INTRODUCTION TO THE CRACKING WITH OLLYDBG

FROM CRACKLATINOS

(_kienmanowar_)

I. Lời nói đầu

Hà Nội trời lạnh nhưng cũng không thể át được không khí hừng hực lửa tại triển lãm Giảng Võ.Hàng nghìn con người hò hét, lắc giật xé tan bầu không khí lạnh lẽo. Sau một đêm "phê" cùng R0ck, toàn thân mệt nhoài, cổ đau đến hôm nay mới đỡ tôi lại tiếp tục dành thời gian để hầu tiếp các bạn phần ba trong loạt tut về Ollydbg. Phần ba này sẽ tập trung giới thiệu tới các bạn ý nghĩa của các thanh ghi, các cờ thường được sử dụng trong quá trình crack hay reverse chương trình. Tôi sẽ cố gắng đúc kết lại sao cho các bạn dễ dàng tiếp cận nhanh nhất có thể… 0k13! L3t's R0ck w1th m3 ☺

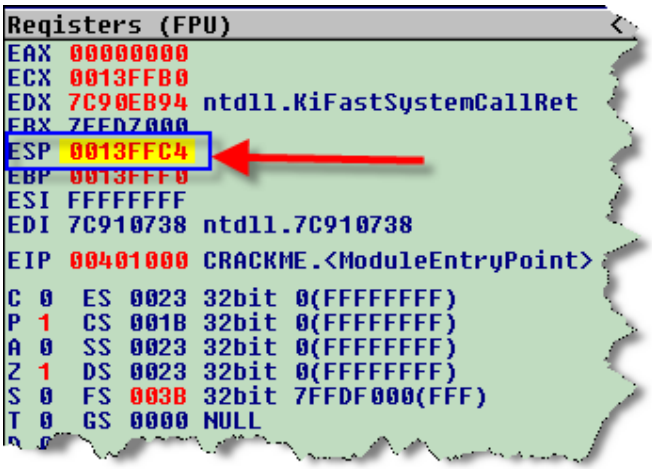
II. Giới thiệu chung

Thông tin được lưu giữ bên trong bộ vi xử lý trong các thanh ghi. Các thanh ghi được phân loại theo chức năng của chúng. Bộ vi xử lý dựa vào sự trợ giúp của các thanh ghi để thực thi một chương trình. Các thanh ghi được phân loại như sau : thanh ghi dữ liệu chứa dữ liệu cho một thao tác, thanh ghi địa chỉ chứa địa chỉ của lệnh hay của dữ liệu và thanh ghi trạng thái lưu trạng thái hiện thời của bộ vi xử lý. Đối với bộ xử lý 8086 có bốn thanh ghi dữ liệu công dụng chung, các thanh ghi địa chỉ được chia ra làm các thanh ghi đoạn, thanh ghi con trỏ, thanh ghi chỉ số; thanh ghi trạng thái còn được gọi là các cờ. Khi mới làm quen với các thanh ghi tôi khuyên bạn không nên học thuộc hết các chức năng của các thanh ghi liền một lúc, các bạn nên làm quen với các thanh ghi dần dần trong quá trình học cũng như trong lúc thực hành với Ollydbg.

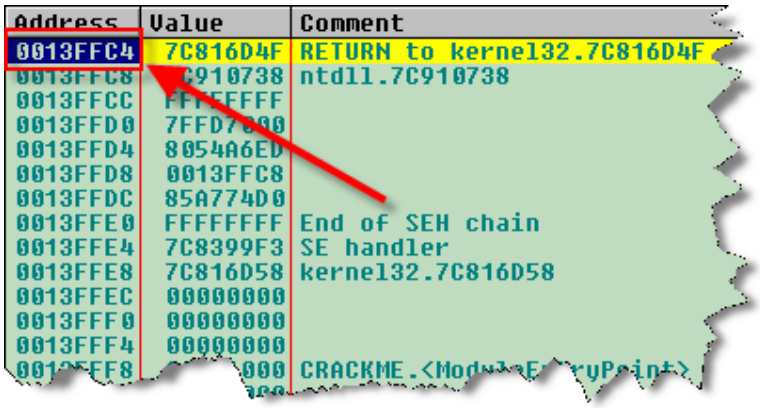
III. Chi tiết về các thanh ghi và công dụng

1. Thanh ghi ESP :

Thanh ghi đầu tiên mà tôi muốn giới thiệu tới các bạn đó chính là thanh ghi **ESP (con trỏ ngăn xếp – Stack pointer)**. Thanh ghi này luôn trỏ tới đỉnh hiện thời của ngăn xếp. Các bạn xem hình minh họa dưới đây :



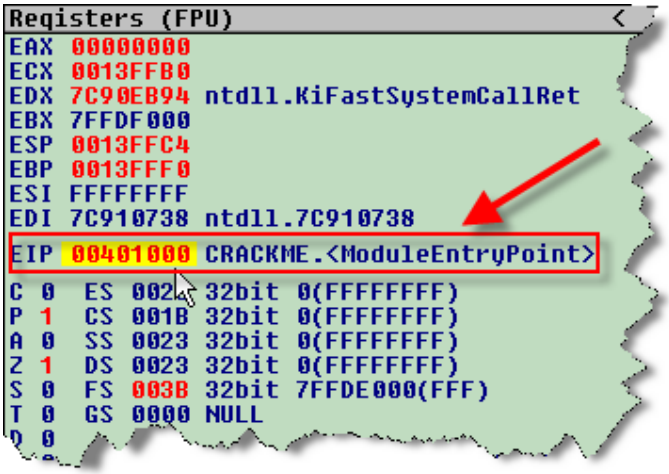
Như các bạn thấy trên hình, giá trị của thanh ghi ESP là **0x0013FFC4h,** quan sát tại cửa sổ Stack các bạn sẽ thấy giá trị này đang nằm tại đỉnh của Stack.



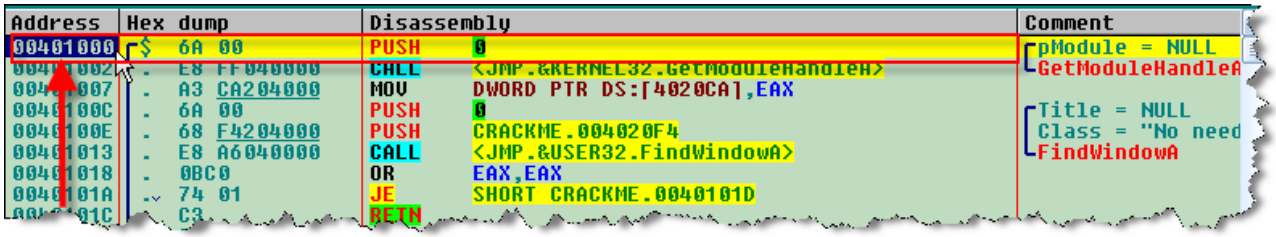
Thanh ghi ESP trỏ tới địa chỉ vùng nhớ nơi mà thao tác stack tiếp theo sẽ được thực hiện.

