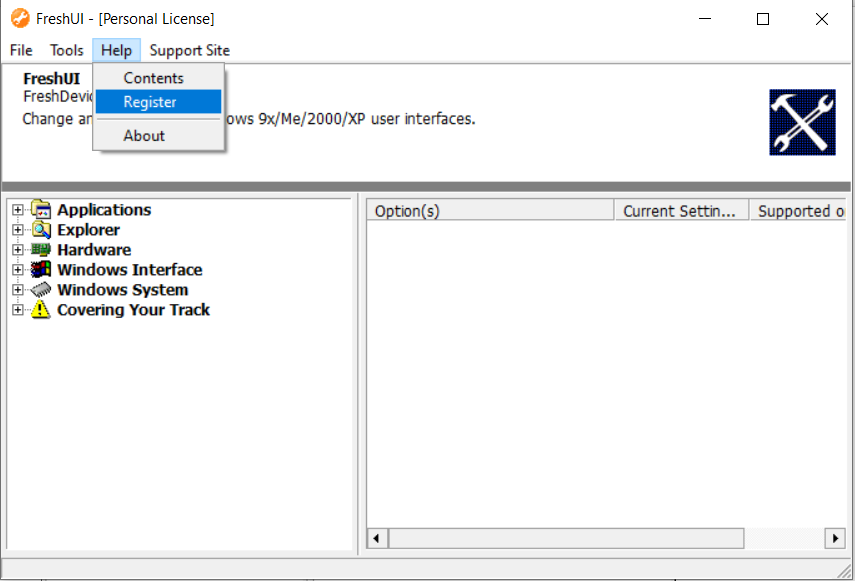
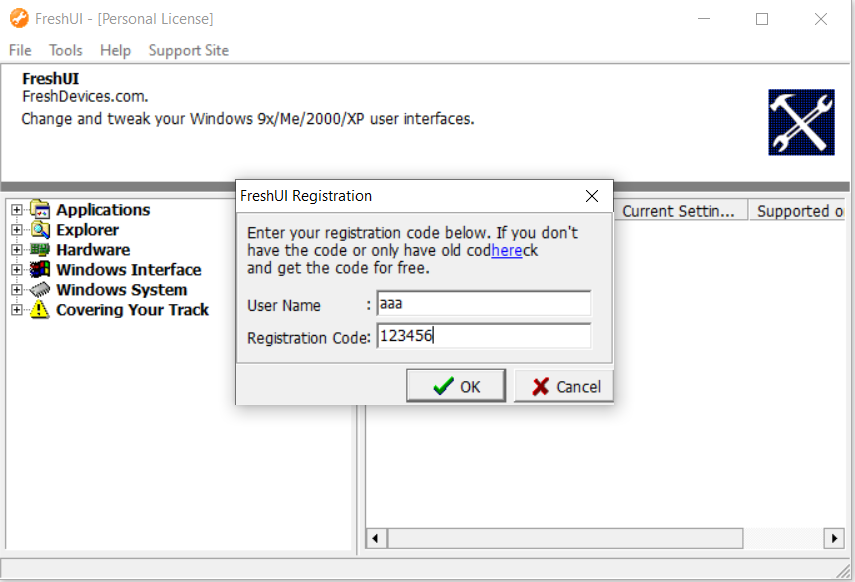
**3\_2:**

