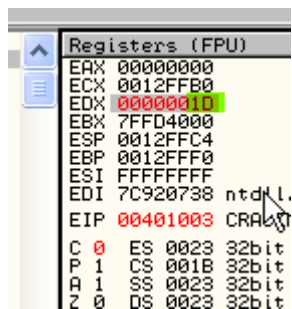
Registers (FPU)				
EAX	00000000			
ECX	0012FFB0			
EDX	0000001E			
EBX	7FFD4000			
ESP	0012FFC4			
EBP	0012FFF0			
ESI	FFFFFFFF			
EDI	7C920738	ntdll.7C		
EIP	00401003	CRACKME.		
C	0	ES	0023	32bit 0(
P	1	CS	001B	32bit 0(
A	1	SS	0023	32bit 0(
Z	0	DS	0023	32bit 0(

Ahora su repito la operación con el FLAG C a 1

Registers (FPU)				
EAX	00000000			
ECX	0012FFB0			
EDX	00000021			
EBX	7FFD4000			
ESP	0012FFC4			
EBP	0012FFF0			
ESI	FFFFFFFF			
EDI	7C920738	ntdll.7C		
EIP	00401000	CRACKME		
C	1	ES	0023	32bit 0
P	1	CS	001B	32bit 0
A	1	SS	0023	32bit 0
Z	0	DS	0023	32bit 0
S	0	FS	003B	32bit 71
T	0	GS	0000	NULL
O	0	LastError	ERROR_M	
EFL	00000217	(NO,B,NI		
ST0	empty	-UNORM BCE		
ST1	empty	0.0		
ST2	empty	0.0		
ST3	empty	0.0		
ST4	empty	0.0		
ST5	empty	0.0		

Apreto F7 y ahora hará EDX-3 y le restara 1 del FLAG C





En este caso el resultado es 1D.

## MUL

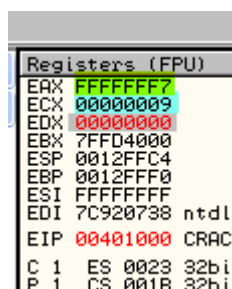
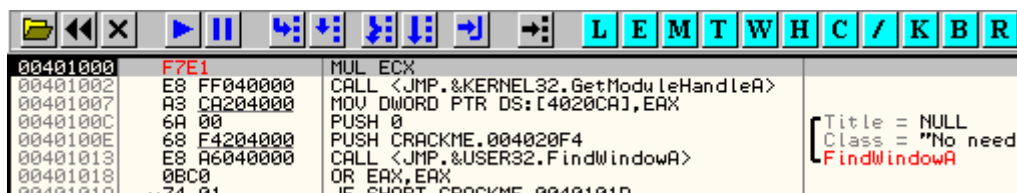
Bueno hay dos instrucciones para realizar multiplicaciones, la primera de ellas es MUL, la cual no considera los signos de los números que va a multiplicar, y usa un solo operando el otro operando es siempre EAX aunque no se escribe y el resultado lo guarda en EDX:EAX que quiere decir esto veamos el ejemplo.

Por ejemplo

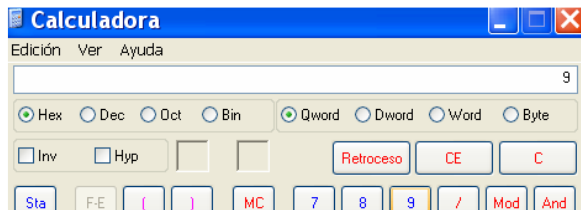
MUL ECX

Esto multiplicara ECX por EAX y guardara el resultado en EDX:EAX y no considerara el signo de los operandos.