2. Thanh ghi EIP :

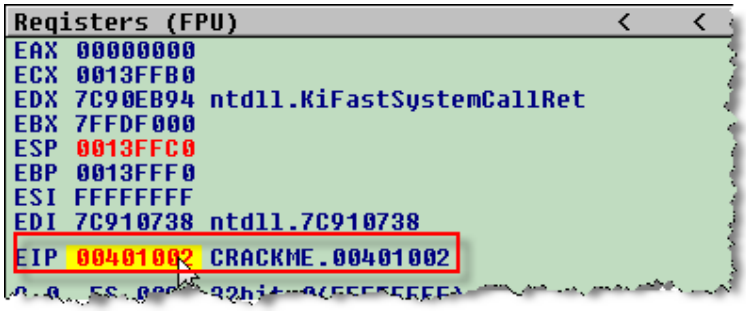
Để truy cập đến các lệnh, 8086 sử dụng thanh ghi **EIP (Instruction Pointer)**. Đây là một thanh ghi rất quan trọng, nó được cập nhật mỗi khi có một lệnh được thực hiện để sao cho nó luôn trỏ đến lệnh tiếp theo. Khác với các thanh ghi khác EIP không thể bị tác động trực tiếp bởi các lệnh, do đó trong một lệnh chúng ta sẽ thấy thường không có mặt thanh ghi EIP như một toán hạng. Ví dụ quan sát cửa sổ Registers trong Olly chúng ta thấy như sau :

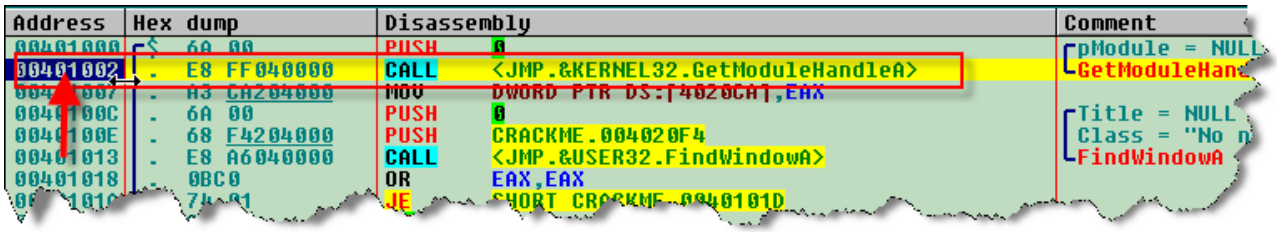


Chúng ta thấy rằng thanh ghi EIP mang giá trị là **0x00401000h**, điều này có nghĩa là địa chỉ 0x00401000h chính là địa chỉ của câu lệnh tiếp theo sẽ được thực hiện. Chúng ta quan sát trên cửa sổ CPU sẽ thấy được câu lệnh tại địa chỉ trên là câu lệnh gì :



Tại cửa sổ CPU, chúng ta nhấn **F8** để thực hiện câu lệnh đầu tiên tại địa chỉ **0x00401000h** và quan sát trên cửa sổ Register xem thanh ghi EIP sẽ thay đổi giá trị như thế nào ? Chúng ta sẽ thấy được như sau :

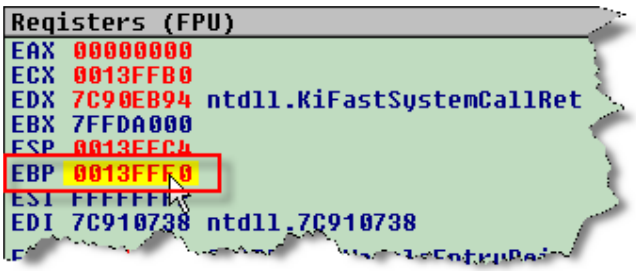


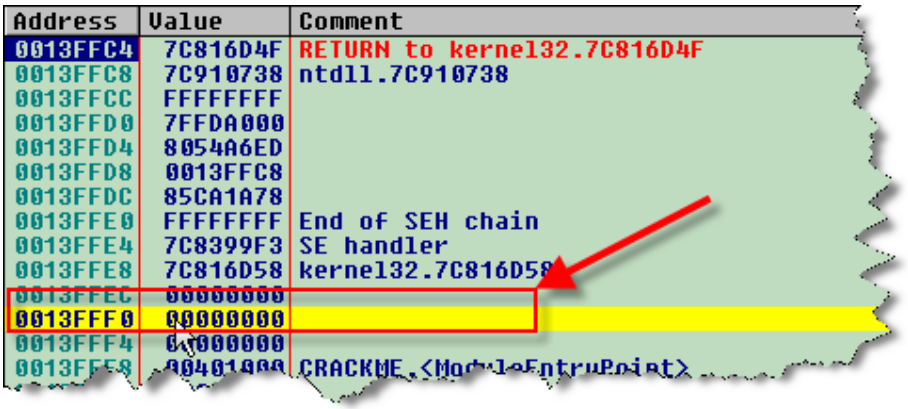


Oh, giá trị thanh ghi đã thay đổi thành **0x00401002h**, đó chính là địa chỉ của câu lệnh tiếp theo sẽ được thực hiện khi bạn quan sát trong màn hình CPU.

3. Thanh ghi EBP :

Đây cũng là một thanh ghi không kém phần quan trọng, thanh ghi **EBP (Con trỏ cơ sở - Base Pointer)** chủ yếu được sử dụng để truy nhập dữ liệu trong ngăn xếp. Tuy nhiên khác với thanh ghi ESP, thanh ghi EBP còn được sử dụng để truy nhập dữ liệu trong các đoạn khác. Thanh ghi EBP thường được kết hợp với ESP khi chúng ta bắt gặp một lời gọi hàm, thì trước khi hàm này được thực hiện địa chỉ trở về của chương trình (tức là địa chỉ của câu lệnh tiếp theo dưới lời gọi hàm) sẽ được cất vào Stack, và bên trong thân hàm giá trị hiện thời của thanh ghi EBP sẽ được đẩy vào Stack, bởi vì giá trị của thanh ghi EBP phải được thay đổi để có thể tham chiếu tới các giá trị trên Stack.





