INTRODUCTION TO THE CRACKING WITH OLLYDBG

FROM CRACKLATINOS

(_kienmanowar_)

I. Lời nói đầu

Hà Nội trời lạnh quá, ngoài trời mưa phùn, quàng khăn đi găng tay cộng thêm cái mũ mà vẫn thấy lạnh. Đêm nay rảnh rỗi tôi lại tiếp tục phần bốn trong loạt tut về Ollydbg như đã hứa với các bạn. Trong bài viết trước tôi đã tập trung giới thiệu ý nghĩa của các thanh ghi, các cờ thường được sử dụng trong quá trình crack hay reverse chương trình, cũng như quan sát thấy các cờ thay đổi trạng thái như thế nào khi ta thực thi một câu lệnh có tác động đến cờ. Trong phần bốn này sẽ đề cập tới những câu lệnh Asm cơ bản, cách thức chúng thi hành. Như những gì chúng ta đã làm trong các phần trước, sẽ không có gì dễ hiểu hơn là khi tìm hiểu về một công cụ chúng ta tiến hành thực hành luôn trên công cụ đó để kiểm nghiệm những kiến thức mà chúng ta tiếp thu được trong quá trình đọc tài liệu. 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 ☺

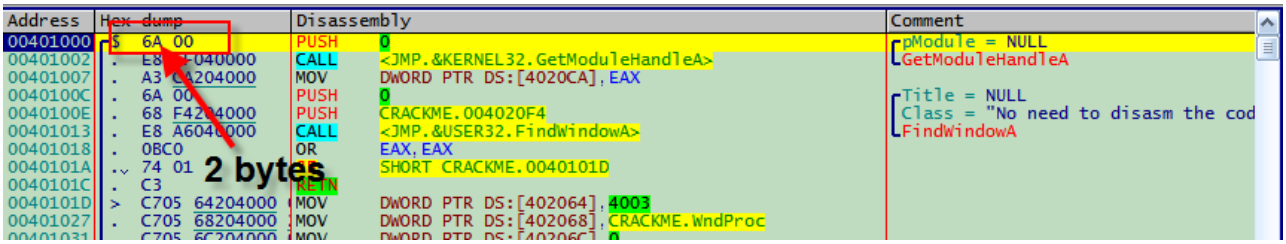
II. Giới thiệu chung

Tập lệnh của bộ vi xử lý có đến hơn trăm lệnh, trong đó có các lệnh được thiết kế dành riêng cho các bộ vi xử lý cao cấp. Trong bài viết này tôi chỉ đề cập đến những câu lệnh hay dùng nhất, chung nhất mà thôi. 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.

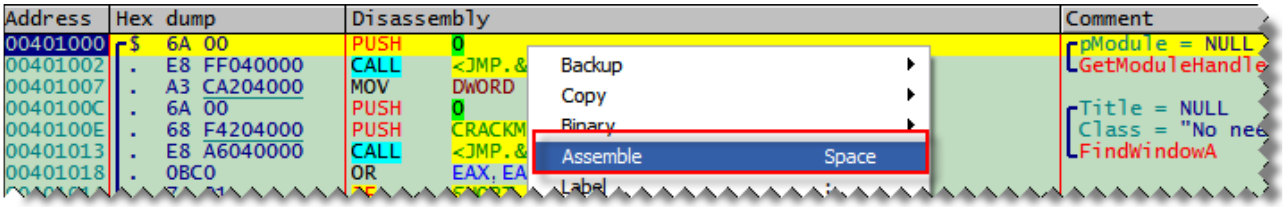
III. Chi tiết về các câu lệnh ASM hay dùng

1. NOP (No Operation) :

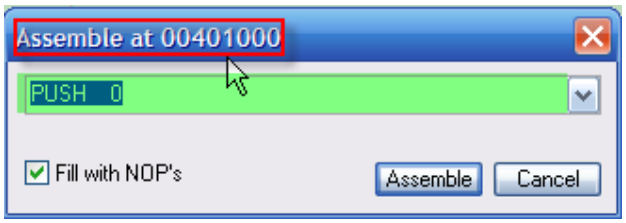
Cái tên của nó đã cho bạn thấy được ý nghĩa. Lệnh này không thực hiện một công việc gì cả ngoại trừ việc tăng nội dung của thanh ghi `EIP`, nó không gây ra bất kì thay đổi nào trong thanh ghi, stack hoặc memory. Chính vì ý nghĩa này của nó mà câu lệnh này thường được dùng vào mục đích hủy bỏ bất kì câu lệnh nào (không cho lệnh đó thực hiện), bằng cách ta thay thế câu lệnh sắp thực hiện bằng lệnh `NOP` chương trình sẽ vẫn thực thi nhưng thay vì thực thi câu lệnh gốc thì giờ đây do được thay thế bằng `NOP` nên nó sẽ không làm gì cả. Đó là lý do tại sao các bạn hay thấy người ta sử dụng `NOP` (ví dụ như : tôi muốn loại bỏ một thông báo nào đó, để làm được điều này tôi thay thế lệnh Call đến thông báo bằng lệnh `NOP`, vậy là thông báo đó sẽ biến mất ☺). Okie, mở Cruehead crackme trong Olly, ta có như sau :



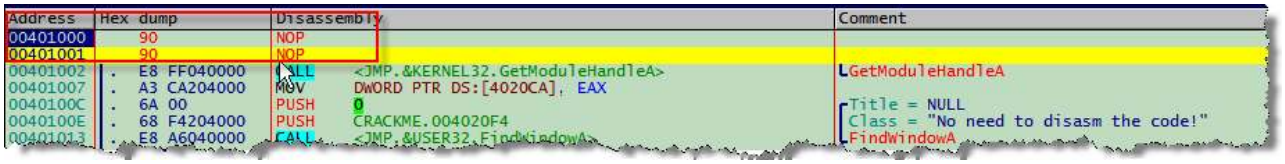
Trong hình trên, mới đầu khi load vào Olly ta sẽ có được đoạn code gốc của chương trình đã được Olly phân tích sang ASM. Bây giờ chúng ta sẽ thay thế câu lệnh `PUSH 0` như trên hình minh họa bằng lệnh NOP. Các bạn để ý vùng tôi khoanh đỏ, lệnh Push này có 2 bytes mà trong khi lệnh NOP chỉ có 1 byte mà thôi, vậy cho nên khi ta thay thế lệnh PUSH bằng lệnh NOP chúng ta sẽ thấy trên màn hình Olly xuất hiện 2 lệnh NOP. Để có thể thay đổi một câu lệnh trong Olly, chúng ta làm như sau : chuột phải trên lệnh cần thay đổi và chọn **Assemble**(hoặc nhấn Space Bar).



Ngay lập tức một cửa sổ bật ra thông báo cho chúng ta biết chúng ta đang chuẩn bị thay đổi lệnh ở địa chỉ nào, và một textbox để cho phép ta nhập lệnh mà chúng ta muốn thay đổi :

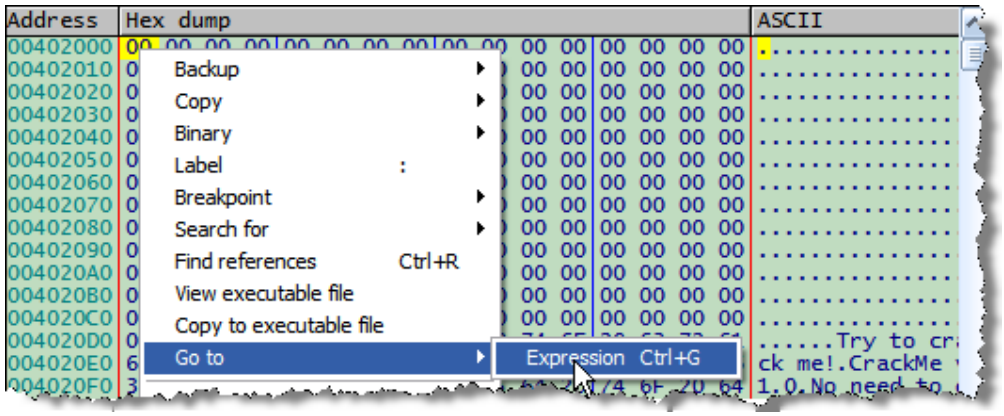


Như các bạn thấy trên hình, bây giờ ta muốn thay lệnh `PUSH 0` bằng lệnh NOP. Rất đơn giản ta xóa `PUSH 0` đi và gõ vào NOP và nhấn **Assemble**.