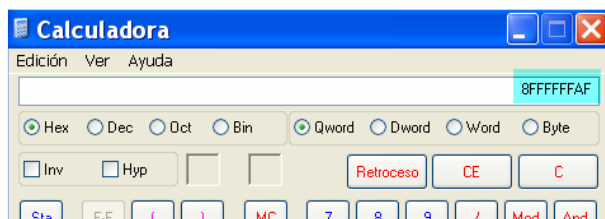
Por ejemplo veamos en OLLYDBG



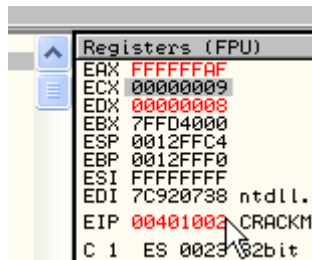
Pongo EAX a FFFFFFFF7 y ECX a 9 si realizo la multiplicación en la calculadora de Windows veo que



El resultado es



Y no entra en EAX veamos que pasa en OLLY al apretar F7



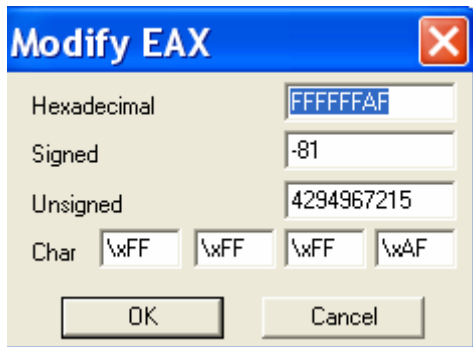
Allí vemos en rojo que EDX y EAX cambiaron y como vemos guarda en EAX los bytes que entran allí y luego guarda los que no entran en EDX, en este caso el 8 que no puede mostrar EAX, lo guardo en EDX, por eso se dice que en este caso el resultado se muestra en EDX:EAX ya que usa ambos como si fueran un solo registro del doble de largo.

En el caso

MUL DWORD PTR DS:[405000]

Multiplicara el contenido de 405000 por EAX y como siempre guardara el contenido en EDX:EAX sin considerar el signo de los operandos.

Siempre que en OLLY queramos ver cuanto es el valor de un numero hexadecimal sin signo, hacemos doble click en cualquier registro



Como habíamos visto la segunda línea nos daba el valor con signo en este caso FFFFFFFAF es -81 considerado con SIGNO, pero en operaciones como MUL donde los números se consideran SIN SIGNO el valor decimal se lee en la tercera línea la que dice UNSIGNED, allí vemos 4294967215 sería el valor tomando al número como positivo o sin SIGNO.

### IMUL (multiplicación con signo)

La instrucción IMUL no solo es multiplicar con signo de la misma forma que lo hacía MUL

IMUL ECX

Es igual que antes ECX por EAX y el resultado se guarda en EDX:EAX salvo que se considera el signo de los operandos.

Además de la similitud con la instrucción anterior IMUL permite poner más de un operando, lo que no estaba permitido en MUL.

Del tute de CAOS

Además de la utilización de los registros EAX y EDX, así como de sus subdivisiones, pueden especificarse otros orígenes y destinos de datos y puede haber hasta tres operandos. El primero, es el lugar donde se va a guardar el resultado, que debe ser siempre un registro, el segundo y el tercero son los dos valores a multiplicar. En estos ejemplos vemos cómo estas instrucciones con dos o tres operandos, tienen el mismo espacio para el resultado que para cada uno de los factores:

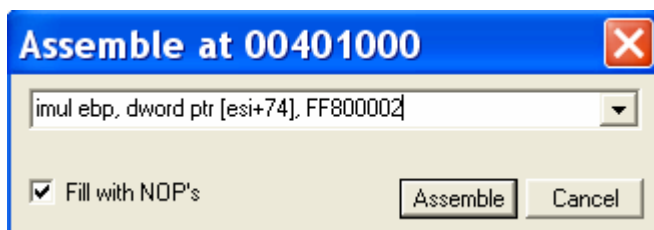
F7EB                      imul ebx                      EAX x EBX -> EDX:EAX

Este primer ejemplo es el conocido y similar a MUL salvo que se consideran los signos