4. Các thanh ghi dữ liệu EAX, EBX, ECX, EDX:
Đây là 4 thanh ghi đa năng 32 bit, điều đặc biệt là khi cần chứa dữ liệu 16 bit ta có các thanh ghi sau AX, BX, CX, DX và đặc biệt hơn khi ta cần chữa dữ liệu 8 bit thì các thanh ghi AX, BX, CX, DX này có thể phân tách ra thành 2 thanh ghi 8 bit cao và thấp để làm việc độc lập, đó là các thanh ghi AH và AL, BH và BL, CH và CL, DH và DL. Bốn thanh ghi này ngoài ý nghĩa là những thanh ghi công dụng chung thì nó còn mang những ý nghĩa và chức năng đặc biệt sau :

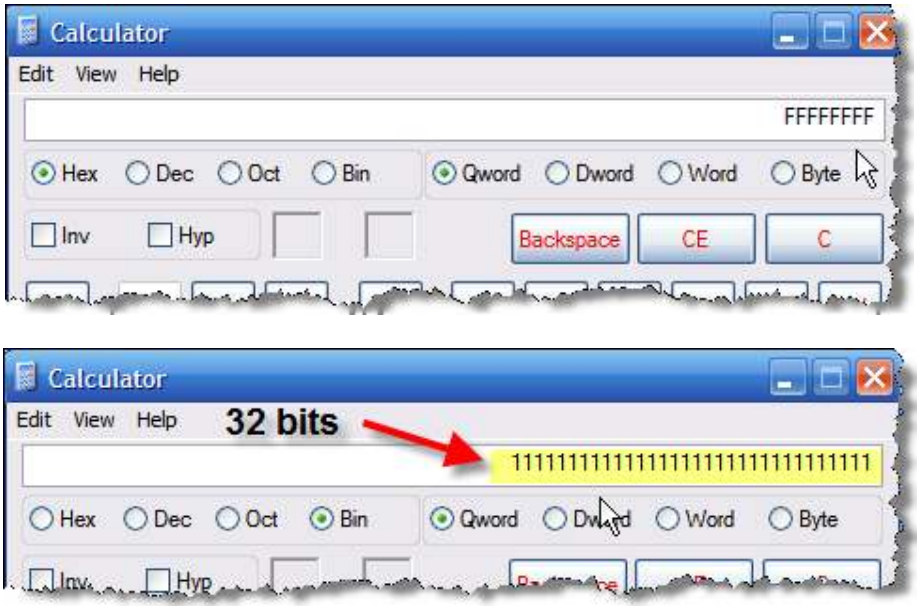
- Thanh ghi EAX (thanh ghi chứa) : thường được sử dụng trong các lệnh số học, logic và chuyển dữ liệu. Trong các thao tác nhân và chia thường sử dụng đến thanh ghi này.
- Thanh ghi EBX (thanh ghi cơ sở) : thanh ghi này cũng đóng vai trò là thanh ghi địa chỉ.
- Thanh ghi ECX (thanh ghi đếm) : thanh ghi này thường được sử dụng như một bộ đếm số lần lặp. Ngoài ra nó cũng được sử dụng như là biến đếm trong các lệnh dịch hay quay các bit.
- Thanh ghi EDX (thanh ghi dữ liệu) : thanh ghi này cùng với thanh ghi EAX tham gia vào thao tác của phép nhân hoặc phép chia. Bên cạnh đó nó cũng thường được sử dụng trong các thao tác vào ra.

5. Các thanh ghi chỉ số ESI, EDI :
Hai thanh ghi **ESI (chỉ số nguồn)** và **EDI (chỉ số đích)** thường được sử dụng trong các thao tác làm việc với chuỗi hoặc mảng.

Trong Ollydbg như các bạn đã làm quen trong các bài trước có một cửa sổ cho chúng ta quan sát trạng thái hiện thời của tất cả các thanh ghi, đó chính là cửa sổ **Registers**:

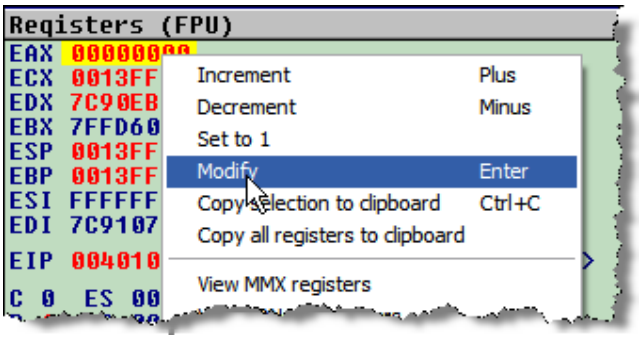
```
Registers (FPU)
EAX 00000000
ECX 0013FFB0
EDX 7C90EB94 ntdll.KiFastSystemCallRet
EBX 7FFD5000
ESP 0013FFC4
EBP 0013FFF0
ESI FFFFFFFF
EDI 7C910738 ntdll.7C910738

EIP 00401000 CRACKME.<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 SUCCESS (00000000)

EFL 00000246 (NO,NB,E,BE,NS,PE,GE,LE)

ST0 empty -UNORM D0A8 01050104 006500xE
ST1 empty +UNORM 0076 0020004B 00590044
ST2 empty +UNORM 002E 00320031 005B0030
ST3 empty +UNORM 0079 006C006C 004F005C
ST4 empty 0.10773512978743219 50e-4933
ST5 empty 0.0
ST6 empty 1.000000000000000000
ST7 empty 1.000000000000000000
               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 1372  Prec NEAR,64  Mask    1 1 0 0 1 0
```

Những thanh ghi này với chữ cái E ở đầu cho chúng ta biết được chúng là những thanh ghi 32 bits. Trong hình minh họa này các bạn thấy Ollydbg biểu diễn nội dung các thanh ghi ở dạng Hexa. Lấy thanh ghi EAX làm ví dụ, ta thấy giá trị của nó là 0x00000000h đây là giá trị nhỏ nhất của một thanh ghi, giá trị lớn nhất mà thanh ghi này có thể lưu trữ là 0xFFFFFFFFh, nếu như chúng ta chuyển nó sang dạng nhị phân thì chúng ta sẽ có được như sau :