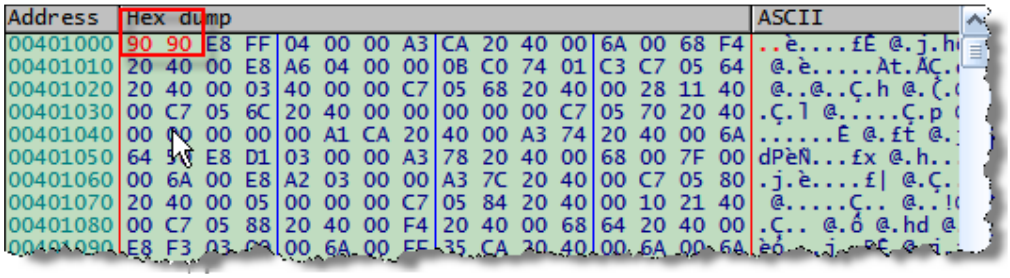
Ở đây chúng ta nhận thấy rằng Olly ngoài việc sẽ thay thế bằng lệnh NOP, chương trình này còn nhận biết được rằng lệnh PUSH 0 trước khi thay thế là một câu lệnh 2 bytes cho nên nó sẽ tự động điền thêm một lệnh NOP ở địa chỉ `0x00401001` cho vừa đủ. So sánh lại với hình trước thì các bạn thấy rằng lệnh PUSH 0 đã hoàn toàn được thay thế bằng 2 lệnh NOP, 2 lệnh này sẽ không thực hiện bất kì công việc gì. Để kiểm chứng bạn nhấn F8 để trace lần lượt qua 2 lệnh NOP này và quan sát các cửa sổ xem có sự thay đổi gì không.Oh, các thanh ghi không thay đổi, stack cũng không biến chuyển, các cờ vẫn bình thường chỉ có mỗi một thay đổi xảy ra đó là với thanh ghi `EIP` (điều này là hiển nhiên rồi vì thanh ghi `EIP` sẽ luôn trỏ tới câu lệnh tiếp theo được thực hiện).

Tiếp theo chúng ta sẽ quan sát sự thay đổi của 2 bytes này trong cửa sổ Dump, để tìm chúng ta sẽ tìm kiếm theo địa chỉ chứa hai lệnh này. Như các bạn thấy ở trên đó là `0x00401000` và `0x00401001`. Chúng ta tới cửa sổ DUMP, chuột phải và chọn như sau :



Một cửa sổ hiện ra, tại đây ta gõ vào : `00401000` và nhấn OK.

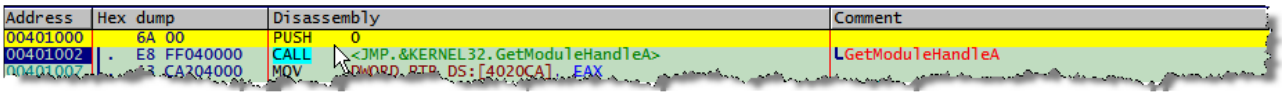
Kết quả ta có được như sau :



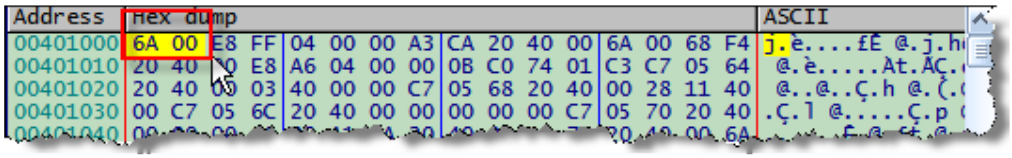
Olly sẽ biểu diễn những gì chúng ta thay đổi bằng màu đỏ, trong hình trên chúng ta thấy có 2 mã lệnh `90 90` tương đương với 2 lệnh NOP NOP. Tiếp theo đó là `E8 FF ..` tương đương với câu lệnh CALL như các bạn quan sát thấy trong hình bên trên.

Có một câu hỏi nhỏ đặt ra : Tôi có thể loại bỏ những gì tôi vừa thay đổi và quay trở lại đoạn code gốc được không ? Hoàn toàn có thể được, Olly hỗ trợ cho chúng ta chức năng Undo.Để làm điều này thì trong cửa sổ Dump hoặc cửa số CPU chúng ta chỉ việc đánh dấu những bytes mà chúng ta đã thay đổi, nhấn chuột phải và chọn như trong hình dưới đây :

Sau khi làm đúng như trên, bạn sẽ có được lệnh `PUSH 0` ban đầu :



Quan sát tại cửa sổ DUMP, chúng ta có được như sau :



Y34h! Với lệnh NOP thế là quá đủ! Chúng ta chuyển sang các lệnh liên quan đến STACK.

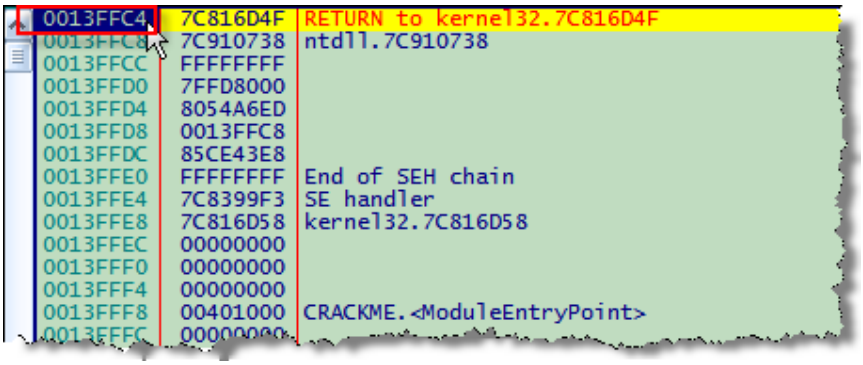
2. Các lệnh liên quan đến STACK :

Trong các phần trước chúng ta đã tìm hiểu về chức năng cơ bản của STACK rồi, phần này tôi sẽ đề cập đến các lệnh làm việc với STACK mà cụ thể ở đây là lệnh PUSH (đẩy dữ liệu vào Stack) và POP (lấy dữ liệu ra khỏi Stack).

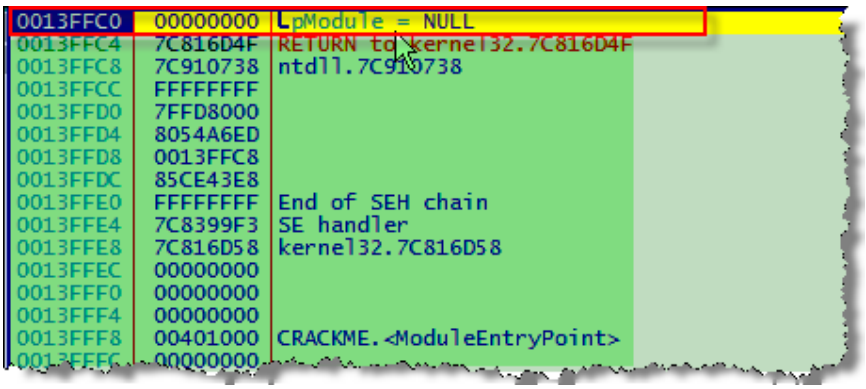
2.a. Lệnh PUSH

Về cơ bản lệnh PUSH được dùng để thêm/cất 1 từ (Word (letters hoặc value)) vào trong ngăn xếp. Như các bạn thấy trong Olly câu lệnh đầu tiên của Cruehead crackme là PUSH, cụ thể trong trường hợp này là PUSH 0.Khi chúng ta thực hiện câu lệnh này thì điều gì sẽ xảy ra, nó sẽ đẩy 0 vào đỉnh của Stack sau đó giảm thanh ghi ESP tuy theo kích thước của toán hạng.

