INTRODUCTION TO THE CRACKING WITH OLLYDBG

FROM CRACKLATINOS

(_kienmanowar_)

I. Lời nói đầu

Khà khà lâu quá rồi tí nữa thì quên mất cái pr0j3ct về Olly còn đang dang dở. Cũng lâu quá rồi nên chẳng nhớ những bài trước mình viết những gì và viết đến đâu, hôm nay lục lại thì thấy mình mới viết đến bài thứ tư ☺, chà chậm quá. Tối nay rỗi rãi và cũng muốn tránh tình trạng Pending quá dài, tôi viết tiếp phần 5 trong loạt tuts về Ollydbg mà tôi ấp ủ. 0k13! L3t's R0ck w1th m3 ☺

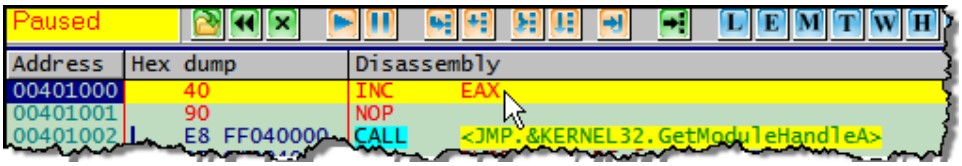
II. Giới thiệu chung

Trong phần trước của bài viết tôi đã đề cập đến những câu lệnh hay dùng nhất, chung nhất khi chúng ta làm quen với Olly. Trong phần 5 này tôi sẽ tiếp tục với các câu lệnh liên quan tới việc tính toán cũng như các lệnh logic, nói tóm lại chưa có gì đặc biệt hơn và có thể gây nhàm chán với những anh em nào đã quá Pr0 khà khà ☺. Xin nhắc lại một lần nữa việc cung cấp tất cả các lệnh vượt quá khuôn khổ cho phép của bài viết, cũng như tôi cũng không đủ sức để mà thực hiện điều này. Do đó việc tham khảo thêm các nguồn tài liệu khác để bổ sung thêm kiến thức là điều hết sức cần thiết cho các bạn.

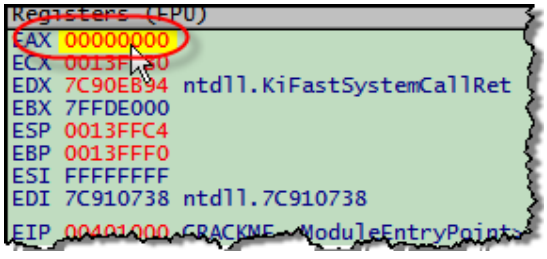
III. Các lệnh dùng trong việc tính toán

1. Lệnh INC và DEC :

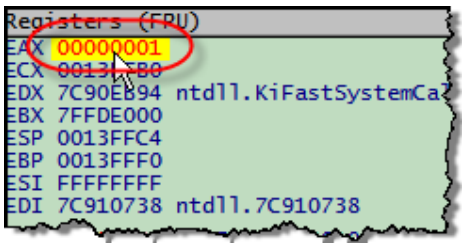
INC (Increment) được dùng để cộng thêm 1 vào nội dung của một thanh ghi hay một ô nhớ. *DEC (Decrement)* thực hiện công việc ngược lại, trừ 1 từ nội dung của một thanh ghi hay ô nhớ. Để minh họa cho hai câu lệnh này bạn load crackme vào trong Olly và edit như sau :



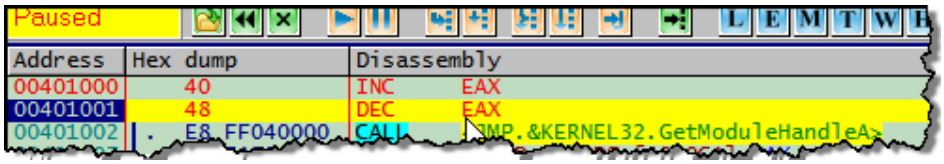
Chuyển qua cửa sổ Register và quan sát giá trị của thanh ghi EAX trước khi chúng ta thực hiện câu lệnh trên là bao nhiêu :



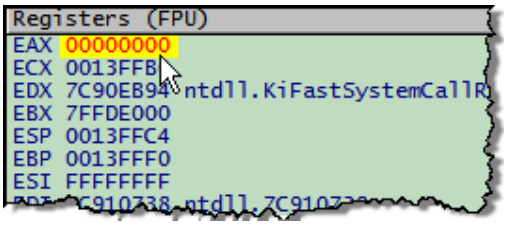
Vậy khi chúng ta thực hiện lệnh `INC EAX` thì tức là tương được với việc ta thực hiện `EAX = EAX + 0x1`.Nhấn F7 để trace qua câu lệnh INC EAX ta có được kết quả như sau :



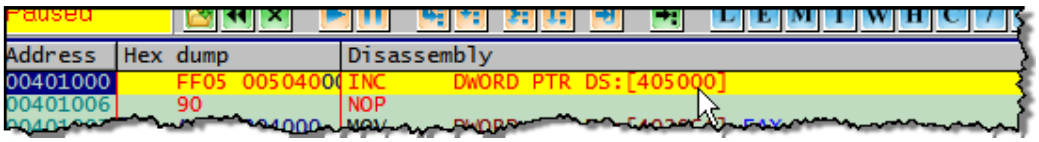
Rất dễ hiểu phải không ? Tiếp tục với lệnh DEC, tôi edit lại lệnh NOP thành DEC EAX :