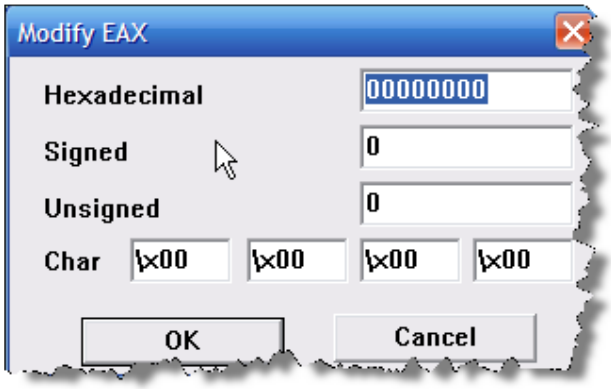


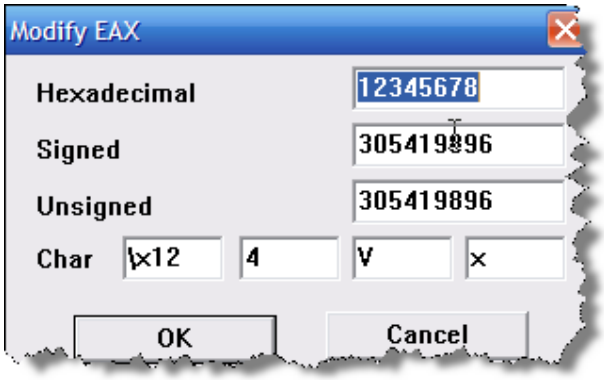
Chúng ta thấy rằng khi chuyển sang dạng nhị phân sẽ biểu diễn đúng 32 bits, 32 bits này sẽ có thể mang một trong hai giá trị 0 hoặc 1. Tuy nhiên trong lập trình ASM không phải lúc nào chúng ta cũng sử dụng hết 32 bits, để tránh lãng phí chúng ta có thể thao tác, tính

toán chỉ trên một phần của các thanh ghi này, trong trường hợp này của tôi tôi có thể chia nhỏ thanh ghi EAX ra. Ta sẽ làm một ví dụ cụ thể : Giả sử thanh ghi EAX tôi muốn thay đổi giá trị của nó thành là 0x12345678h. Đầu tiên ta load crackme vào trong Ollydbg, sau khi analyse xong chúng ta sẽ dừng lại tại EP (Entry Point) của chương trình. Bây giờ tôi sẽ thay đổi giá trị của thanh ghi EAX, chuột phải tại thanh ghi này và chọn **Modify** :

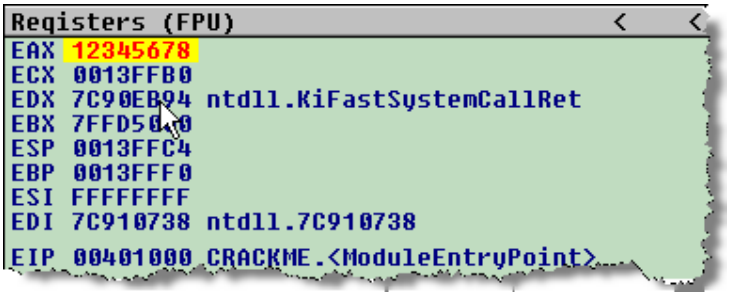


Chỉnh sửa lại giá trị của thanh ghi EAX như hình dưới đây :





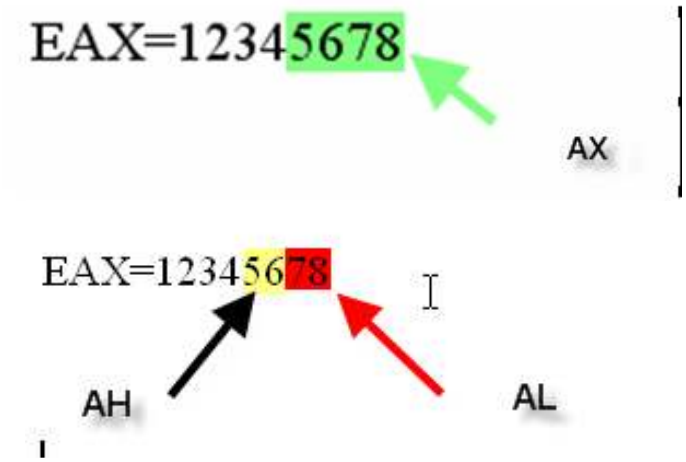
Kết quả sau khi chỉnh sửa :



Khi bạn thay đổi giá trị của một thanh ghi thì kết quả chỉnh sửa sẽ được đánh dấu màu đỏ. Như tôi đã nói ở trên nhiều khi chúng ta không sử dụng hết 32 bits trong quá trình tính toán mà ta chỉ cần một phần của nó thôi. Thông thường người ta thường sử dụng thanh ghi AX (16 bits) - một phần của thanh ghi EAX. Do AX là thanh ghi 16 bits nên nó chiếm ½ trên tổng số bits, vậy ta có giá trị của thanh ghi AX là 0x5678h. Để biết được kết quả này có chính xác hay không, chúng ta sẽ sử dụng Command Bar :

Đúng như những gì chúng ta đã lập luận ở trên, giá trị của AX là 0x5678h. Tuy nhiên bản thân thanh chi AX cũng lại được phân tách thành các thanh ghi AH (8 bits cao) và AL (8 bits thấp). Ta thử quan sát giá trị của thanh ghi AL :

Tổng kết lại ta có một hình ảnh minh họa trực quan như sau :

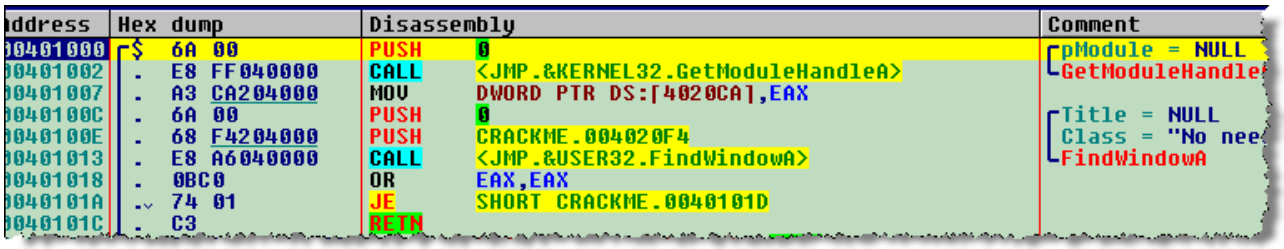


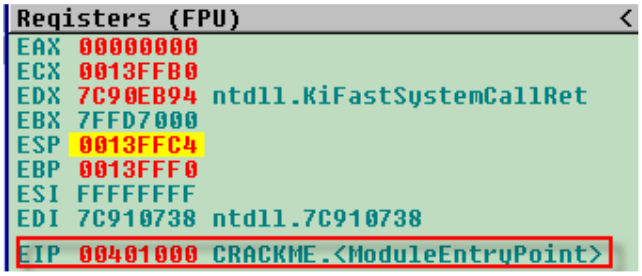
Tất cả những thanh ghi đa năng khác cũng sẽ được phân tách tương tự như thanh ghi EAX.

