MASM – Microsoft Macro Assembler:

1. Cách Cài Đặt?

* Vào Link
* <https://www.masm32.com/download.htm>
* Click “US Site 2” + chờ nó tải + giải nén File vừa tải được File cài đặt + chạy File cài đặt + tiến hành cài đặt 1 cách mặc định
* Sau đó, trong thư mục “C:\Windows” sẽ xuất hiện thư mục “assembly”, và trong ổ đĩa C xuất hiện thư mục “masm32”, trong đây có chứa thư mục “help”, trong đó có các File hướng dẫn
* Thêm đường dẫn “C:\masm32\bin” vào biến hệ thống Path

1. Phần Mở Rộng File Assembly?

* “.asm”

1. Cú Pháp Assembly?

* Khác nhau với các trình biên dịch khác nhau, ở đây ta dùng cú pháp của MASM

1. Biên Dịch Và Chạy File Assembly?

* Mở CMD tại thư mục chứa File Assembly nhập

ml /c /Zd /coff <Tên File Assembly Có Phần Mở Rộng>

* Flag /c ngăn không sử dụng Linker
* Flag /Zd giúp thêm thông tin Debug vào File được tạo ra
* Flag /coff chỉ định File được tạo ra là Object
* Ví dụ

ml /c /Zd /coff foo.asm

* Khi đó sẽ có 1 File với phần mở rộng “.obj” cùng tên xuất hiện
* Tiếp tục nhập vào CMD

link /SUBSYSTEM:CONSOLE <Tên File Object Có Phần Mở Rộng>

* Tag /SUBSYSTEM:CONSOLE chỉ định liên kết với Console
* Ví dụ

link /SUBSYSTEM:CONSOLE foo.obj

* Khi đó sẽ có 1 File EXE xuất hiện, đây chính là File bạn muốn chạy
* Ngoài Console ta còn có thể liên kết với Windows, ví dụ khi muốn hiện 1 hộp thoại, thì nếu ta liên kết Console, hộp thoại sẽ bị nằm trong 1 CMD, còn khi liên kết với Windows, hộp thoại sẽ nằm tự do

1. Tài Liệu Hướng Dẫn?

* <https://doc.lagout.org/operating%20system%20/Windows/winasmtut.pdf>

Disassembler:

1. Cách Cài Đặt?

* Vào Link
* <https://hex-rays.com/ida-free/>
* Cuộn xuống Click vào “IDA Free for Windows” + chờ tải + chạy File vừa tải cài đặt mặc định
* Khi này sẽ xuất hiện thư mục “C:\Program Files\IDA Freeware 8.3”, trong đây chứa File “ida64” là File thực thi bạn muốn chạy
* Chạy File này, chọn các tùy chọn mong muốn lần đầu, rồi thoát, nhớ Untick “Display at startup”

1. Cách Mở Phần Mềm?

* Click vào biểu tượng “ida64” + Click “OK”
* Khi này cửa sổ IDA sẽ hiện ra, 1 lúc bạn chỉ có thể làm việc trên 1 File EXE

1. Phân Dịch 1 File?

* Click Tab “File” + chọn “Open” + chọn File mong muốn + Click “Open” + cửa sổ tùy chọn phân dịch hiện ra + chọn các tùy chọn mong muốn, thông thường để mặc định + Click “OK” + cửa sổ thông tin Debug hiện ra + chọn các tùy chọn mong muốn, thông thường để mặc định + Click “Yes” hoặc “No”, thông thường Click “Yes”
* Sau 1 lúc File sẽ được phân dịch và hiện trên màn hình

1. Mở Nhanh 1 Cửa Sổ IDA Trống Khác?

* Click Tab “File” + chọn “New instance” + Click “OK”

1. Đóng File Lại Nhưng Vẫn Mở Cửa Sổ IDA?

* Click Tab “File” + chọn “Close”

1. Mở Nhanh Các File Mới Mở Gần Đây?

* Click Tab “File” + chọn File mong muốn ở phần dưới cùng

Preprocessor Directive – Tiền Xử Lí Điều Hướng:

1. Khai Báo Mình Sẽ Sử Dụng Instruction Set Nào?

.<Tên Instruction Set>

* Ví dụ

.386

* Tức là ta sẽ sử dụng 386 Instruction Set, nghĩa là Instruction Set dành cho máy 32 Bit
* Dễ thấy cái này có dạng x86, x có thể là 3, 4, 6, …, càng lên cao thì càng mới, 386 là cũ nhất
* Kể từ bây giờ, ta chỉ làm việc trên máy 32 Bit