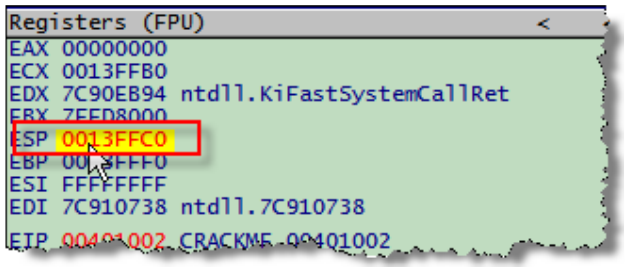
Quan sát cửa sổ Stack trước khi chúng ta thực thi câu lệnh PUSH.Lưu ý giá trị của thanh ghi ESP có thể khác nhau tùy theo từng máy.



Okie, ban đầu khi chưa thực hiện câu lệnh gì thì cửa sổ Stack của tôi giống như hình minh họa ở trên.Đây là những giá trị khởi tạo ban đầu của Stack do đó trên máy của bạn có thể khác máy của tôi chứ không nhất thiết phải giống nhau hoàn toàn.Bây giờ tôi sẽ thực hiện câu lệnh PUSH bằng cách nhấn F8 để trace qua câu lệnh này, ngay lập tức các bạn sẽ thấy được sự thay đổi :

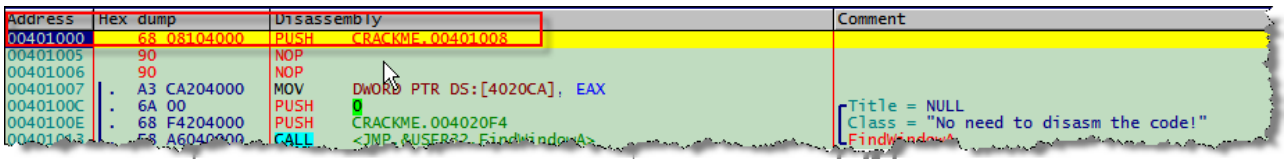


Như các bạn thấy giá trị tại đỉnh của Stack đã thay đổi là `0x0013FFC0` và tại đây lưu trữ giá trị 0x0 do câu lệnh PUSH thực hiện. Bạn hãy tưởng tượng như chúng ta có 1 chồng đĩa có sẵn, giờ ta muốn cất thêm một cái đĩa lên trên chồng đĩa này thì ta phải tịnh tiến chồng đĩa đi để dành ra một khoảng không gian cho ta đặt cái đĩa mới, sau khi có được khoảng không gian vừa đủ ta chỉ việc đặt cái đĩa mới lên trên. Vậy là khi chúng ta muốn lấy đĩa ra thì cái đĩa mới được cất vào sẽ được lấy ra đầu tiên.
Thanh ghi liên quan trực tiếp đến Stack là ESP, thanh ghi này luôn trỏ vào đỉnh của Stack. Quan sát tại cửa sổ Registers chúng ta thấy được như sau :

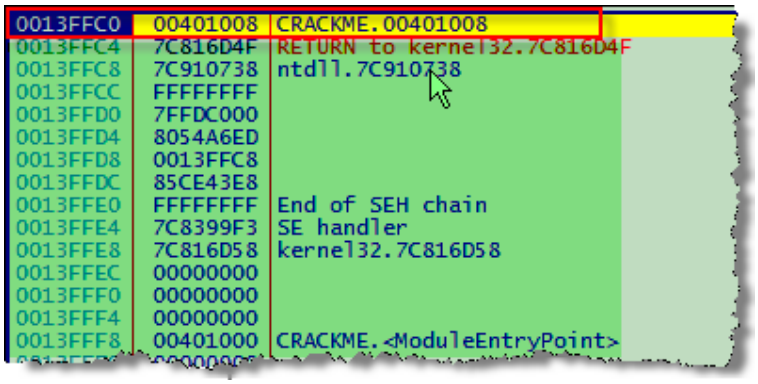


Có nhiều cách thức để đẩy dữ liệu vào Stack chứ không đơn thuần như chúng ta thấy ở trên. Ví dụ nếu như tôi thực hiện câu lệnh PUSH EAX, thì lệnh này sẽ đẩy giá trị của thanh ghi EAX vào đỉnh Stack. Chúng ta sẽ tìm hiểu thêm sự khác nhau giữa câu lệnh PUSH và PUSH [].

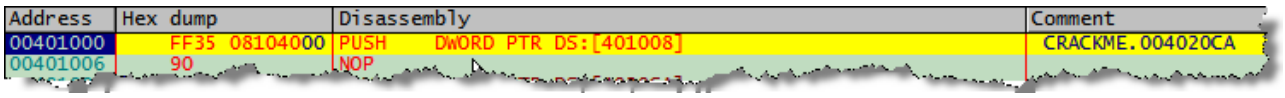
Giả sử tôi có câu lệnh `PUSH 401008`. Khi thực hiện câu lệnh này nó sẽ đẩy `401008` vào đỉnh của Stack. Quan sát hình minh họa dưới đây :

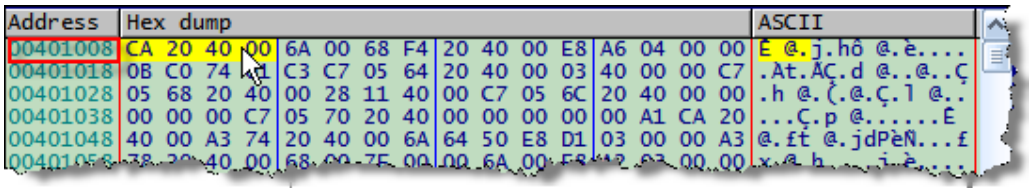


Thực hiện lệnh PUSH, quan sát trên stack :

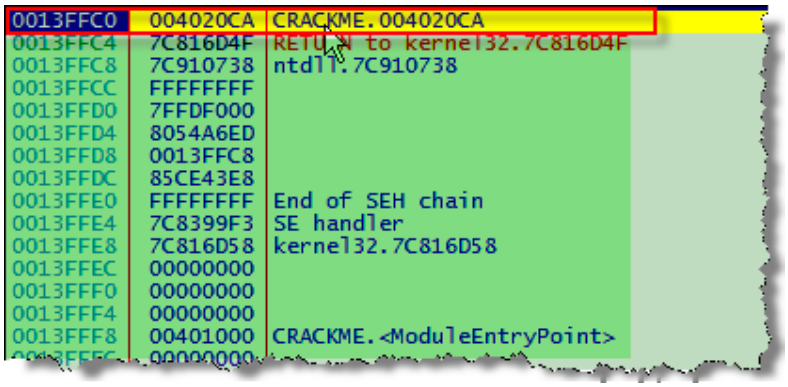


Tuy nhiên nếu như tôi thực hiện câu lệnh `PUSH [401008]` thì điều gì sẽ xảy ra trên Stack? Lệnh này sẽ đẩy nội dung của bộ nhớ tại 401008 vào đỉnh của Stack, chúng ta sẽ quan sát cửa sổ DUMP xem tại 401008 lưu trữ giá trị gì.



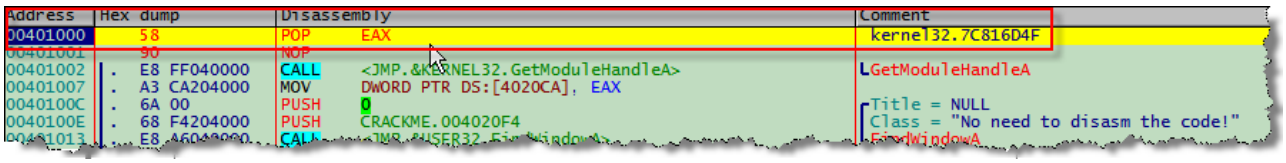


Chúng ta thấy được giá trị là `CA 20 40 00`. Khi cất vào stack nó sẽ được đảo ngược lại thành `004020CA`, giống như dòng Comment như bạn quan sát thấy ở hình bên trên. Nhấn F8 để thực hiện lệnh PUSH và quan sát kết quả trên Stack.