6. Thay đổi giá trị của những thanh ghi :

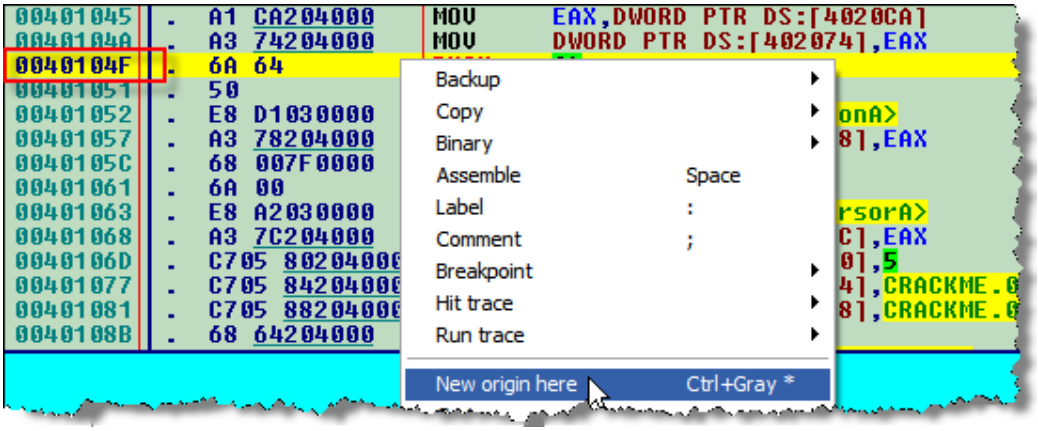
Ở phần trên, các bạn đã quan sát thấy quá trình thay đổi giá trị của một thanh ghi , ví dụ như thanh ghi EAX diễn ra rất đơn giản, gọn nhẹ. Cách thức thay đổi giá trị bằng cách nhấp chuột phải tại thanh ghi đó và chọn **Modify**.

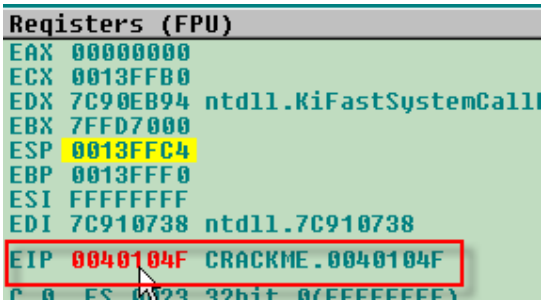
Chúng ta sẽ làm thêm một ví dụ nữa, như các bạn đã biết thanh ghi EIP được sử dụng để trỏ tới lệnh tiếp theo sắp được thực hiện, giờ đây tôi muốn thay đổi nội dung của thanh ghi này thì phải làm thế nào ? Giả sử khi ta load crackme vào trong Olly thì ta có được như sau :





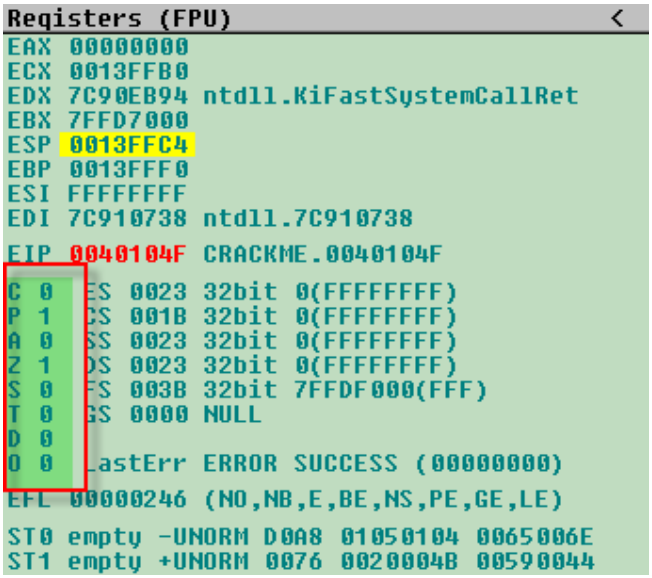
Trong hình minh họa này thì thanh ghi EIP mang giá trị là 0x00401000h, điều này có nghĩa là câu lệnh tại địa chỉ đó sẽ được thực hiện. Nhưng nếu ta muốn thay đổi giá trị EIP để cho chương trình sẽ thực hiện một câu lệnh ở địa chỉ khác chứ không phải là câu lệnh tại địa chỉ 0x00401000h thì làm thế nào? Rất đơn giản, chúng ta chỉ việc chọn dòng lệnh đó nhấn chuột phải và chọn **New origin here**, ngay lập tức giá trị của EIP sẽ thay đổi theo :





IV. Các cờ (Flags) hay được sử dụng

Trong phần tiếp theo của bài viết này, chúng ta sẽ làm quen với các cờ. Trong cửa sổ **Registers** thì các cờ sẽ nằm bên dưới các thanh ghi ☺

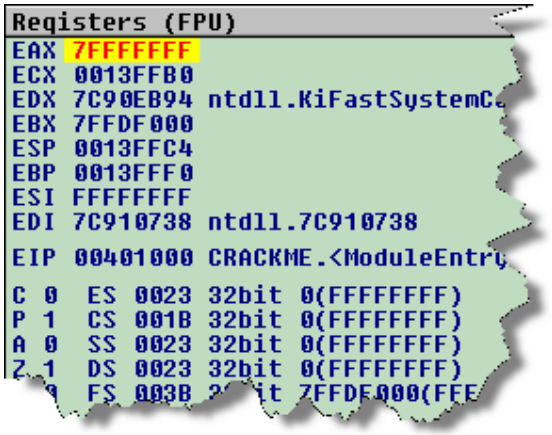


Trong hình trên bạn quan sát sẽ thấy có các cờ sau : C, P, A, Z, S, T, D và O.Thêm vào đó các bạn cũng thấy là các cờ này chỉ có 1 bit mà thôi do đó giá trị của nó chỉ có thể là 0 hoặc 1. Các cờ này được đặt trong thanh ghi cờ và chúng được phân chia ra thanh hai loại là cờ trạng thái và cờ điều khiển. Cờ trạng thái phản ánh kết quả của các phép tính. Cờ điều khiển được sử dụng để cho phép hoặc không cho phép một thao tác nào đó của bộ vi xử lý. Nói tóm lại là mỗi một thao tác của CPU đều dựa vào các cờ để ra quyết định. Chúng ta sẽ đi vào xem xét từng cờ một.