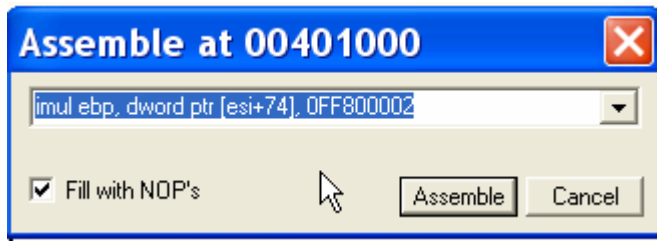
696E74020080FF              imul ebp, dword ptr [esi+74], FF800002              [ESI+74] x FF800002 -> EBP

Este es el segundo ejemplo que vemos en este caso hay tres operandos, multiplica el contenido de ESI+74 por FF800002 y el resultado lo guarda en EBP, podemos hacerlo en OLLY

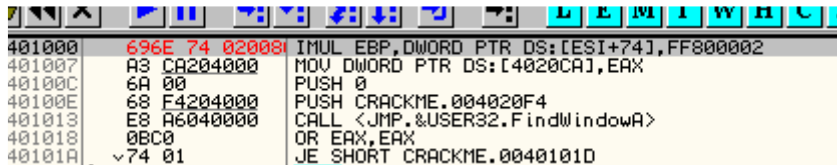
Copio la línea `imul ebp, dword ptr [esi+74], FF800002` al OLLY



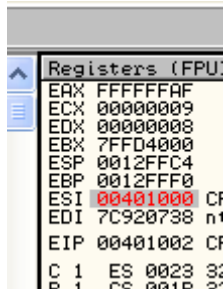
Veo que al apretar ASSEMBLE me da error y eso es porque en el OLLY los números que empiezan por letras deben agregársele un cero delante si no, no los interpreta, corrijamos



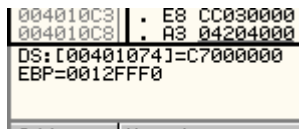
Ahora apreto ASSEMBLE y lo acepta



Cambio el valor de ESI a 401000 para asegurarme que la dirección ESI mas 74 exista y se pueda leer su contenido



Veamos en la aclaración del OLLY



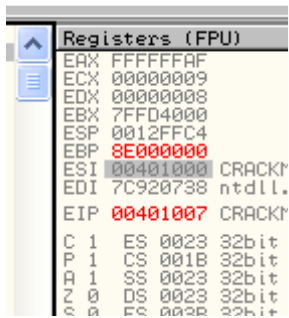
Allí nos dice que ESI + 74 es la dirección 401074 y que su contenido es C7000000, veamos en el dump con GOTO EXPRESSION 401074.

Address	Hex dump	ASCII
00401074	00 00 00 C7 05 84 20 40	...Z* @
0040107C	00 10 21 40 00 C7 05 80	...!@.Z* @
00401084	20 40 00 F4 20 40 00 68	@. @ @.h
0040108C	64 20 40 00 E8 F3 03 00	d @. @.h
00401094	00 6A 00 FF 35 CA 20 40	.j. 5 @
0040109C	00 6A 00 6A 00 68 00 80	.j.j.h.Ç
004010A4	00 00 68 00 80 00 00 6A	..h.Ç..j
004010AC	6E 68 B4 00 00 00 68 00	nh-...h.
004010B4	00 CF 00 68 E7 20 40 00	.@.h @.
004010BC	68 F4 20 40 00 6A 00 E8	h @. @.j. @
004010C4	CC 03 00 00 A3 04 20 40	if...u @

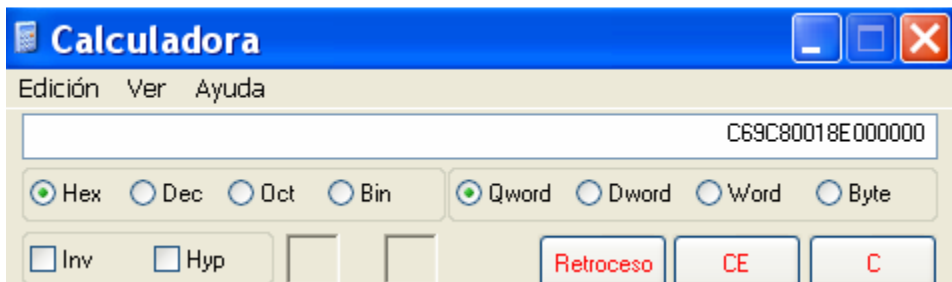
Y si, es cierto el contenido leído al revés sería C7000000, eso lo multiplicara por FF800002 y como el primer operando es EBP pues guardara allí el resultado