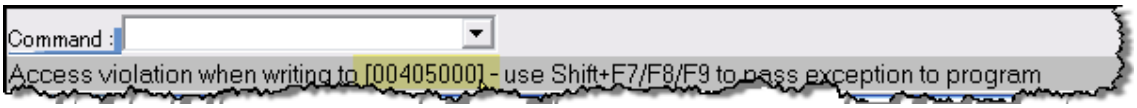
Nhấn F7 để trace qua lệnh này, quan sát cửa sổ Registers sau khi thực hiện lệnh ta sẽ có được kết quả của EAX. Kết quả này tương đương với việc ta thực hiện `EAX = EAX - 0x1`.



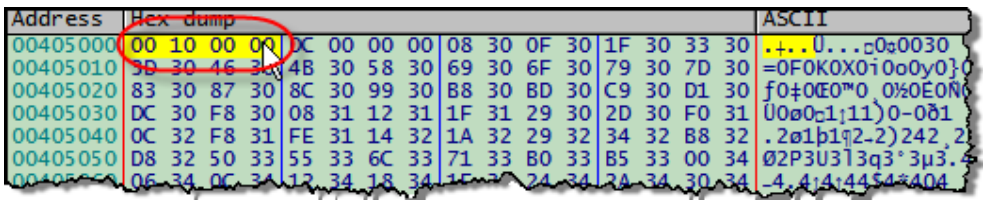
Như các bạn thấy ở trên là những ví dụ đối với thanh ghi, tiếp theo tôi sẽ lấy ví dụ với ô nhớ.Trong Olly các bạn sửa thành lệnh sau :



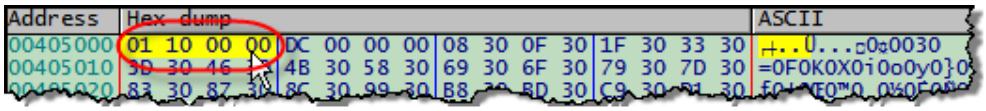
Trong phần 4 tôi có để cập đến quyền truy cập tại section có địa chỉ `0x405000`, do nó không có quyền ghi cho nên khi ta thực hiện câu lệnh INC đối với nội dung của nó sẽ thì sẽ không thực hiện được và chúng ta sẽ dính exception trong Olly :



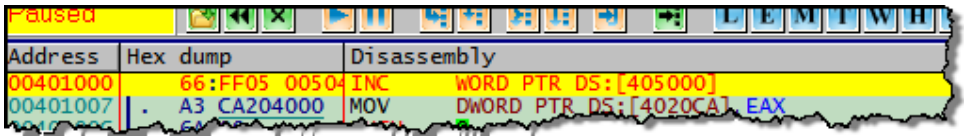
Nhưng nếu như tôi gán lại quyền cho section này và thực hiện câu lệnh trên thì kết quả sẽ ra sao? Trước tiên ta cần xem xét nội dung tại [405000] đã :

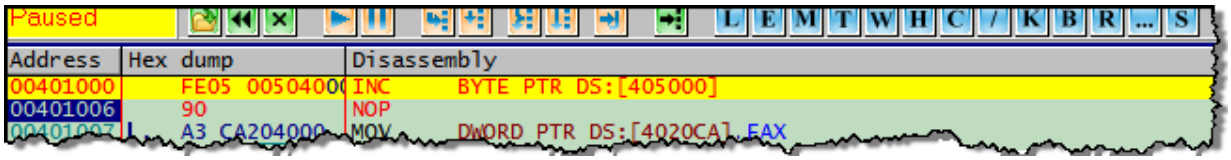


Vậy là giá trị tại ô nhớ này là 0x00001000, theo lý thuyết nếu ta thực hiện lệnh INC thì kết quả sẽ là 0x00001001. Ok bây giờ kiểm tra lại trong Olly :



Kết quả trong Olly đúng như những gì chúng ta đã tính toán trên lý thuyết. Đây là ví dụ thực hiện với DWORD, bạn có thể tự thực hành thêm các ví dụ với WORD hoặc BYTE :



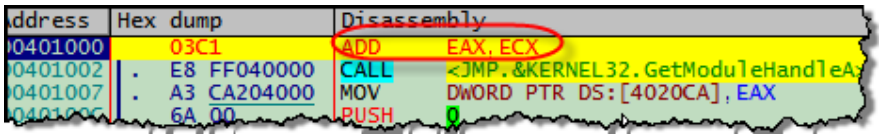


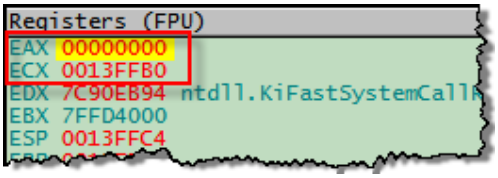
2. Lệnh ADD :