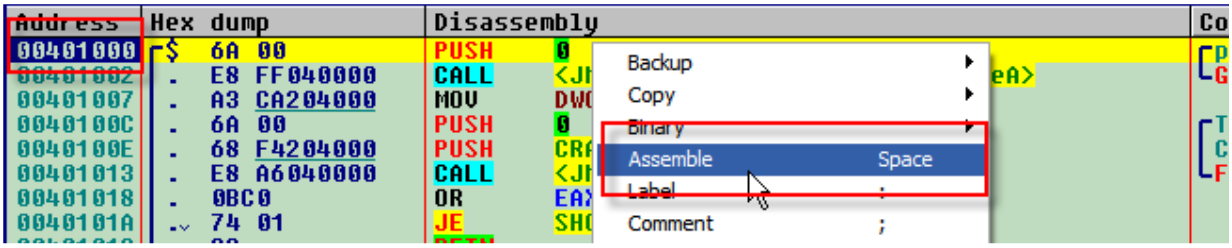
a. Cờ trạng thái :

Cờ tràn (Overflow Flag):

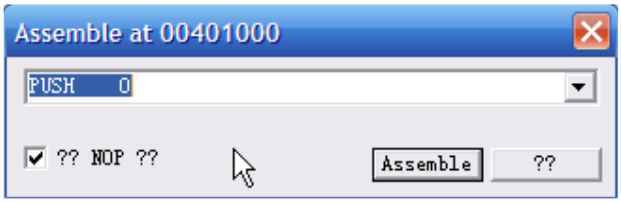
Cờ OF = 1 khi xảy ra hiện tượng tràn, thường là khi kết quả là một số bù hai vượt ra ngoài giới hạn biểu diễn dành cho nó. Để minh họa về cờ OF các bạn mở Ollydbg và load crackme vào. Sau đó tiến hành sửa giá trị của thanh ghi EAX = 0x7FFFFFFFh (giá trị của số dương lớn nhất), ta có được như sau :



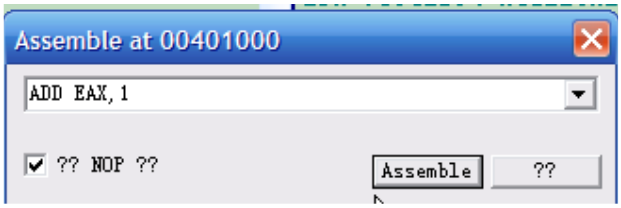
Bây giờ chúng ta sẽ thực hiện cộng thêm 1 vào thanh ghi EAX, chúng ta sẽ nghĩ ngay đến rằng giá trị của thanh ghi EAX sau khi cộng một sẽ là 0x80000000h (giá trị này tương đương với số âm). Để thực hiện được phép cộng này ta sẽ làm như sau : Chuột phải tại lệnh mà bạn muốn thay đổi và chọn **Assemble**.



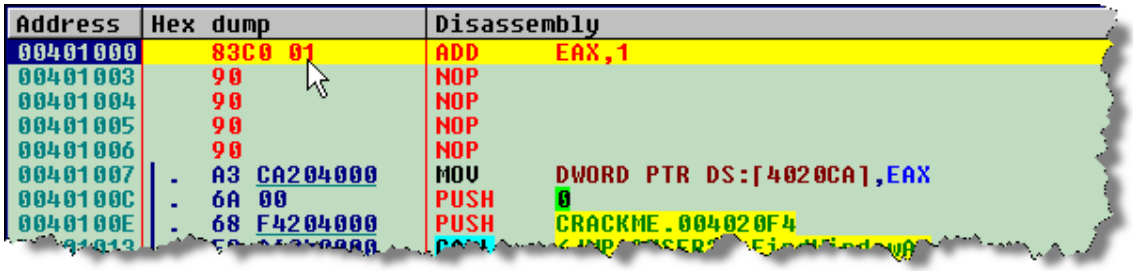
Ta có được như sau :



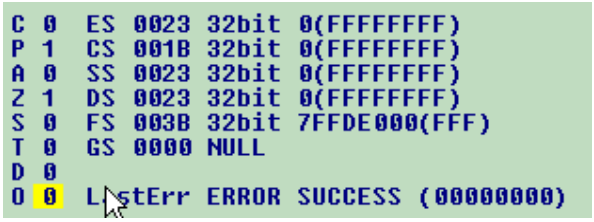
Thay câu lệnh `Push 0` thành câu lệnh `Add EAX, 1`.



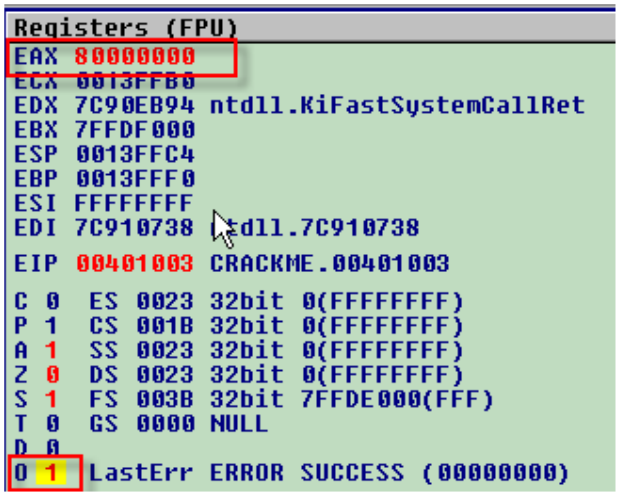
Sau khi thay đổi xong chúng ta quan sát kết quả thay đổi trên cửa sổ CPU.