`imul ebp, dword ptr [esi+74], FF800002`

Al apretar F7 vemos que se puso rojo EBP ya que ahora contiene el resultado



El resultado en la calculadora nos da  $C7000000 * FF800002$



Pero como especificamos que se muestre en EBP pues solo muestra los bytes que caben allí, el resto los descarta.

En el tercer ejemplo que hay solo dos operandos se multiplican ambos y el resultado se guarda en el primero.

0FAF55E8

`imul edx, dword ptr [ebp-18]`

$EDX \times [EBP-18] \rightarrow EDX$

Como vemos la mejor opción para multiplicar números largos es usando IMUL con solo un operando pues en este caso el resultado lo guarda en EDX:EAX con la posibilidad del doble de largo lo cual no ocurre cuando usamos dos o tres operandos, dichas opciones son mas útiles en operaciones pequeñas.

### [DIV \(Unsigned Divide\) / IDIV \(Signed Divide\)](#)

Estos son la contrapartida de MUL Y IMUL respectivamente

DIV solo tiene un operando y no considera los signos y el resultado se guarda en EDX:EAX

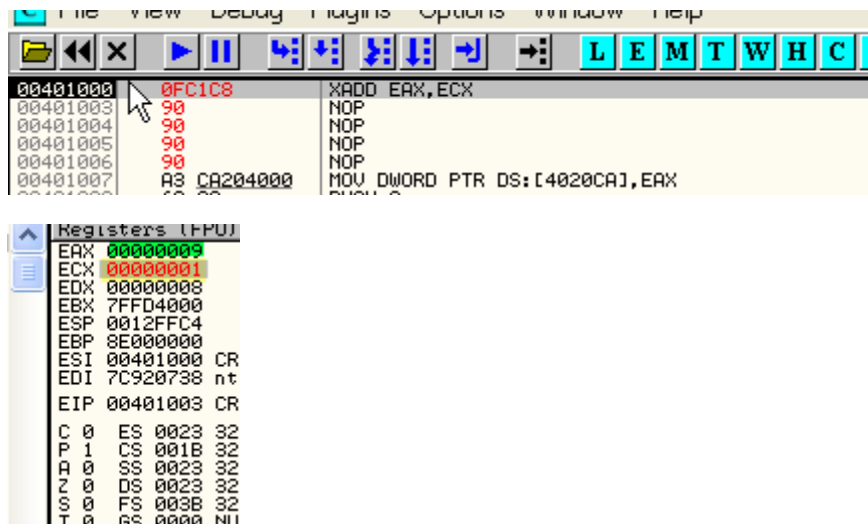
IDIV siempre considera los signos si usa un solo operando será como DIV y guardara en EDX:EAX y en el caso de dos operandos dividirá ambos y guardara en el primero, y en el de tres operandos dividirá el segundo y el tercero y guardara en el primero.

No creo que sea necesario repetir los ejemplos pues son similares a los de MUL e IMUL.

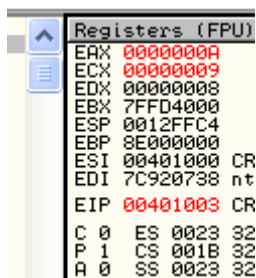
### [XADD \(Exchange and Add\)](#)

Es como realizar en una sola instrucción XCHG y ADD o sea que si tenemos

`XADD EAX,ECX`



Allí vemos ECX que es 1 y EAX es 7 al apretar F7 se intercambian o sea que EAX pasa a valer 1 y ECX 9 luego se suman