1. **Tìm GOODBOY, BADBOY:**



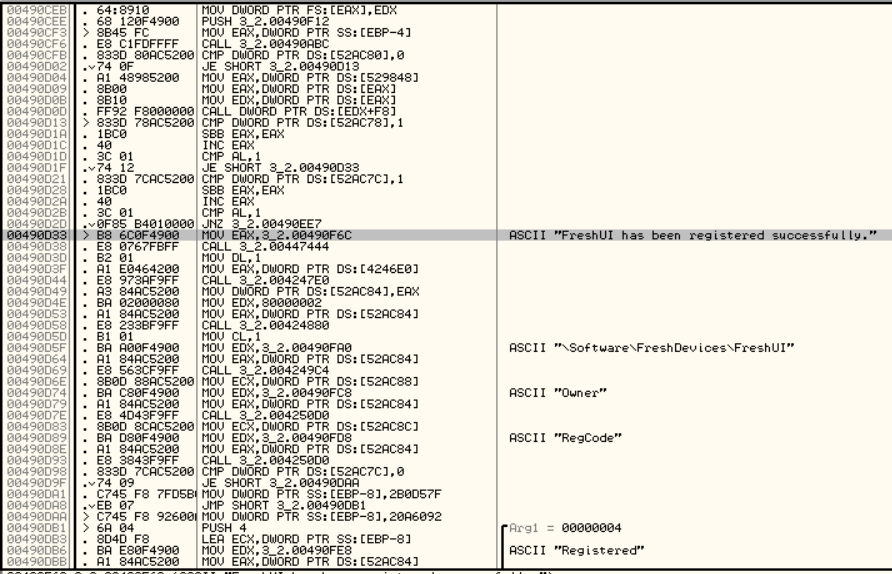
* Mở chương trình và chọn Help🡪Register.



* Nhập thử một User Name, một Registration Code và chọn OK. Chương trình không hiện thông báo nhưng ta hiểu rằng register không thành công. **Như vậy chương trình có BADBOY nhưng không hiển thị thông báo.**



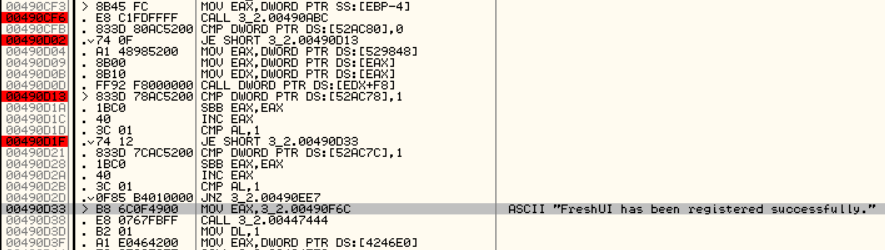
* Mở chương trình bằng OllyDbg và dùng chức năng tìm kiếm chuỗi để tìm thông báo GOODBOY.



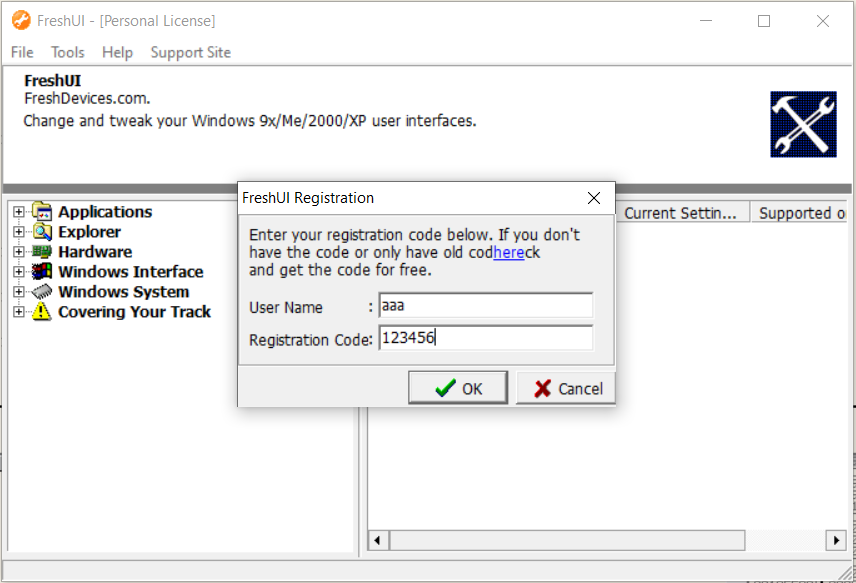
* Đi tới đoạn code chứa GOODBOY.