Trước khi tôi và các bạn thực thi câu lệnh `ADD EAX,1` – chúng ta sẽ quan sát giá trị của của cờ OF. Ta có được kết quả như sau :



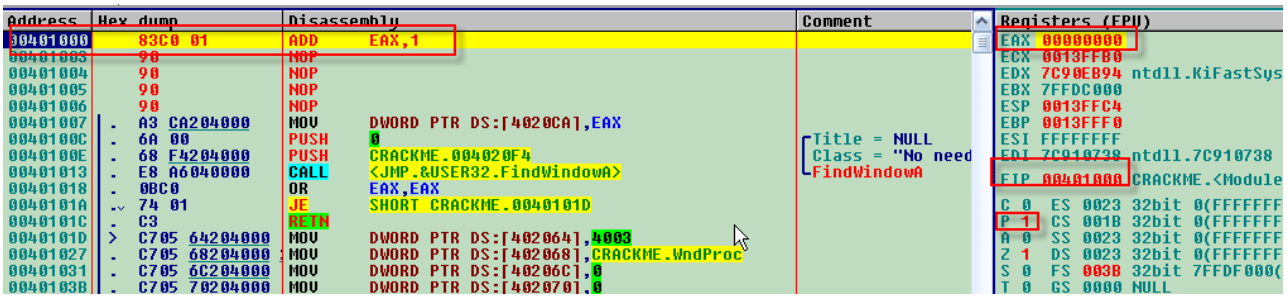
Okie, lúc này giá trị của cờ tràn O vẫn đang là 0. Chúng ta sẽ thử thực thi câu lệnh `Add EAX,1` xem thế nào. Nhấn F8 để thực thi lệnh và quan sát cửa sổ **Registers** :



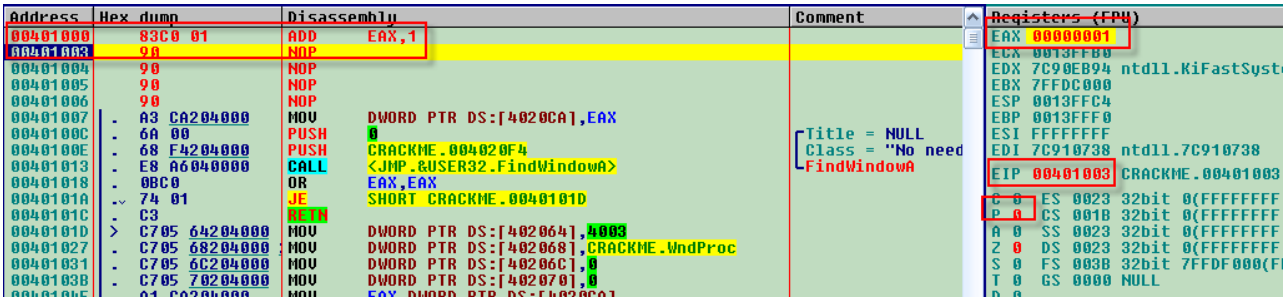
Note: nhận xét về tràn với số có dấu là nếu cộng 2 số cùng dấu mà kết quả có dấu ngược lại thì có tràn xảy ra. Khi ta cộng hai số khác dấu thì không bao giờ có tràn.

Cờ chẵn lẻ (Parity Flag):

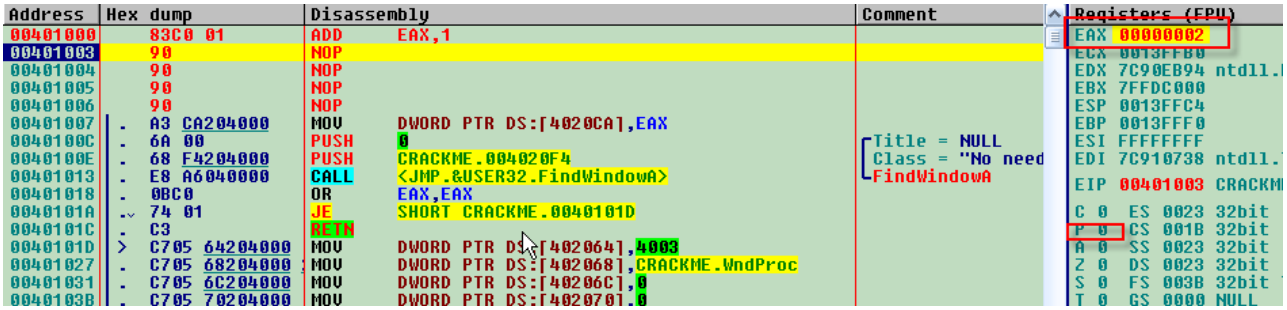
Cờ này phản ánh tính chẵn lẻ của tổng số bit 1 có trong kết quả. Cờ PF = 1 khi tổng số bit 1 trong kết quả là chẵn – parity chẵn.Nó bằng không trong trường hợp ngược lại.Lấy ví dụ cụ thể như sau. Chúng ta có thanh ghi EAX = 0x0h,cờ PF lúc đầu bằng 1, chúng ta sẽ thực hiện lệnh cộng EAX với 1 và xem kết quả giá trị trên cờ PF.



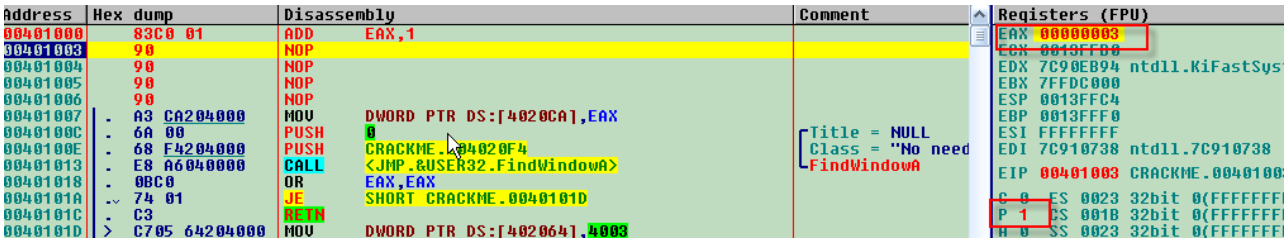
Sau khi thực hiện lệnh ta có được như sau :



Ok chúng ta thấy rằng PF đã bằng 0 bởi vì giá trị thanh ghi EAX = 1, mà tổng số bit 1 trong thanh ghi EAX lúc này là 1, vậy theo định nghĩa thì giá trị PF = 0 là hoàn toàn chính xác. Quay trở lại vấn đề, lúc này ta chuột phải tại địa chỉ 0x00401000 và chọn **New Origin**. Nhấn F8 để thực hiện lại câu lệnh. Chúng ta có được kết quả như sau :