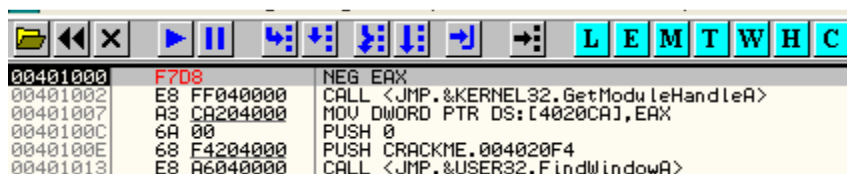


Como vemos el resultado se sigue guardando en el primer operando, lo único que cambio fue que se intercambiaron los valores antes de sumarlos, vemos que ECX quedo valiendo 9 que era lo que valía EAX antes de intercambiarse y sumarse.

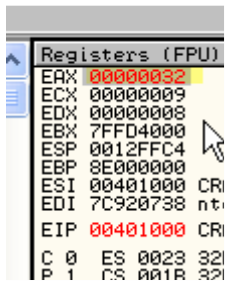
## NEG

Esta instrucción tiene la finalidad de cambiar de signo el número representado o sea que si tenemos el número 32 en hexa y le aplicamos NEG el resultado será el negativo del mismo.

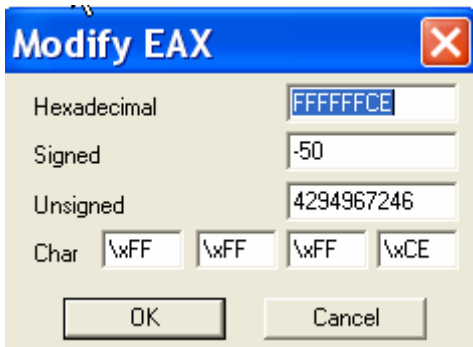
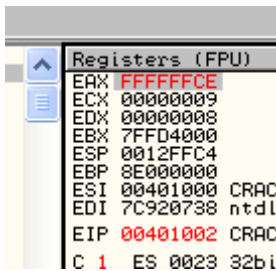
Ejemplo



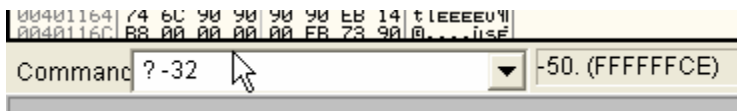
Escribo en OLLY NEG EAX y pongo en EAX el valor 32



Al apretar F7 quedara en EAX el negativo de 32 veamos



Allí vemos el resultado -50 en decimal como nos muestra la segunda columna corresponde a -32 en hexa.



Allí esta si tipeamos en la comandbar que nos de el valor de -32 hexa en decimal, nos dice -50 y nos aclara que se escribe FFFFFFFE ya que no se puede escribir el signo en OLLY.

Pues como vemos la instrucción NEG nos convierte el operando en su negativo.

## **INSTRUCCIONES LOGICAS**

Proviene de realizar operaciones lógicas entre dos operandos pasados a binario bit a bit y guardando el resultado en el primer operando

### **AND**

El resultado es 1 si los dos bits son 1, y 0 en cualquier otro caso.

- 1 and 1 = 1
- 1 and 0 = 0
- 0 and 1 = 0

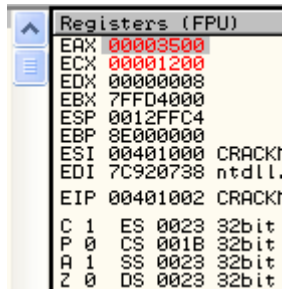
0 and 0 = 0

Vemos un ejemplo en OLLYDBG

AND EAX,ECX

00401000	21C1	AND ECX,EAX
00401002	E8 FF040000	CALL <JMP.&KERNEL32.GetModuleHandleA>
00401007	A3 CA204000	MOV DWORD PTR DS:[4020CA],EAX
0040100C	40 00	PUSH 0

Pongamos ECX=0001200 y EAX=3500



Si lo hiciéramos a mano deberíamos pasar a binario ambos

1200 en binario seria 01001000000000

3500 en binario seria 11010100000000

Aplicándole la tablita de la operación AND bit a bit vemos que por ejemplo la ultima cifra