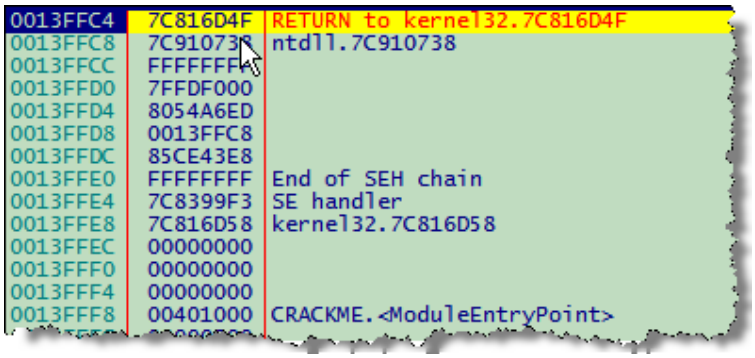


2.b. Lệnh POP

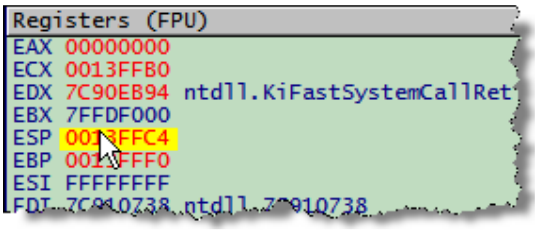
Lệnh POP có ý nghĩa hoàn toàn ngược lại với lệnh PUSH.Nó được sử dụng để lấy ra phần tử (giá trị) từ đỉnh của Stack và đặt vào một nơi mà chúng ta chỉ đỉnh để nhận giá trị được lấy ra.Lấy ví dụ, ta thực hiện câu lệnh POP EAX, điều này có nghĩa là chúng ta sẽ lấy giá trị tại đỉnh của Stack và lưu nó vào thanh ghi EAX, đồng thời giá trị tại đỉnh của Stack sẽ được thay đổi để trỏ tới phần tử tiếp theo. Lấy một ví dụ minh họa, ta thay thế lệnh PUSH 0 bằng lệnh POP EAX :



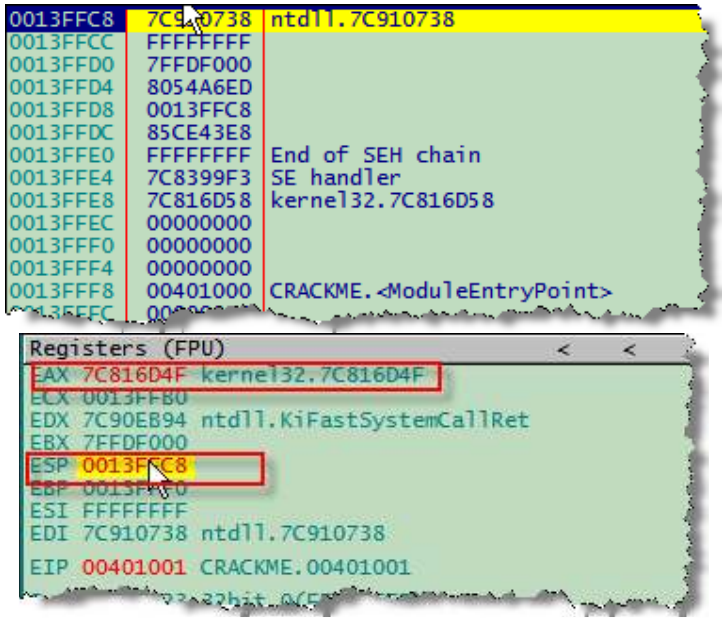
Giá trị tại đỉnh của Stack trước khi thực hiện lệnh POP :



Giá trị của thanh ghi ESP đang trỏ vào đỉnh của stack, giá trị của thanh ghi EAX lúc chưa thực hiện POP :



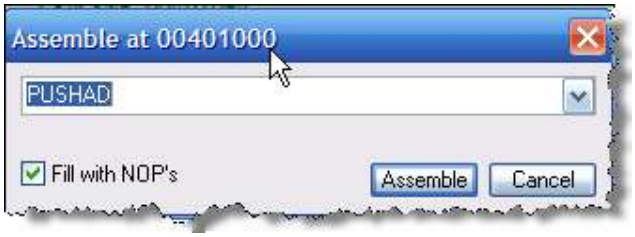
Bây giờ chúng ta nhấn F8 để trace và thực hiện lệnh POP EAX.Quan sát cửa số Stack và cửa sổ Registers chúng ta có được kết quả như sau :



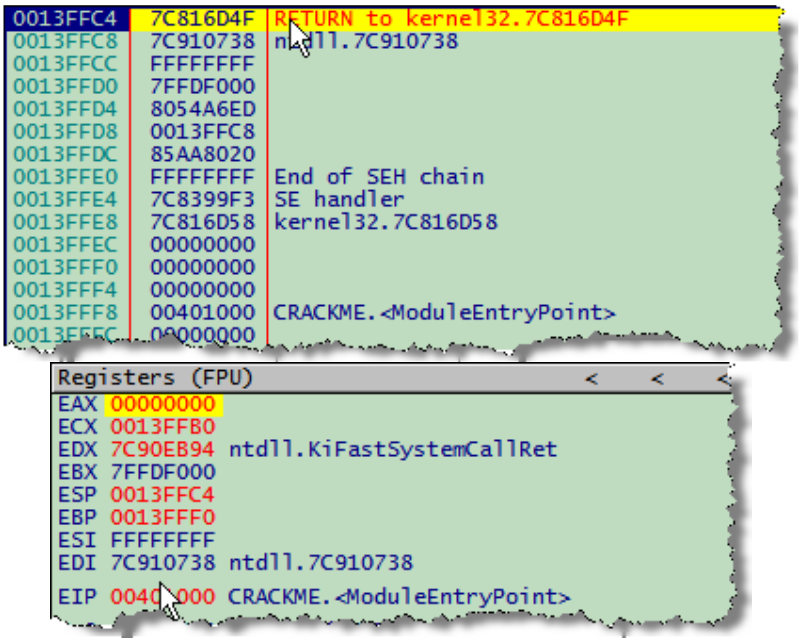
Ok, như vậy là các bạn đã có cái nhìn tổng quan về 2 lệnh PUSH và POP.Ta chuyển sang lệnh khác đó là hai lệnh PUSHAD và POPAD, đây là hai lệnh không kém phần quan trọng. Tại sao ta phải lưu ý tới câu lệnh này, lý do đơn giản đó là hai câu lệnh này thường được sử dụng để nhận biết các packers có cơ chế thực hiện gần giống với một Packer nổi tiếng mà có thể các bạn đã biết hoặc đã từng nghe tới đó là : UPX.

2.c. Lệnh PUSHAD

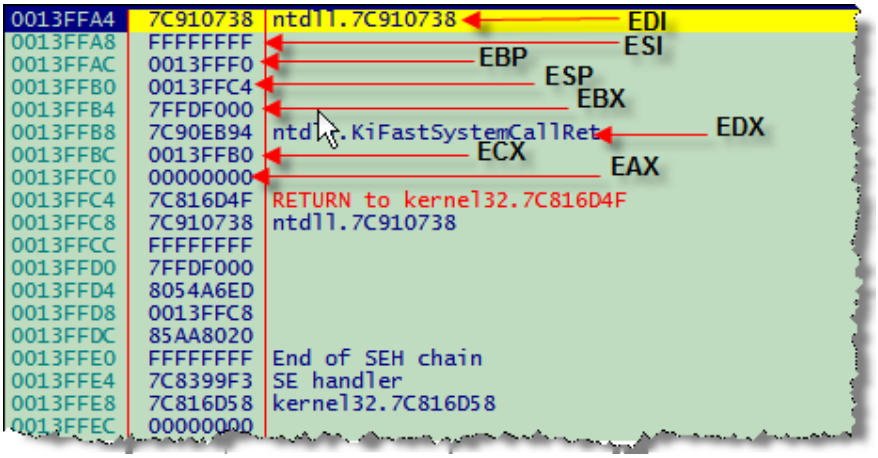
Lệnh PUSHAD khi thực hiện sẽ lưu tất cả nội dung của các thanh ghi vào trong Stack. Vì vậy có thể nói lệnh PUSHAD sẽ tương đương với một loạt các câu lệnh Push như sau : `PUSH EAX, ECX, EDX, EBX, ESP, EBP, ESI, EDI`. Chúng ta sẽ quan sát câu lệnh này trong Olly, load crackme vào và thay câu lệnh Push 0 bằng lệnh PUSHAD.