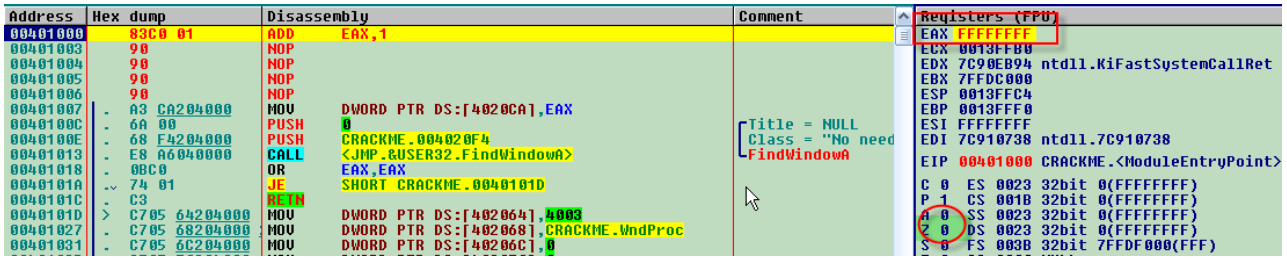


Ta vẫn thấy cờ PF giữ nguyên kết quả. Đó là bởi vì thanh ghi EAX có giá trị là 0x02 (tổng số bit 1 là lẻ). Lại làm như trên và thực hiện lại câu lệnh `add EAX,1`. Quan sát hình minh họa dưới đây :

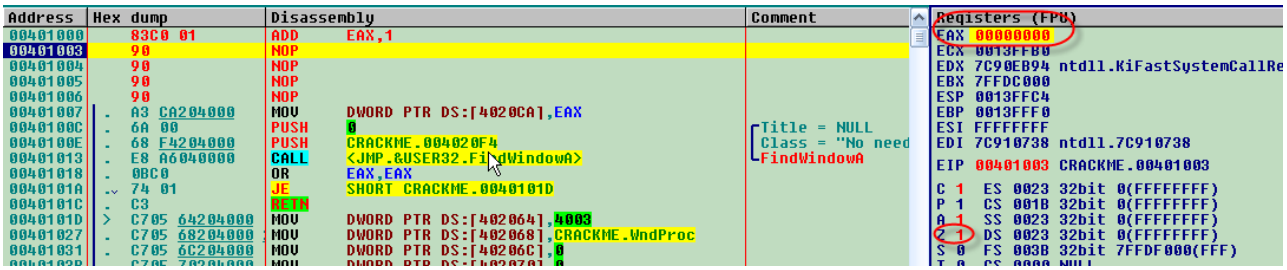


Cờ Zero (Parity Flag):

Cờ này được thiết lập 1 khi kết quả bằng không và ngược lại. Cờ này khá quan trọng đối với việc crack. Để minh họa cho cờ này chúng ta vẫn sẽ sử dụng lệnh `Add EAX, 1` nhưng lần này chúng ta sẽ thay lại giá trị của thanh ghi EAX thành 0xFFFFFFFF (-1).



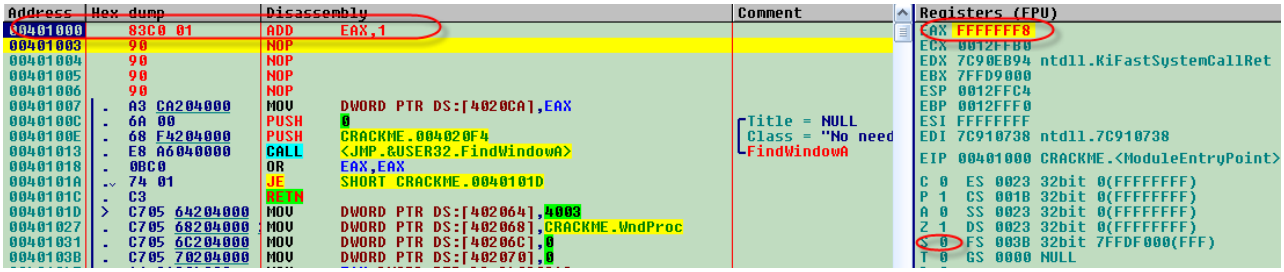
Nhấn F8 để thực hiện câu lệnh add, ta sẽ có được thông tin như sau :



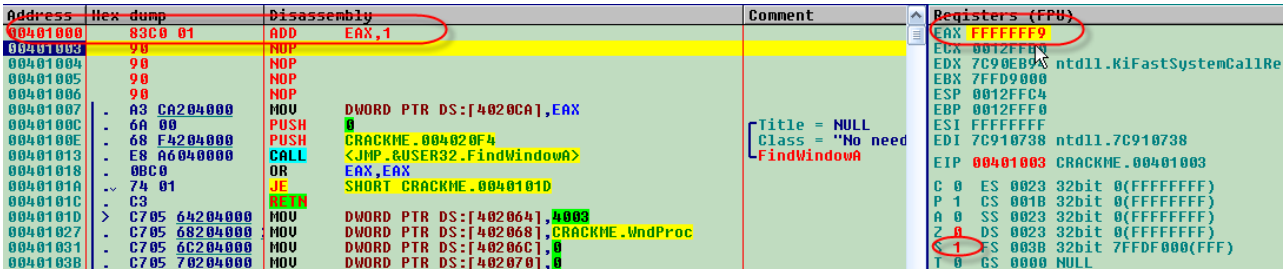
Khà khà, do kết quả của phép cộng thì thanh ghi EAX có giá trị là 0x0h cho nên cờ Z sẽ được bật lên thành 1.

Cờ dấu (Sign Flag):

Cờ SF được thiết lập là 1 khi bít msb của kết quả bằng 1 có nghĩa là kết quả âm nếu bạn muốn làm việc với số có dấu. Tôi sẽ minh họa về cờ SF như sau, trong Olly chúng ta thay giá trị của thanh ghi EAX thành 0xFFFFFFF8 (=-8). Thực hiện câu lệnh `add eax, 1` để tính lại giá trị của thanh ghi EAX.



Thực thi lệnh, kết quả của thanh ghi EAX sẽ là 0xFFFFFFFF9 (=-7) :



Cờ nhớ(Carry Flag):
Cờ CF được thiết lập 1 khi có nhớ hoặc mượn từ bit msb.

b.Cờ điều khiển
Có 3 cờ được sử dụng trong việc điều khiển đó là cờ T. I và cờ D. Cờ T(cờ bẫy) bằng 1 thì CPU làm việc ở chế độ chạy từng lệnh –chế độ này thường được dùng khi cần tìm lỗi trong chương trình.)
Cờ I (cờ ngắt) , cờ này bằng 1 thì CPU cho phép các yêu cầu ngắt được tác động.
Cờ D (cờ hướng), cờ này bằng 1 khi CPU làm việc với chuỗi kí tự theo tứ tự từ phải sang trái.

Okie vậy là phần 3 đến đây là kết thúc, trong phần này tôi có tham khảo thêm một số tài liệu liên quan tới lập trình ASM. Các bạn có thể tham khảo thêm các tài liệu liên quan để có được một cái nhìn sâu hơn.

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