1. **Phân tích code:**

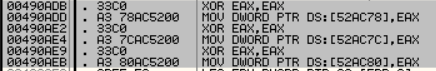
* Nhìn vào địa chỉ dòng lệnh báo *“FreshUI has been registered successfully”*, ở trên ta thấy có một lệnh nhảy tới thông báo này, đặt breakpoint ở đó. Trên dòng **JE SHORT 3\_2.00490D33** có lệnh **CMP AL, 1**. Ta hiểu là sau khi thực hiện hàm nào đó, nếu giá trị AL = 1 thì nhảy tới thông báo success. Dò lên trên tìm lệnh gọi hàm gần nhất:



* Để thực thi lệnh **JE SHORT 3\_2.00490D33**, ta cần phải thực hiện lệnh **JE SHORT 3\_2.00490D13**. Như vậy, phải cho giá trị ở ô nhớ **[52AC80] = 0** để có thể thực hiện lệnh jump. Ta xét lệnh **CALL 3\_2.00490ABC**, đây là hàm gần nhất để nhảy tới lệnh thông báo nhập key đúng. Đặt breakpoint ở dòng CALL này, nhấn F9 để chạy tới khi gặp lệnh này thì ngưng lại.



* Nhập thử giá trị cho User Name và Registration Code và nhấn OK. Tiếp tục nhấn F7 để đi tới đoạn code mà lệnh **CALL 3\_2.00490ABC** gọi tới. Chú ý đoạn code đánh dấu sau:



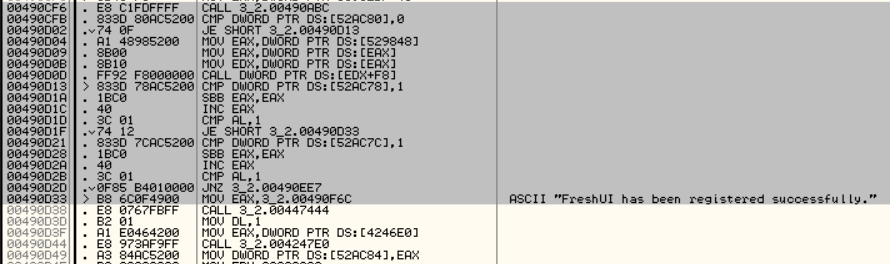
* Ta thấy những ô nhớ **[52AC78, [52AC7C]** và **[52AC80]** đều là những ô nhớ mang giá trị quyết định cho điều kiện lệnh nhảy để nhảy tới GOODBOY như những hình trên. Gán giá trị những ô nhớ đó bằng EAX, mà **XOR EAX, EAX** nghĩa là gán **EAX=0**. Như vậy, giá trị ô nhớ bằng EAX và bằng 0 => **Có thể đây là ô nhớ chứa biến Bool dùng kiểm tra xem key nhập vào đúng hay sai.**
* Để kiểm tra giả thuyết, ta chọn dòng có ô nhớ [52AC78] 🡪 nhấn chuột phải 🡪 Find references to 🡪 Address constant. Một hộp thoại với những dòng lệnh có gọi ô nhớ [52AC78] sẽ xuất hiện.



* Giá trị của ô nhớ này chỉ thay đổi bởi 3 lệnh, nhấn đúp vào từng lệnh để xem vị trí thì thấy nó đều nằm trong hàm chúng ta đang xét. Giá trị của ô nhớ được khởi tạo lần đầu khi gán với EAX, và chỉ thay đổi xoay quanh giá trị 0 và -1 => ***Giả thuyết đặt ra có thể đúng (nếu key đúng thì gán 0, key sai gán -1).***
* Làm tương tự với hai ô nhớ còn lại cũng cho kết quả tương tự. Tại sao lại cần tới 3 biến kiểm tra key? => ***Có thể có 3 loại key.***
* Tiếp tục F7 chạy chương trình, ta thấy có 3 vòng lặp như sau:



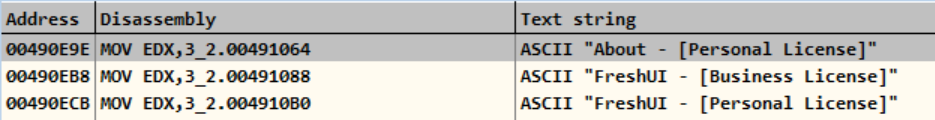
* Đây là đoạn code kiểm tra xem registration code ta nhập vào có khớp với key của chương trình hay không.
* Quay lại với đoạn code quyết định trả về GOODBOY hay không:



* Như đã trình bày ở trên, ba ô nhớ **[52AC80]**, **[52AC78]** và **[52AC7C]** là những ô nhớ lưu kết quả so sánh key từng loại với registration code người dùng nhập vào. Ở đây sau khi thực hiện hàm 3\_2.00490ABC, ta có 3 lệnh CMP tương ứng để kiểm tra xem liệu registration code có khớp với key của hệ thống hay không.
* Sau khi thực hiện lệnh ***CALL 3\_2.00490ABC***), kết quả sẽ trả về 3 ô nhớ **[52AC80]**, **[52AC78]** và **[52AC7C]**. Tuy nhiên giá trị ô nhớ không phải là cái quyết định registration code ta nhập đúng hay sai, mà là giá trị AL. Lệnh “CMP <ô nhớ>, 1” chỉ để tác động tới cờ CF. Nếu ô nhớ có giá trị 1, cờ CF=0 => **SBB EAX, EAX** thì EAX=0 => **INC EAX** thì EAX=1 => AL=1. Lúc đó ta mới tới được lệnh thông báo thành công.
* Chú ý là các ô nhớ không hề được gán giá trị 1 trong bất cứ dòng lệnh nào, nếu registration code ta nhập đúng thì nó sẽ lưu giá trị thành -1. Khi dùng lệnh ‘CMP <ô nhớ>, 1’, nếu ô nhớ có giá trị -1 thì không thay đổi cờ CF, từ đó AL=1 và chương trình vẫn sẽ thông báo ta nhập đúng.

1. **Các loại key:**

* Ta đã biết có 3 loại key trong chương trình nhưng chưa biết cụ thể đó là những loại key gì. Tìm trong bảng Search for all referenced text strings, ta thấy có dòng sau:

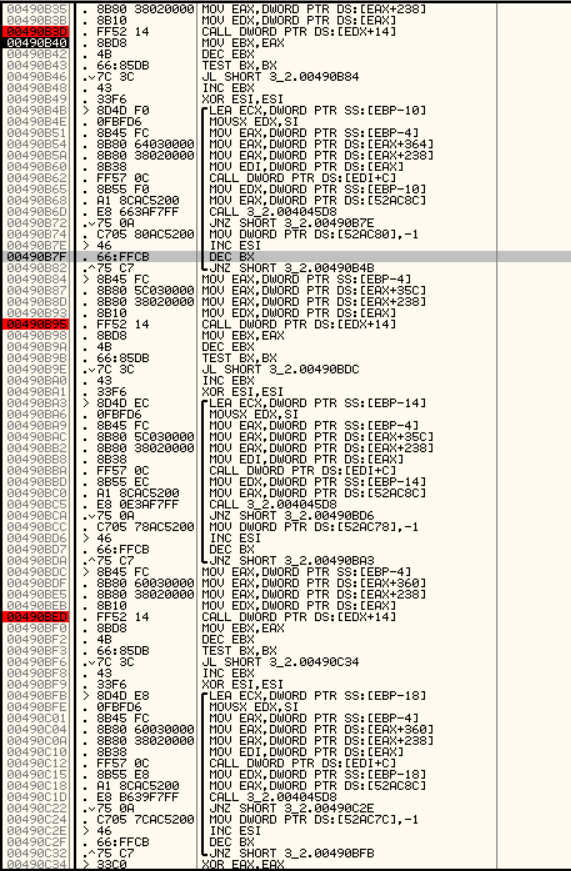


* Vậy một loại key là **Personal**, một loại key là **Business**. Và còn một loại key nữa.
* Quay trở lại đoạn code sau khi gọi lệnh **CALL 3\_2.00490ABC**(hàm kiểm tra registration code ta nhập vào với key của hệ thống), đặt breakpoint ở lệnh **CMP [52AC80], 0**để kiểm tra. Nhấn F9 để chương trình chạy tới lệnh này, sau đó nhấn F7 tới câu lệnh **JE SHORT 3\_2.00490D13**và thay đổi cờ ZF bằng cách nhấn đúp vào giá trị cờ ZF. Như vậy lệnh JE sẽ không được thực hiện, tức là ta đang giả sử registration code đúng ở loại key này. Tiếp tục nhấn F7, chương trình FreshUI hiện ra thông báo như sau:



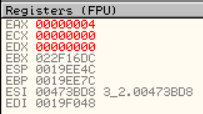
* Vậy loại key cuối cùng là **Old** và do ô nhớ **[52AC80]** đánh dấu.