Lệnh này được sử dụng để cộng nội dung của hai thanh ghi, một thanh ghi và một ô nhớ hoặc cộng một số vào một thanh ghi hay một ô nhớ, kết quả sẽ được lưu vào toán hạng đầu tiên. Lấy một ví dụ như sau :

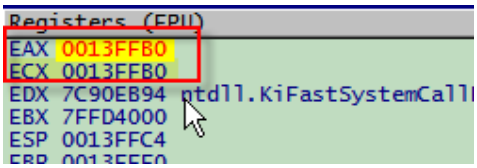
```
ADD WORDX, EAX
```

Lệnh này sẽ thực hiện "cộng EAX vào WORDX", tức là cộng nội dung của thanh ghi EAX với nội dung của ô nhớ WORDX và chứa tổng cuối cùng trong WORDX và EAX không bị thay đổi. Vậy nếu ta có câu lệnh ADD EAX, 1 thì sẽ tương đương với INC EAX. Ta thực hiện một ví dụ nhỏ trong Olly :

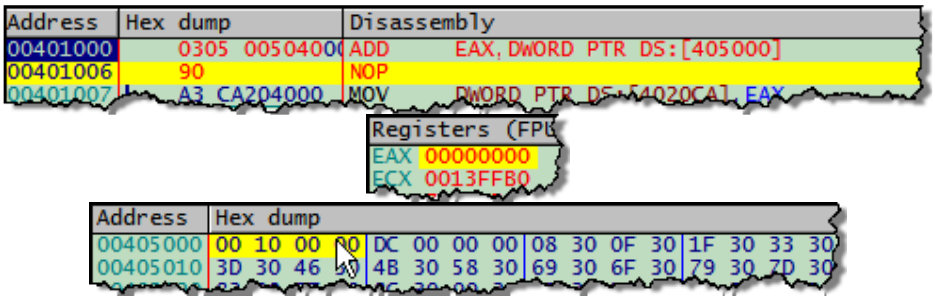




Giá trị hai thanh ghi EAX và ECX trên máy tôi có thể sẽ khác với máy của các bạn. Nhấn F7 để thực hiện lệnh ADD và quan sát trên cửa sổ Registers, ta thấy như sau :



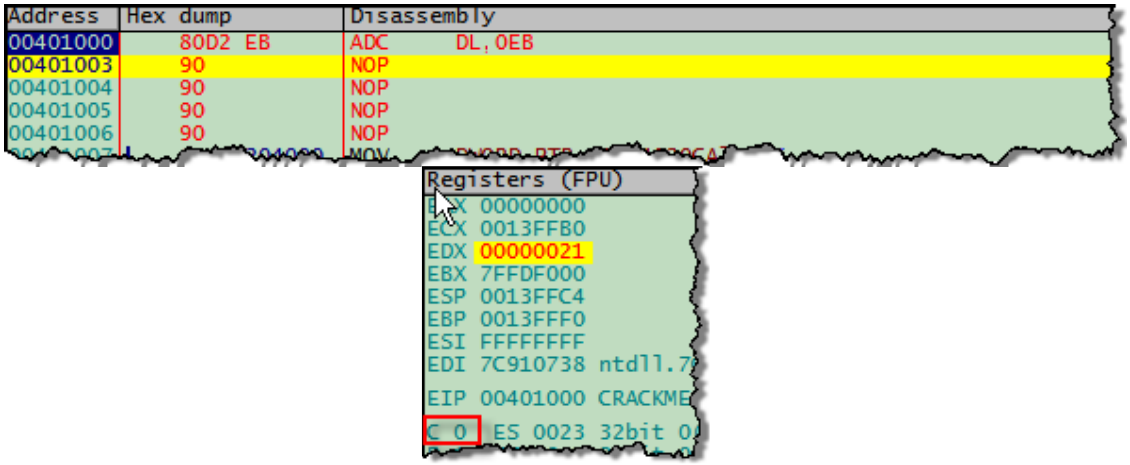
Kết quả sau phép cộng EAX lưu giá trị cuối cùng là tổng và thanh ghi ECX vẫn giữ nguyên không thay đổi.
Thực hiện thêm một ví dụ minh họa nữa, lần này là cộng nội dung thanh ghi với nội dung ô nhớ :



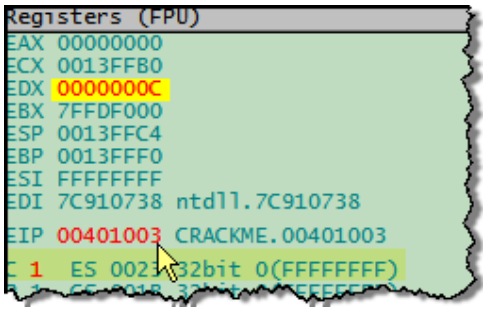
Nhấn F7 để thực hiện lệnh ADD ta có được giá trị của EAX như sau :