Quan sát cửa sổ Stack và cửa sổ Registers trước khi thực hiện lệnh :



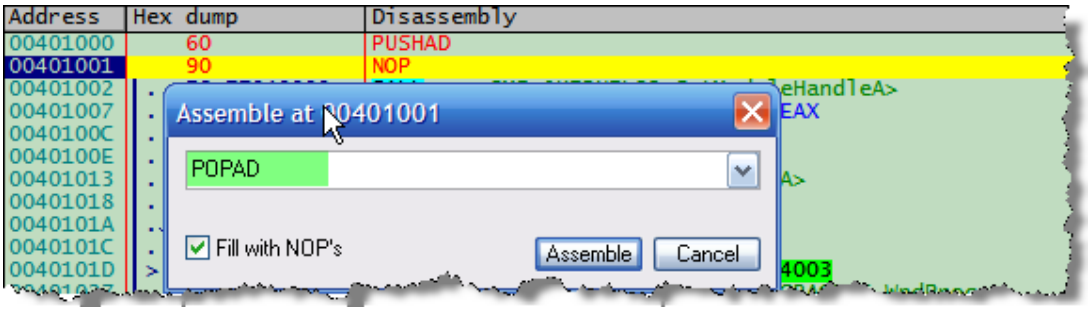
Nhấn F8 để thực hiện lệnh, đồng thời quan sát cửa sổ Stack các bạn sẽ thấy được như sau :



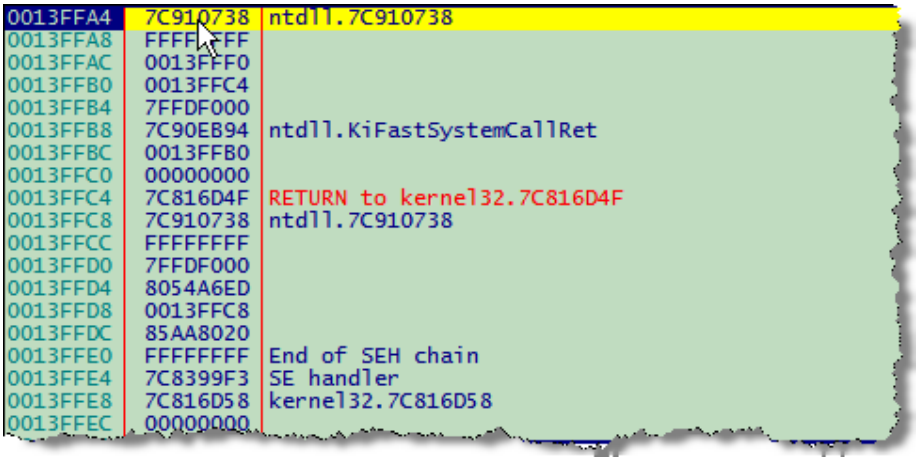
Theo hình minh họa trên các bạn thấy rằng các thanh ghi lần lượt được đẩy vào Stack theo thự tự từ EAX cho tớ EDI. Vậy giá trị của EDI sẽ nằm ở đỉnh của Stack và lúc này ESP sẽ có giá trị là đỉnh của Stack : 0x0013FFA4.

2.d. Lệnh POPAD

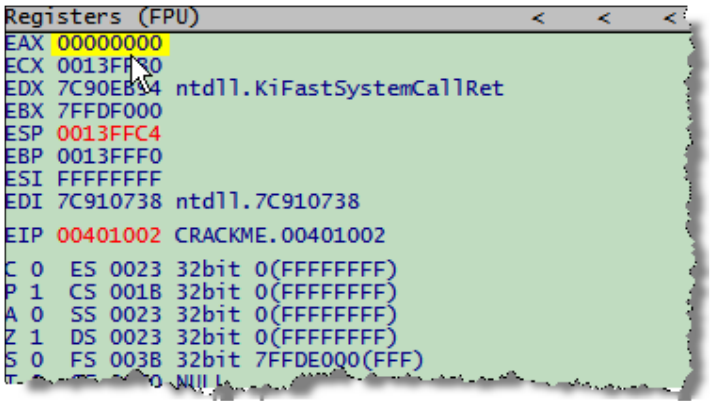
Lệnh POPAD thực hiện nhiệm vụ ngược lại với những gì lệnh PUSHAD đã thực hiện.Nó sẽ lấy giá trị tại Stack và cất vào các thanh ghi theo thứ tự từ EDI về EAX. Chính vì vậy câu lệnh này sẽ tương đương với : `POP EDI, ESI, EBP, ESP, EBX, EDX, ECX, EAX`. Quan sát minh họa bằng Olly sẽ cho bạn thấy một cách trực quan :



Các giá trị tại Stack :



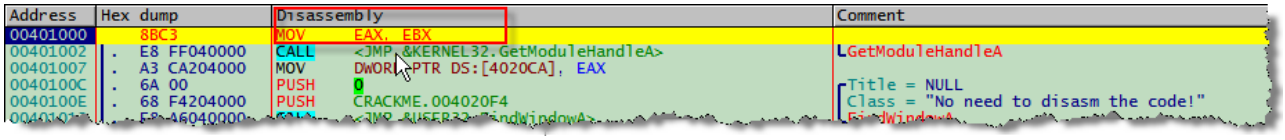
Nhấn F8 để thực hiện lệnh POPAD. Quan sát tại cửa sổ Registers chúng ta thấy được giá trị của các thanh ghi đã được phục hồi :



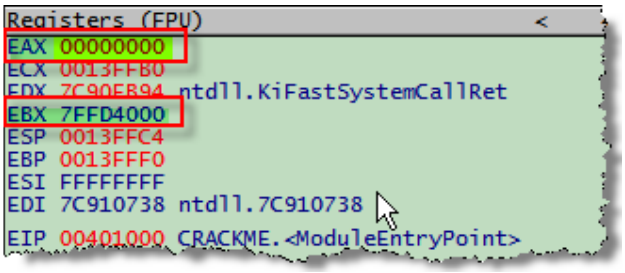
Qua đây ta có thể rút ra một kết luận nho nhỏ là cặp lệnh PUSHAD và POPAD thường đi cùng với nhau. Nếu như ta thấy ở đâu đó trong mã của chương trình xuất hiện lệnh PUSHAD thì có nghĩa là đâu đó ở bên dưới chắc chắn sẽ có câu lệnh POPAD.

3. Câu lệnh MOV

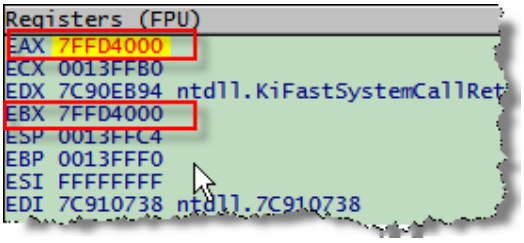
Lệnh MOV được sử dụng để chuyển dữ liệu giữa các thanh ghi, giữa một thanh ghi và một ô nhớ hoặc chuyển trực tiếp một số vào một thanh ghi hay ô nhớ.Một ví dụ minh họa : MOV EAX, EBX. Khi câu lệnh này thực thi nó sẽ chuyển dữ liệu của thanh ghi EBX tới thanh ghi EAX, quan sát trên Olly các bạn sẽ thấy như sau :