Bây giờ ta sẽ tìm cách load key hệ thống và tìm tập hợp các key hệ thống trong chương trình. Thực hiện lại các bước đặt breakpoint tại lệnh **CALL 3\_2.00490ABC**, nhấn F9, nhập User Name, Registration Code và nhấn F7 để chạy vào lệnh CALL. Đặt breakpoint ở tất cả các câu lệnh **CALL DWORD PTR [EDX+14]**để xem số lượng key mỗi loại là bao nhiêu, nhấn F9 để chạy tới dòng đó, sau đó nhấn F8.



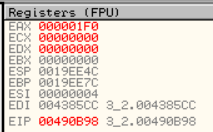
* Quan sát bên thanh ghi Register ta có kết quả như sau:

+ **CALL DWORD PTR [EDX+14]** đầu tiên:



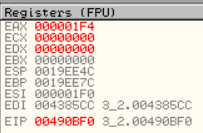
* Key loại Old có số lượng là 4 key

+ **CALL DWORD PTR [EDX+14]** thứ hai:

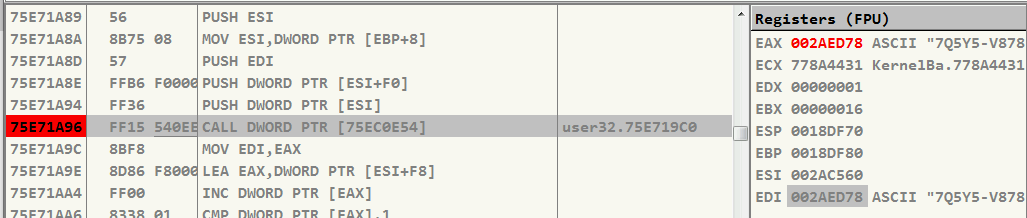


* Key loại Personal có 1F0 = 496 key

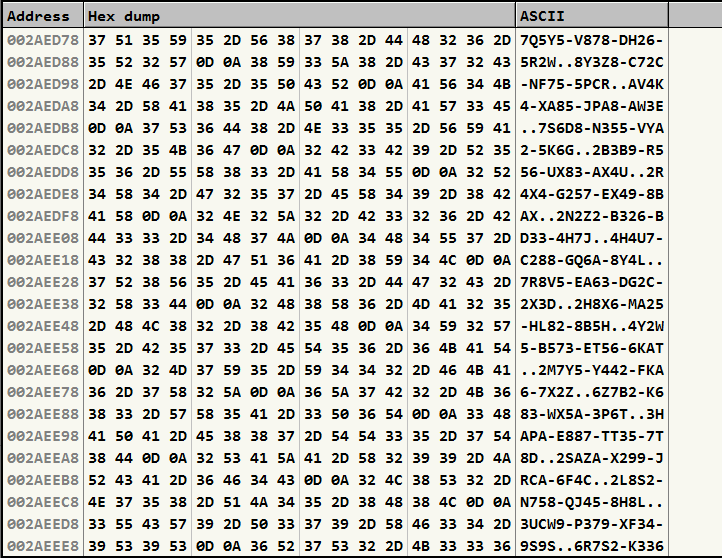
+ **CALL DWORD PTR [EDX+14]** thứ ba:



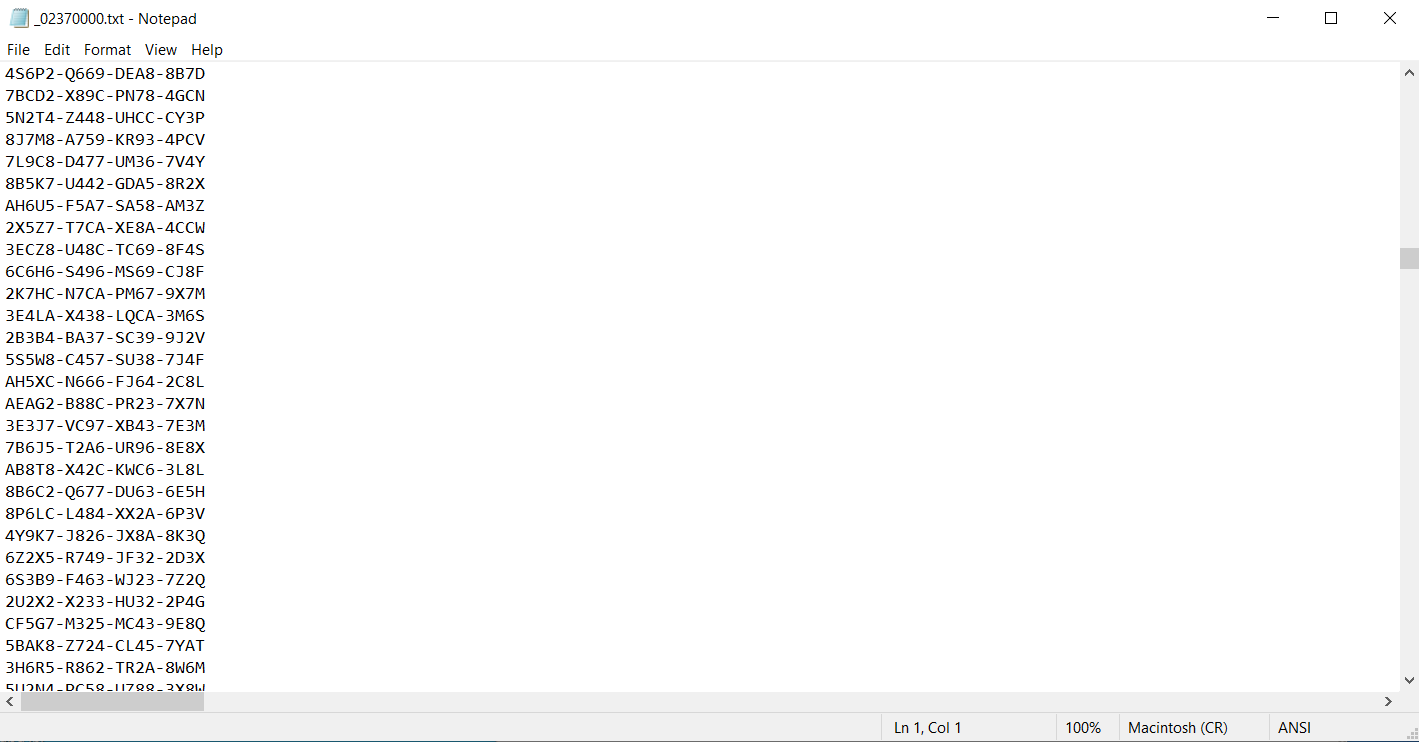
* Key loại Business có 1F4 = 500 key
* Để tìm được danh sách các key, ta đặt breakpoint ở dòng lệnh **CALL DWORD PTR [EDI+C]**, nhấn giữ F7 cho đến khi tới được dòng sau:



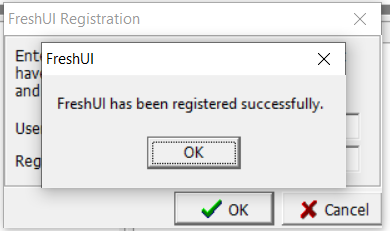
* Nhấn chuột phải vào ô nhớ EDI, chọn Follow in Dump, giá trị của ô nhớ đó sẽ hiện ra ở bảng dưới:



* Đây chính là danh sách các key. Ta xuất ra file bằng cách nhấn chuột phải vào ô nhớ bên cột Address 🡪 Backup 🡪 Save data to file. OllyDbg sẽ xuất dữ liệu ra cho chúng ta. Vào file vừa xuất ra, ta thấy có danh sách key ở đó:



* Hai loại key là Personal và Business sẽ xuất ra GOODBOY của chương trình khi chúng ta nhập vào:



* Key loại Old sẽ hiện thông báo key đã cũ và không chạy ra GOODBOY của chương trình.