3. Lệnh ADC

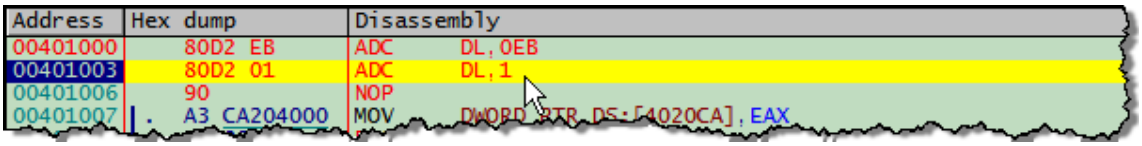
Lệnh này tạm dịch là lệnh cộng có nhớ, tức là Cờ nhớ được cộng vào tổng của toán hạng nguồn và toán hạng đích. Nếu cờ CF=1 thì *toán hạng đích = toán hạng nguồn + toán hạng đích + 1*. Còn nếu CF=0 thì *toán hạng đích = toán hạng nguồn + toán hạng đích*. Ví dụ minh họa :



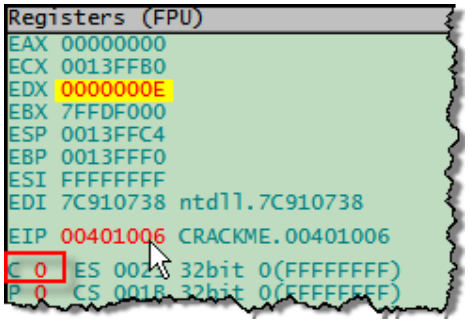
Ta thấy rằng lúc này cờ C đang là 0, thanh ghi EDX lúc này mang giá trị là 0x21(giá trị của DL là 0x21). Ta sẽ thực hiện lệnh ADC DL, 0xEB tức là `DL = DL + 0xEB = 0x21 + 0xEB`. Nhấn F7 để thực hiện lệnh, rõ ràng trong quá trình cộng sẽ có nhớ lên bit MSB cho nên cờ C sẽ được set là 1 :



Ok tiếp tục thực hiện 1 lệnh nữa như sau :

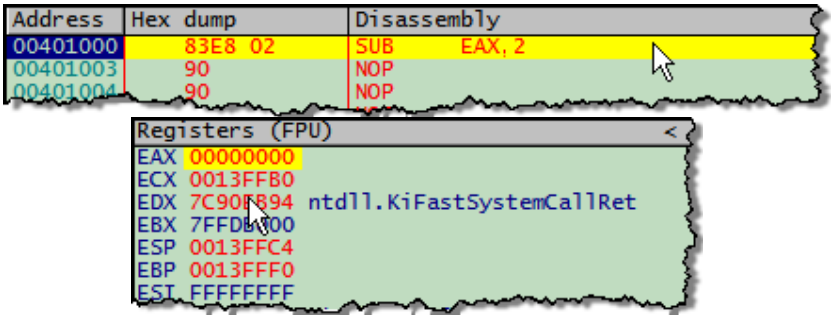


Lúc này giá trị của DL đang là 0xC, cờ C đang được thiết lập là 1. Nếu ta thực hiện lệnh ADC DL,1 tức là `DL = DL + 1 + 1 = 0xE` và cờ C lại được set lại thành 0 ☺.

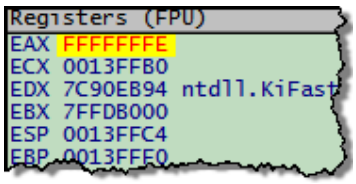


4. Lệnh SUB

Lệnh này thực hiện công việc ngược lại của lệnh ADD. Ví dụ minh họa :



Nhấn F7 để thực hiện lệnh SUB EAX,2. Theo tính toán thông thường của chúng ta thì 0 – 2 sẽ cho kết quả là -2. Giá trị của -2 được biểu diễn ở dạng Hexa là : 0xFFFFFFFE

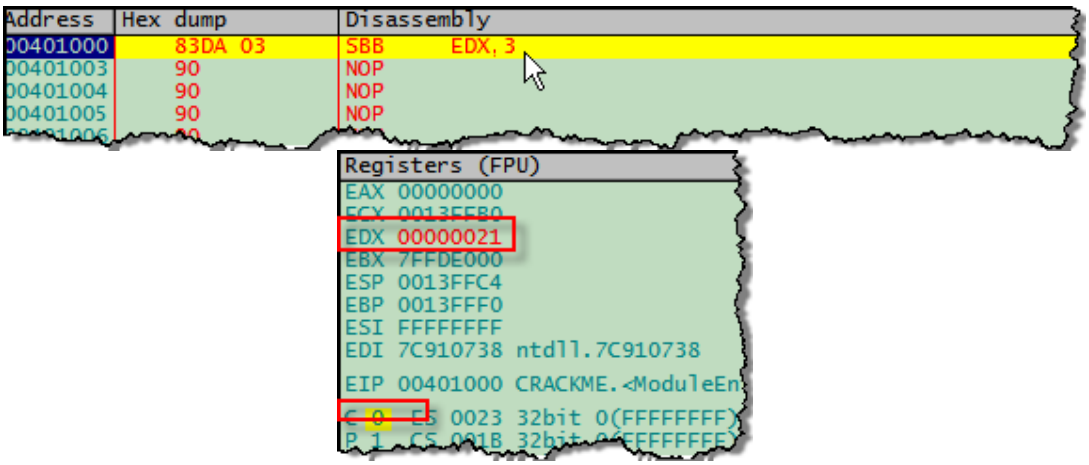


Các bạn hoàn toàn có thể thực hiện thêm các ví dụ như trừ 2 thanh ghi : `SUB EAX, ECX`