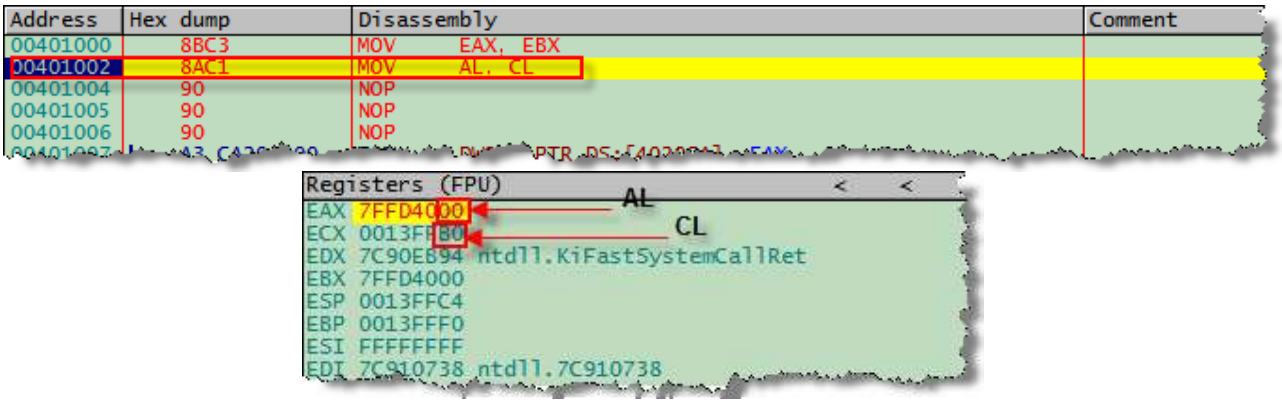
Trên cửa sổ Registers chúng ta quan sát thấy giá trị của 2 thanh ghi EAX và EBX trước khi thực hiện lệnh như sau :



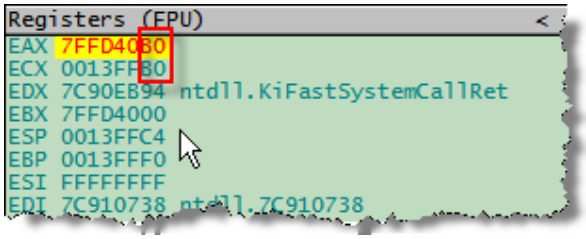
Okie, ta nhấn F8 để thực hiện câu lệnh `MOV EAX, EBX` và quan sát sự thay đổi giá trị của thanh ghi EAX tại cửa sổ Registers :



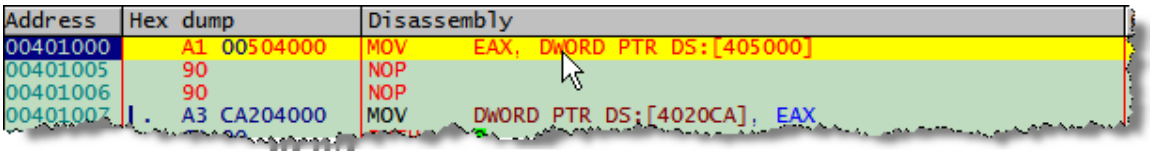
Điều này khiến bạn liên tưởng tới ngôn ngữ lập trình bậc cao khi chúng ta thực hiện một câu lệnh gán A:=B. Ở đây EAX được gán cho giá trị của EBX. Ở ví dụ này các bạn sẽ thấy EAX lưu trữ dữ liệu giống hệt EBX, tuy nhiên trong một vài trường hợp đặc biệt tôi chỉ muốn lấy một phần giá trị thôi thì thế nào? Câu lệnh MOV cũng hỗ trợ các bạn thực hiện việc này. Lấy ví dụ cụ thể, tôi thực hiện cậu lệnh `MOV AL, CL` :



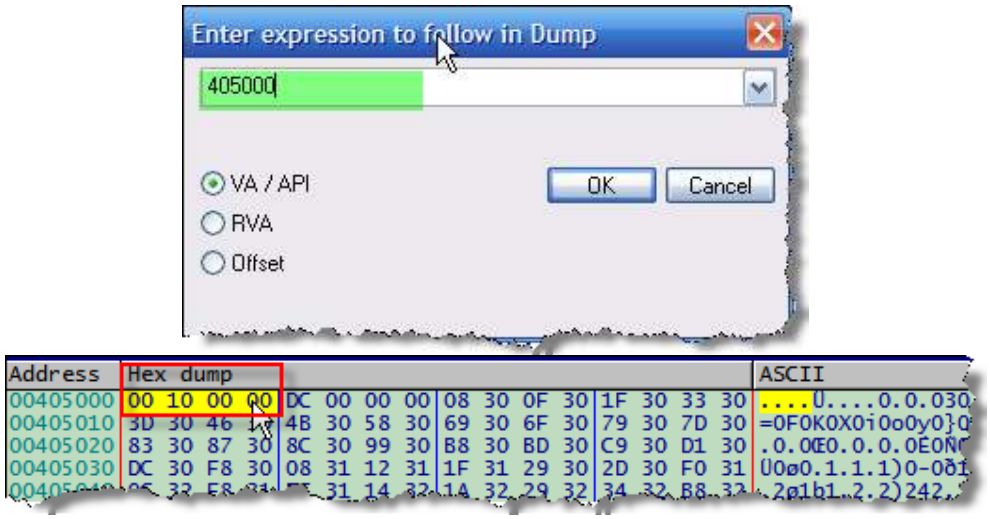
Nhấn F8 thực hiện câu lệnh MOV AL, CL. Quan sát cửa sổ Registers bạn sẽ thấy như hình minh họa dưới đây :



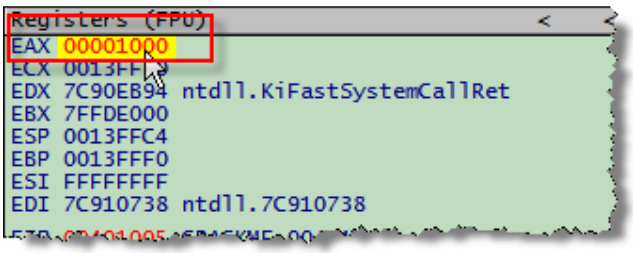
Ngoài việc chuyển dữ liệu qua lại giữa các thanh ghi như những gì các bạn đã nhìn thấy và quan sát ở trên chúng ta còn có lệnh MOV dùng để chuyển nội dung của một ô nhớ vào một thanh ghi.Chẳng hạn chúng ta sẽ thực hiện câu lệnh bên dưới đây :



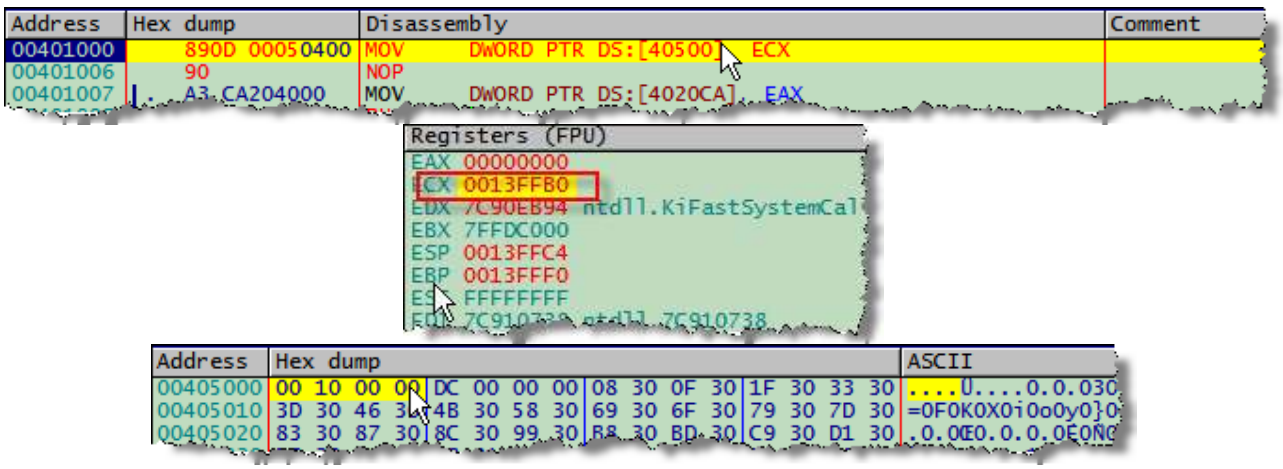
Trong trường hợ này thì nội dung của 405000 sẽ được chuyển vào thanh ghi EAX.Cụm từ DWORD như các bạn nhìn thấy ở trên có nghĩa là toàn bộ 4 bytes tại ô nhớ trên sẽ được chuyển vào EAX.Bây giờ chúng ta vào cửa sổ DUMP và quan sát xem tại 405000 có giá trị như thế nào :



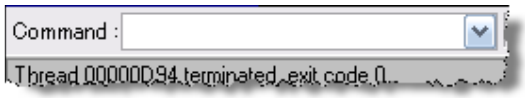
Như ta thấy, tại 405000 là 00 10 00 00, khi thực hiện câu lệnh MOV thì EAX sẽ giữ giá trị là 00001000. Nhấn F8 thực thi lệnh và kiểm tra kết quả :



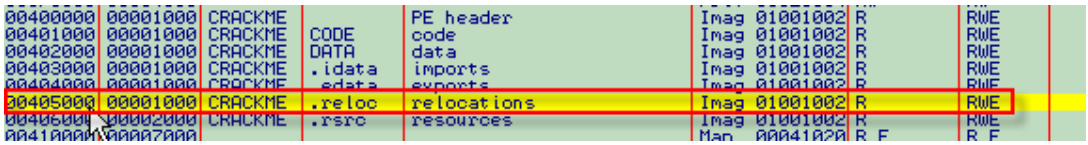
Bây giờ chúng ta thử thực hiện ngược lại và chuyển dữ liệu của một thanh ghi vào ô nhớ xem thế nào.Giả sử tôi muốn thực hiện lệnh : MOV DWORD PTR DS:[40500], ECX.