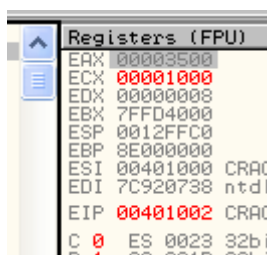
1200 en binario seria 01001000000000  
3500 en binario seria 11010100000000  
0

Al hacer AND entre dos ceros el resultado seria cero, así hay que hacer bit a bit y nos daría

01000000000000 ya que el resultado es uno solo cuando los dos bits son 1 y eso ocurre solo en la columna resaltada.

1200 en binario seria 01001000000000  
3500 en binario seria 11010100000000  
**01000000000000**

Si ejecutamos F7 en el OLLY vemos el resultado en ECX que es 1000 que pasado a binario es 01000000000000





## OR

En esta instrucción realizamos el mismo proceso que la anterior solo que en vez de utilizar la tablita AND para hallar el resultado entre bits, lo hacemos con la tablita OR

El resultado es 1 si uno o los dos operandos es 1, y 0 en cualquier otro caso.

1 or 1 = 1  
1 or 0 = 1  
0 or 1 = 1  
0 or 0 = 0

## XOR

Aquí es similar solo que usamos la tablita de la función XOR para las operaciones entre bits

El resultado es 1 si uno y sólo uno de los dos operandos es 1, y 0 en cualquier otro caso

1 xor 1 = 0  
1 xor 0 = 1  
0 xor 1 = 1  
0 xor 0 = 0

## NOT

Simplemente invierte el valor del único operando de esta función

not 1 = 0  
not 0 = 1

Ejemplo: not 0110 = 1001

Si tenemos por ejemplo EAX=1200 que en binario es 100100000000 convertimos los 0 en 1 y los 1 en 0 considerando que es un numero de 32 bits y que al inicio tiene ceros o sea seria llenado de ceros delante hasta completar los 32 bits.

000000000000000000000000100100000000

Al hacerle NOT quedaría

111111111111111111111111011011111111

que en la calculadora de Windows vemos que es FFF FEDFF en hexa

```

00401000 F7D0 NOT EAX
00401002 E8 FF040000 CALL <JMP.&KERNEL32.GetModuleHandleA>
00401007 A3 C0204000 MOV DWORD PTR DS:[4020CA],EAX
0040100C 6A 00 PUSH 0
0040100E 68 F4204000 PUSH CRACKME.004020F4
00401013 E8 A6040000 CALL <JMP.&USER32.FindWindowA>
00401018 0BC0 OR EAX,EAX
0040101A 74 01 JE SHORT CRACKME.0040101D
0040101C C3 RETN
0040101D C705 64204000 MOV DWORD PTR DS:[402064],4003

```

En OLLY antes de apretar F7 pongo EAX a 1200

```

Registers (FPU)
EAX 00001200
ECX 00001000
EDX 00000008
EBX 7FFD4000
ESP 0012FFC0
EBP 8E000000
ESI 00401000
EDI 7C920738
EIP 00401000
C 0 ES 0023 32bit
P 1 CS 001B 32bit
A 0 SS 0023 32bit

```

Apreto F7

```

Registers (FPU)
EAX FFFFE0FF
ECX 00001000
EDX 00000008
EBX 7FFD4000
ESP 0012FFC0
EBP 8E000000
ESI 00401000
EDI 7C920738
EIP 00401002
C 0 ES 0023 32bit
P 1 CS 001B 32bit
A 0 SS 0023 32bit

```

Vemos que el resultado es el mismo

Bueno aquí terminamos esta quinta parte veo que esto es un poco mas largo de lo que pensaba pero bueno vamos paso a paso, nos quedan ver las comparaciones, los saltos y los call y ret.

Bueno paciencia que despacio se llega a ROMA

Hasta la parte 6

Ricardo Narvaja

14 de noviembre de 2005