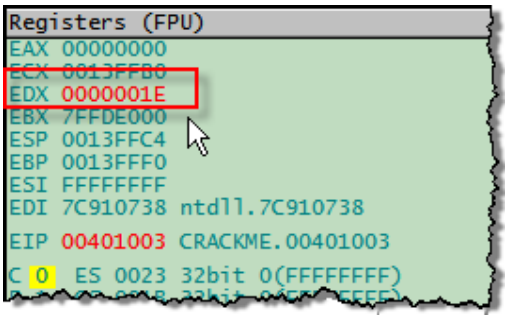
, trừ thanh ghi và ô nhớ : `SUB EAX, DWORD PTR DS:[405000]`

5. Lệnh SBB

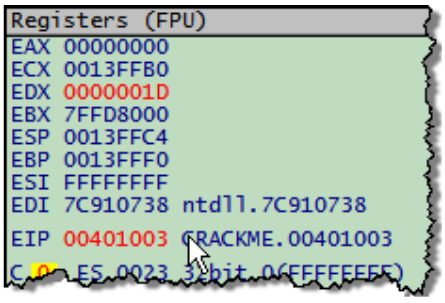
Là lệnh trừ có nhớ, lệnh này thực hiện trừ toán hạng đích cho toán hạng nguồn và nếu CF=1 thì trừ kết quả nhận được đi 1. Kết quả chứa trong toán hạng đích. Tôi lấy một ví dụ nhỏ để minh họa :



Cờ C lúc này đang có giá trị là 0, do đó khi ta thực hiện lệnh SBB thì sẽ tương đương với `EDX = EDX - 3`. Kết quả ta có được :



Ngược lại nếu ta set cờ C có giá trị 1 thì kết quả sẽ là như sau :



6. Câu lệnh MUL

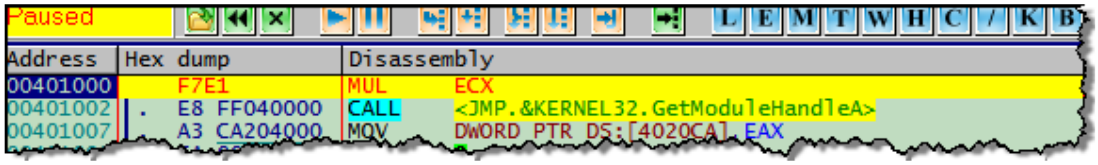
Nhân số không dấu, trong trường hợp này toán hạng gốc là số nhân. Tùy theo độ dài của toán hạng gốc mà ta có ba trường hợp để tổ chức phép nhân :

1. Nếu gốc là số 8 bit: AL * Gốc
   Số bị nhân phải là số 8 bit để trong AL.

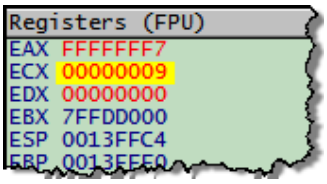
Sau khi nhân : AX <- tích

2. Nếu gốc là số 16 bit: AX * Gốc
   Số bị nhân phải là số 16 bit để trong AX
   Sau khi nhân : DX AX <- tích
3. Nếu gốc là số 32 bit: EAX * Gốc
   Số bị nhân phải là số 32 bit để trong EAX
   Sau khi nhân : EDX EAX <- tích

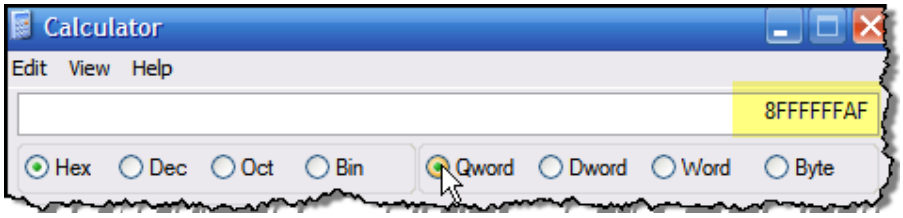
Ta lấy một ví dụ minh họa như sau :



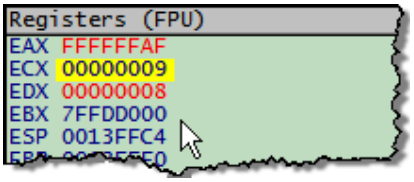
Câu lệnh MUL ECX sẽ thực hiện nhân ECX với EAX sau đó kết quả thu được sẽ được lưu vào EDX:EAX và không quan tâm tới dấu của các toán hạng. Bây giờ ta giả sử giá trị của các thanh ghi EAX, ECX và EDX như sau :



Trước khi xem kết quả thu được trong Olly chúng ta hãy tính toán thử đã. Sử dụng calculator của Windows, ta lấy 0xFFFFFFF7 * 0x9 được kết quả như sau :



Như vậy bạn thấy kết quả ở đây vượt quá sự biểu diễn của thanh ghi EAX do đó nó sẽ được lưu một phần vào thanh ghi EXD. Vậy kết quả cuối cùng sẽ là thanh ghi EDX lưu giá trị 0x8 còn EAX sẽ là 0xFFFFFFAF. Trong Olly ta trace qua lệnh để xem kết quả :



Các bạn tự mình tìm hiểu thêm trong các tài liệu về lệnh này ☺.