F8 thực hiện lệnh MOV, ặc chương trình bị terminate ngay lập tức. Không có lý do gì mà câu lệnh này không thể thực hiện được? Vậy tại sao lại thế ?

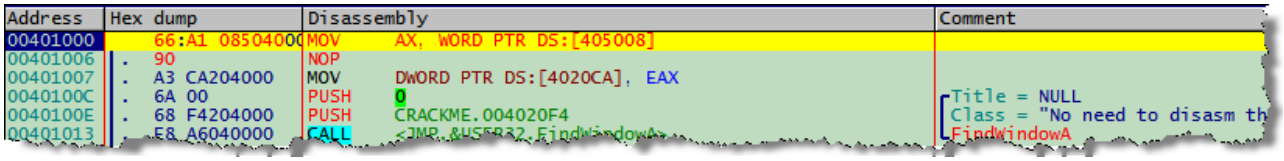


Câu trả lời hết sức đơn giản là chúng ta không có quyền ghi tại ô nhớ đó. Để kiểm tra xem có đúng không ta nhấn Alt + M để mở cửa sổ memory và thấy được như sau :

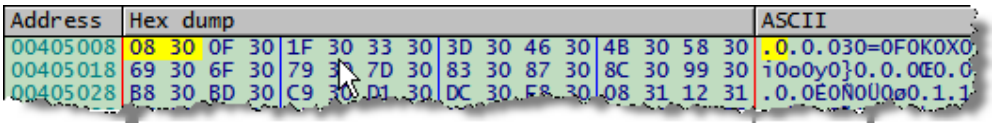


Vây là đã rõ tại 405000 chúng ta chỉ có quyền Read mà thôi, chính vì vậy khi chúng ta thực hiện lệnh MOV liền bị terminate chương trình ngay lập tức ☺.

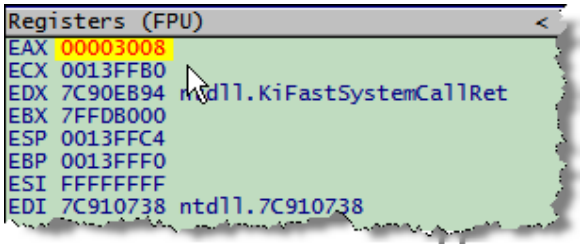
Nếu như tôi muốn chuyển 2 bytes hay 1 bytes dữ liệu của một ô nhớ vào một thanh ghi thì ta có thể dùng WORD và BYTE trong câu lệnh MOV. Ví dụ tôi có lệnh sau đây :
MOV AX, WORD PTR DS:[405008]



Tại sao ở đây chúng ta không sử dụng thanh ghi EAX đó là vì ở đây chúng ta chỉ muốn lấy 2 bytes (16 bit) thôi nên phải sử dụng thanh ghi AX là thanh ghi 16 bits. Điều này cũng đã được qui định bởi lệnh MOV : toán hạng đích và gốc có thể tìm được theo các chế độ địa chỉ khác nhau nhưng phải có cùng độ dài và không được phép đồng thời là 2 ô nhớ hoặc 2 thanh ghi đoạn. Ok ta chuyển qua cửa sổ DUMP và xem nội dung của ô nhớ 405008 :

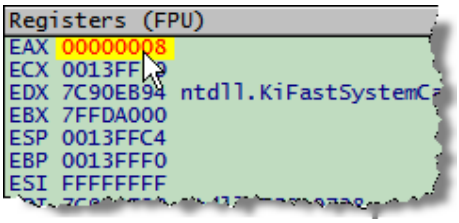


Nhấn F8 để thực hiện câu lệnh MOV và quan sát thanh ghi AX ta có được kết quả như sau :



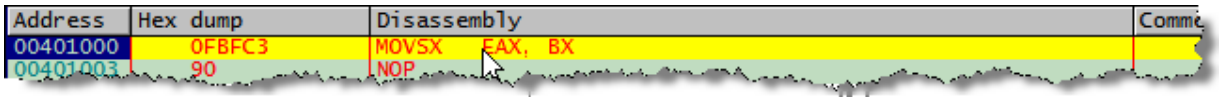
Giờ chúng ta muốn lấy 1 bytes thì thay thế thanh ghi AX bằng thanh ghi AL. Do đó câu lệnh MOV sẽ là như sau : MOV AL, BYTE PTR DS:[405008]

Vậy khi thực hiện lệnh MOV thanh ghi AL sẽ nhận giá trị như hình minh họa dưới đây :



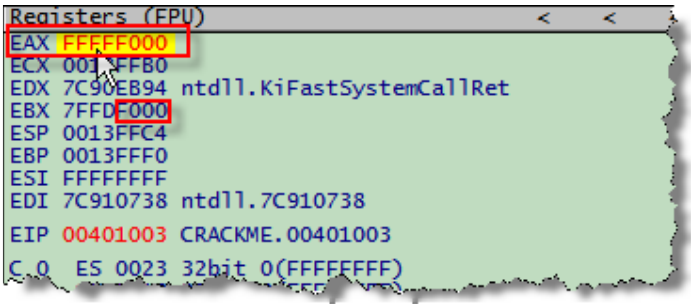
4. Câu lệnh MOVSX

Câu lệnh này sẽ thực hiện sao chép nội dung của toán hạng thứ hai, có thể là thanh ghi hoặc ô nhớ (với điều kiện toán hạng thứ hai phải có độ dài nhỏ hơn toán hạng thứ nhất) vào toán hạng thứ nhất đồng thời sẽ điền đầy các bit bên trái của toán hạng thứ nhất bằng bít có trọng số cao nhất của toán hạng thứ hai.Chúng ta sẽ lấy ví dụ cụ thể để minh họa, giả sử trong Olly chúng ta bắt gặp lệnh sau :

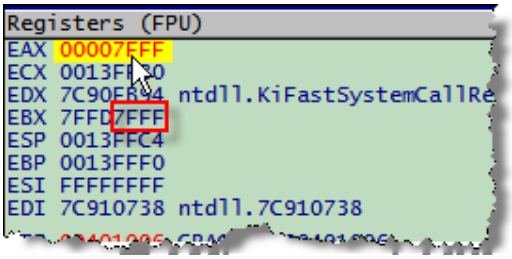


Để biết nội dung của 2 thanh ghi EAX và BX trước khi thực hiện câu lệnh trên sẽ như thế nào ta có thể nhờ cậy đến cửa sổ Tip Window.Cửa sổ này sẽ cho chúng ta thông tin chi tiết :

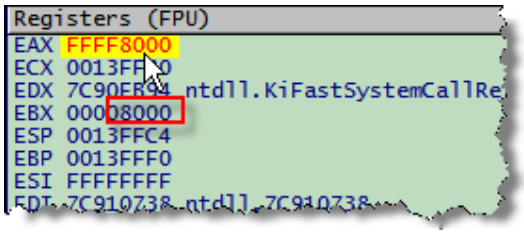
OK thanh ghi EAX là 0x0 còn BX là 0xF000.Bây giờ chúng ta nhấn F8 để thực hiện lệnh và quan sát trên cửa sổ Registers :



Như trên ta thấy rằng nội dung của BX đã được sao chép sáng AX và điền phần còn lại với giá trị là FFFF, đó là bởi vì thanh ghi BX nếu xét cụ thể thì nó biểu diễn cho số âm, mà số âm thì bít có trọng số cao nhất sẽ là 1. Do đó số 1 này sẽ được điền hết vào trong thanh ghi EAX như các bạn đã thấy trên hình. Nếu như thanh ghi BX biểu diễn một số dương chẳng hạn 0x1234 thì thanh ghi EAX sẽ có giá trị như sau : 0x00001234. Để chứng mình ta giả sử thanh ghi EBX biểu diễn một số dương là 0x7FFF vậy khi thực hiện lệnh MOV ta sẽ được như sau :



Nếu BX là 0x8000 thì thanh ghi EAX sẽ có giá trị là 0xFFFF8000.

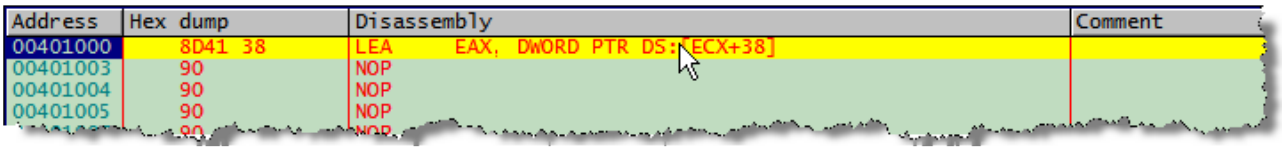


5. Câu lệnh MOVZX

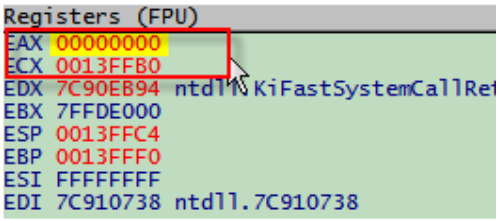
Lệnh này cũng tương tự như lệnh MOVSX, nhưng thay vì phụ thuộc vào bít dấu thì phần bên trái luôn luôn được điền đầy bằng những con số 0. Bạn tự kiểm chứng trong Olly nhé ☺.

6. Câu lệnh LEA

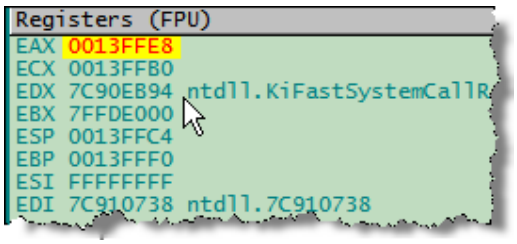
Lệnh này cũng tương tự như lệnh MOV nhưng khác ở chỗ toán hạng đầu tiên thường là các thanh ghi công dụng chung còn toán hạng thứ hai là một ô nhớ.Câu lệnh này thực sự rất hữu dụng khi ô nhớ này tương ứng với một phép tính toán trước đó. Lấy một ví dụ trực quan trong Olly :



Quan sát cửa sổ Registers để thấy được giá trị hiện thời của các thanh ghi khi chưa thực hiện lệnh :



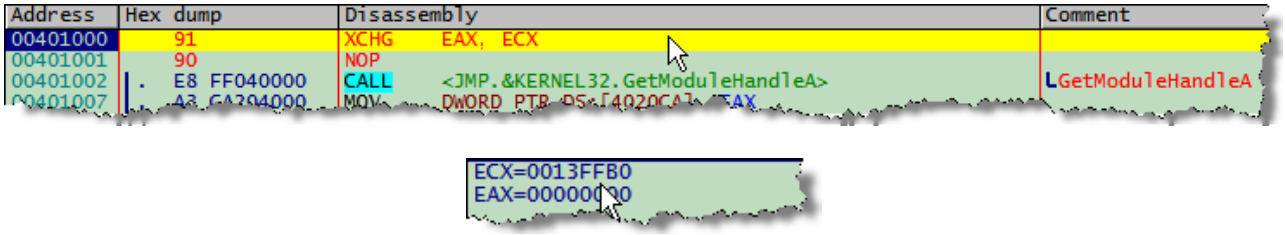
Trong trường hợp tại máy của tôi thì giá trị thanh ghi ECX là 0x0013FFB0, mà trong lệnh trên tôi cho thanh ghi ECX + 38 vậy tức là tôi sẽ có giá trị = 0x0013FFE8. Nếu hiểu như lệnh MOV bạn sẽ nghĩ là nó sẽ chuyển dữ liệu tại ô nhớ có giá trị ở trên vào thanh ghi EAX nhưng ở đây thì không phải như thế. Nó sẽ nạp giá trị của ECX + 38 vào thanh ghi EAX, kết quả cuối cùng là thanh ghi EAX sẽ có giá trị là 0x0013FFE8. Lập luận là như vậy, bây giờ ta thử nhấn thử F8 để thực hiện lệnh và kiểm tra kết quả xem có chính xác hay không ☺



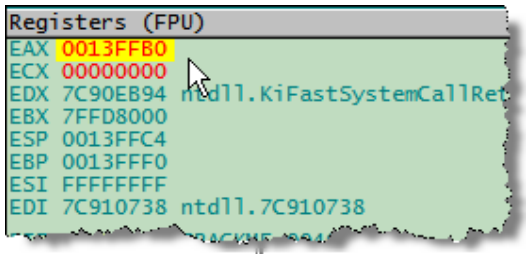
Trong quá trình làm việc nhiều với các chương trình các bạn sẽ hiểu rõ hơn về lệnh LEA.

7. Câu lệnh XCHG

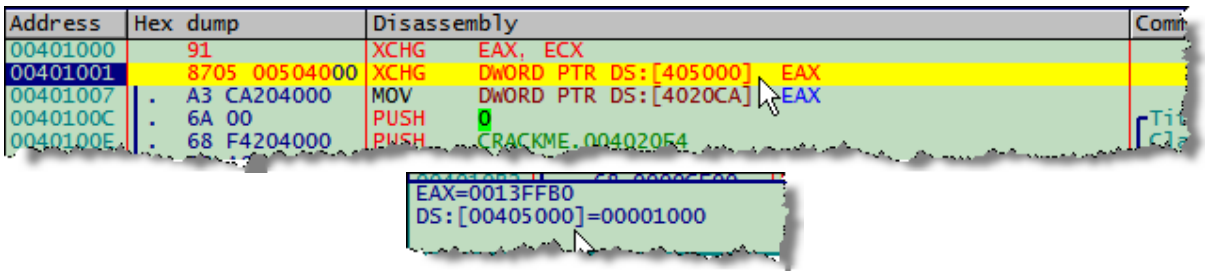
Câu lệnh này còn được gọi là lệnh tráo đổi nội dung của hai toán hạng. Lệnh XCHG được dùng để hoán chuyển nội dung của hai thanh ghi, thanh ghi và một ô nhớ.Chúng ta thử kiểm tra bằng ví dụ trong Olly, tôi có một lệnh đơn giản như sau :



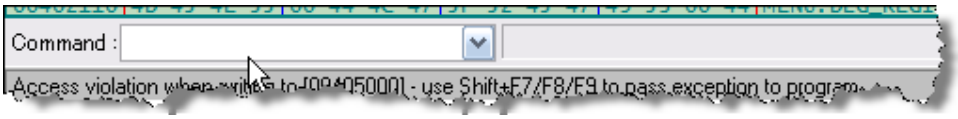
Nhìn vào câu lệnh trên các bạn hoàn toàn có thể suy ra ngay được kết quả của câu lệnh này sau khi thực hiện ☺. Thanh ghi EAX sẽ mang giá trị của ECX và ngược lại.



Cũng tương tự trong trường hợp câu lệnh MOV, bạn cũng không thể thực hiện lệnh XCHG giữa thanh ghi và một ô nhớ nếu như ô nhớ đó không có quyền ghi mà chỉ có quyền đọc.



Tôi thử thực hiện câu lệnh XCHG ở trên, câu lệnh này không thực hiện được và tôi nhận được một Exception :



Okie tôi nghĩ đến đây là đã đủ cho một bài viết và phần 4 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