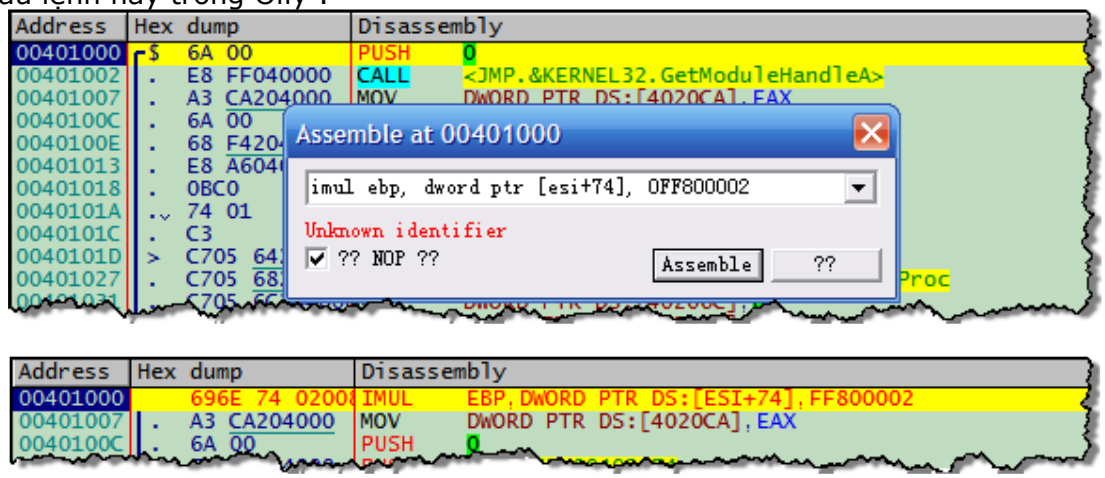
7. Câu lệnh IMUL

Nhân số có dấu, cách thức thì cũng tương tự như lệnh MUL. Ta lấy 2 ví dụ như sau :

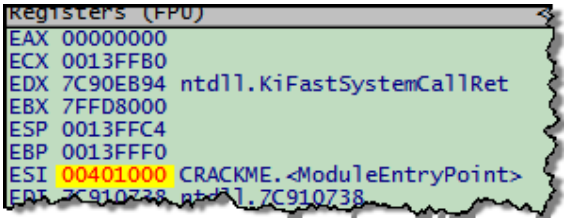
a. `IMUL EBX` : Câu lệnh này sẽ tương đương với việc lấy EAX * EBX và kết quả sẽ được lưu vào EDX:EAX tương tự như lệnh MUL nhưng trong đó dấu của toán hạng được quan tâm đến.

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

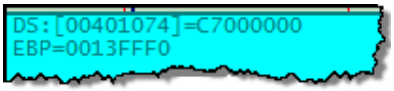
Trong ví dụ thứ hai này chúng ta sẽ thấy có 3 toán hạng và nó thực hiện như sau, lấy nội dung của [ESI+74] nhân với FF800002 kết quả được bao nhiêu lưu vào EBP.Ta sẽ thử thực hiện câu lệnh này trong Olly :



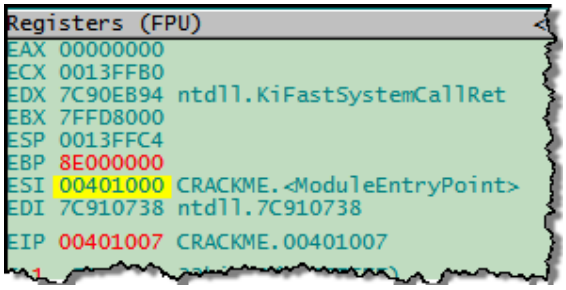
Ta sửa lại giá trị của thanh ghi ESI thành 0x401000 để chúng ta chắc chắn rằng giá trị của [ESI+74] là có và thể đọc được nội dung của nó.



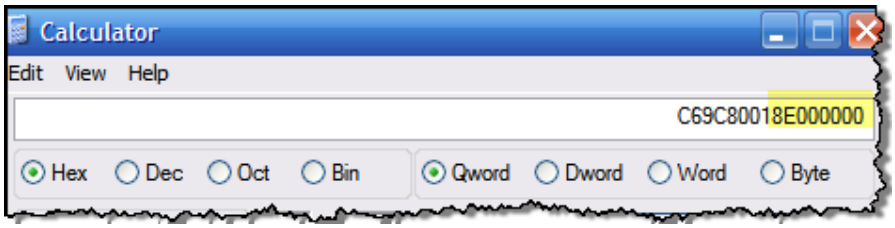
Ok sau khi đổi chúng ta xem nội dung của nó là gì :



Ta nhấn thử F7 để thực hiện câu lệnh IMUL :

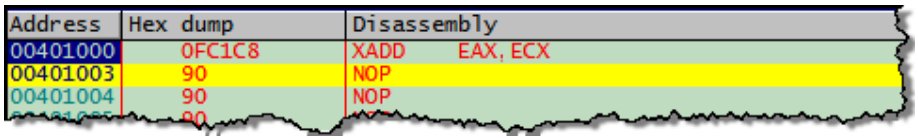


Nếu ta tính toán trong calculator của windows ta sẽ được như sau :

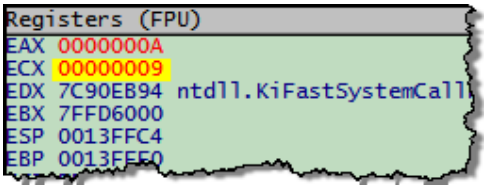


8. Câu lệnh XADD

Câu lệnh này đồng thời thực hiện lần lượt hai lệnh XCHG và ADD. Ví dụ : XADD EAX, ECX thì lúc này giá trị của ECX được thay thế bằng giá trị của EAX còn giá trị của EAX = EAX + ECX. Minh họa :

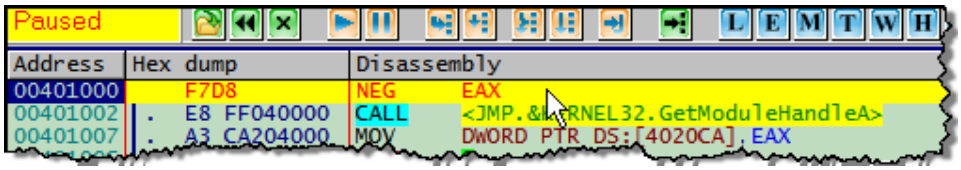


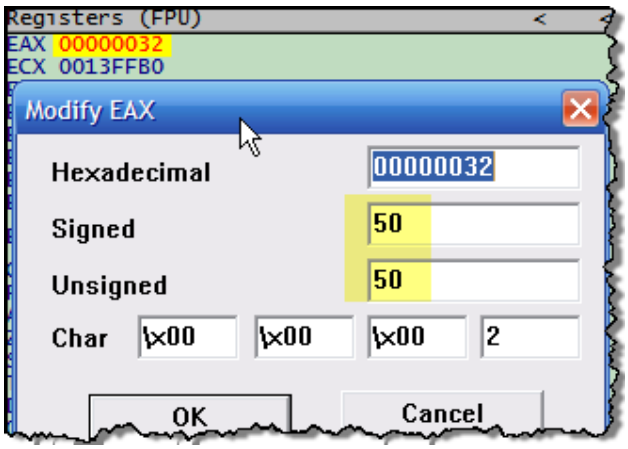
Thực hiện lệnh và quan sát kết quả :



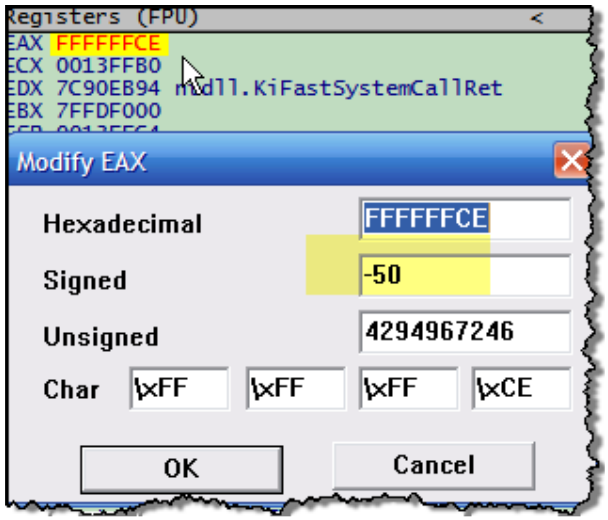
9. Câu lệnh NEG

Lệnh này thực hiện việc lấy bù hai của một toán hạng hay còn gọi là đảo dấu của một toán hạng. Điều này sẽ tương đương với công việc đảo bít của toán hạng và cộng với 1. Ví dụ :





Chúng ta thực hiện lệnh NEG và quan sát kết quả của EAX :



IV. Các lệnh Logic

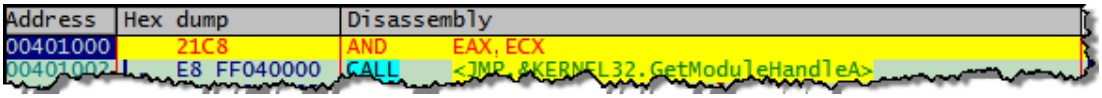
Trong phần này sẽ giới thiệu về các lệnh AND, OR, XOR, NOT. Các lệnh này có thể sử dụng để xóa, thiết lập và kiểm tra từng bit trong các thanh ghi hay các biến, khả năng thao tác với từng bit riêng biệt chính là một trong những ưu điểm của hợp ngữ.
Chúng ta có thể thay đổi từng bit trong máy tính bằng các lệnh logic. Các giá trị nhị phân 0 và 1 được xem như là các giá trị logic TRUE hoặc FALSE một cách tương ứng.

1. Lệnh AND

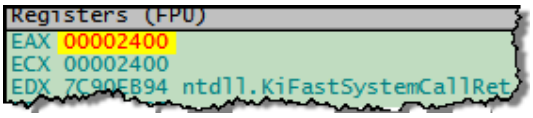
Kết quả của lệnh AND là 1 nếu như hai bit là 1, ngược lại là 0 :

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

Minh họa bằng Olly lệnh : AND EAX, ECX



Dạng binary của EAX: 0x3500	11010100000000
Dạng binary của ECX: 0x2400	10010000000000
AND EAX, ECX	10010000000000



Lệnh AND có thể được dùng để xóa các bit nhất định của toán hạng đích trong khi giữ nguyên những bit còn lại. Bit 0 của mặt nạ xóa bit tương ứng, còn bit 1 của mặt nạ giữ nguyên bit tương ứng của toán hạng đích

2. Lệnh OR :

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

Lệnh OR có thể được dùng để thiết lập các bit xác định của toán hạng đích trong khi vẫn giữ nguyên những bit còn lại. Bit 1 của mặt nạ sẽ thiết lập bit tương ứng trong khi bit 0 của nó sẽ giữ nguyên bit tương ứng trong toán hạng đích.

3. Lệnh XOR :

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

Lệnh XOR có thể được dùng để đảo các bit xác định của toán hạng đích trong khi vẫn giữ nguyên các bit còn lại. bit 1 của mặt nạ làm đảo bit tương ứng còn bit 0 của mặt nạ giữ nguyên bit tương ứng của toán hạng đích.

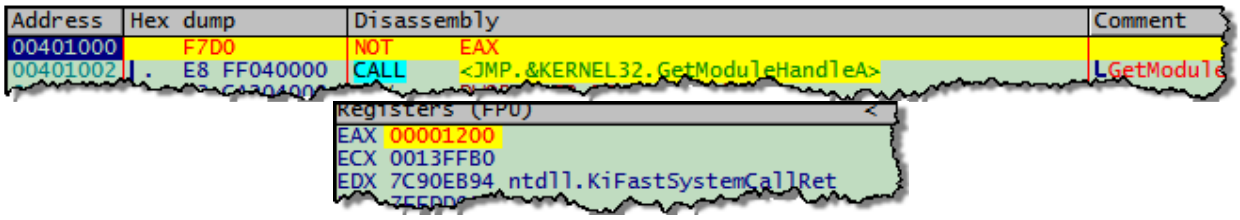
4. Lệnh NOT :

Thực hiện việc đảo bít :

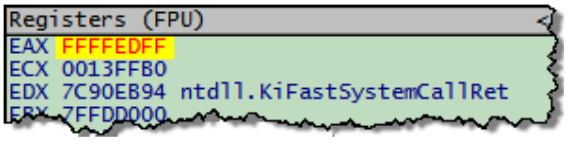
not 1 = 0
not 0 = 1

Ví dụ : not 0110 = 1001.
Giả sự ta có thanh ghi EAX mang giá trị là 0x1200:

Biểu diễn ở dạng Bin	00000000000000000001001000000000
Thực hiện đảo bit	11111111111111111110110111111111
Đưa về dạng Hexa	FFFFEDFF



Kết quả sau khi thực hiện lệnh :



Okie tôi nghĩ đến đây là đã đủ cho một bài viết và phần 5 về Ollydbg xin được kết thúc tại đây, trong phần này tôi có tham khảo thêm một số tài liệu liên quan tới lập trình ASM. Tôi

tin là với những ví dụ minh họa trực quan như ở trên các bạn sẽ nắm được vấn đề nhanh hơn.Tuy nhiên câu lệnh của ASM không phải chỉ có thế,các bạn có thể tham khảo thêm các tài liệu liên quan để có được một cái nhìn sâu hơn. Hẹn gặp lại các bạn trong phần 5 của loạt bài viết về Olly, By3 By3!! ☺

Best Regards
[Kienmanowar]

--++--==[Greatz Thanks To]==--++--
My family, Computer_Angel, Moonbaby , Zombie_Deathman, Littleboy, Benina, QHQCrker, the_Lighthouse, Merc, Hoadongnoi, Nini ... all REA's members, TQN, HacNho, RongChauA, Deux, tlandn, light.phoenix, dqtln, ARTEAM .... all my friend, and YOU.

--++--==[Thanks To]==--++--
iamidiot, WhyNotBar, trickyboy, dzungltvn, takada, hurt_heart, haule_nth, hytkl v..v.. các bạn đã đóng góp rất nhiều cho REA. Hi vọng các bạn sẽ tiếp tục phát huy ☺

I want to thank **Teddy Roggers** for his great site, Reversing.be folks(especially **haggar**), Arteam folks(**Shub-Nigurrath, MaDMAn_H3rCuL3s**) and all folks on crackmes.de, thank to all members of **unpack.cn** (especially **fly** and **linhanshi**). Great thanks to **lena151**(I like your tutorials). And finally, thanks to **RICARDO NARVAJA** and all members on **CRACKSLATINOS.**

>>>> If you have any suggestions, comments or corrections email me:
kienmanowar[at]reaonline.net