1. Khai Báo Mô Hình Bộ Nhớ, Cách Truyền Tham Số Cho Hàm, …?

.model <Các Mô Hình>

* Ví dụ

.model flat, stdcall

* Mô hình bộ nhớ flat là chương trình chạy trên bộ nhớ ảo tại đĩa cứng, không bị chia thành các Page rải rác
* Mô hình stdcall là khi bạn truyền tham số cho hàm, nó sẽ đọc các tham số từ phải sang trái
* Ví dụ hàm nhận 4 tham số theo thứ tự là A, B, C, D, thì khi bạn truyền tham số bạn phải truyền D, truyền C, truyền B, truyền A rồi mới gọi hàm

1. Import Thư Viện?

* Tất cả hàm mặc định nằm trong thư mục “C:\masm32\include\”, các hàm này lại sử dụng các thư viện trong “C:\masm32\lib\” để chạy
* Để Import thư viện

include C:\masm32\include\<Tên File Thư Viện Có Phần Mở Rộng Inc>

includelib C:\masm32\lib\<Tên File Cần Cho Thư Viện Có Phần Mở Rộng Lib>

* Ví dụ

include C:\masm32\include\kernel32.inc

includelib C:\masm32\lib\kernel32.lib

1. Để Chỉ Định Tên Nhãn Khác Nhau Khi 1 Cái Viết Hoa 1 Cái Viết Thường?

* Mặc định là không phân biệt hoa thường
* Để phân biệt hoa thường

option casemap :none

Data Type – Kiểu Dữ Liệu?

1. Các Kiểu Dữ Liệu Cơ Bản?

|  |  |
| --- | --- |
| Kiểu dữ liệu | Kích thước |
| db | 1 Byte |
| dw | 2 Byte |
| sdword, dd | 4 Byte |
| dq | 8 Byte |
| dt | 10 Byte |

* Thật ra các kiểu dữ liệu đều giống nhau, chỉ khác kích thước

Variable – Biến:

1. Khai Báo Biến Toàn Cục Lưu Ở Phần Nhớ Toàn Cục?

.data

<Tên Biến 1> <Kiểu Dữ Liệu 1> <Các Giá Trị Liên Tiếp 1>

<Tên Biến 1> <Kiểu Dữ Liệu 1> <Các Giá Trị Liên Tiếp 1>

…

* Ví dụ

.data

foo db "Con chim non", "bu cu", 123

bar dd 123, 456

bob dw "AC", "DE", 987

bab dq -16760255

boo dd 12.1414222717

* Phần nhớ toàn cục lúc này

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| C | o | n |  | c | h | i | m |  | n | o | n | { | 123 | 0 | 0 | 0 | 200 | 1 | 0 | 0 |
| C | A | E | D | 219 | 3 | 41 | 42 | 00 | FF | FF | FF | FF | FF | D | C | B | A |  |  |  |

* Địa chỉ tăng từ trái sang phải ở hàng đầu rồi tới hàng tiếp theo rồi tiếp theo, ô màu vàng là 8 Bit đã chuyển thành kí tự ASCII, ô màu cam là 8 Bit đã chuyển thành số thập phân, pp màu xanh là 8 Bit đã chuyển thành số thập lục phân
* Như sau là sẽ báo lỗi do số kí tự hoặc giá trị > kích thước <Kiểu Dữ Liệu>

.data

foo dd "ABCDE"

bar db 300

* Như vậy, các giá trị toàn cục sẽ đặt cạnh nhau liên tiếp trong phần nhớ toàn cục
* <Các Giá Trị Liên Tiếp> ngăn cách bởi dấu phẩy, và chiếm số lượng ô Byte = nhau
* 1 kí tự sẽ được chuyển sang 8 Bit nhị phân = bảng ASCII
* Kí tự trái hơn sẽ ở địa chỉ cao hơn, nói cách khác, MSB hơn
* Số Floating Point phải gán cho biến kích thước 4 Byte thì mới có hành vi bình thường, thứ tự các Bit khi chưa áp dụng Little Endian là 1 Bit dấu, 8 Bit Exponent, 23 Bit Mantissa, 32 Bit này sẽ chia thành 4 cụm 8 Bit rồi đảo ngược vị trí 4 cụm, sau đó đặt vào bộ nhớ ở trên
* Giá trị không được vượt quá kích thước <Kiểu Dữ Liệu>
* Tên biến sẽ đại diện cho địa chỉ của ô Byte đầu tiên hay có địa chỉ thấp nhất trong giá trị của biến
* Số nguyên âm sẽ chuyển sang dạng bù 2

1. Trả Về Khoảng Cách Của Biến Toàn Cục So Với Instruction Đầu Tiên?

offset <Biến Toàn Cục>

* Ví dụ Instruction đầu tiên của hàm main, hay ngõ vào của chương trình nằm ở địa chỉ 100, địa chỉ của <Biến Toàn Cục> là 240, thì lệnh trên sẽ trả về 140

Program – Chương Trình:

1. Quá Trình Chương Trình Chạy Từ Đầu Tới Cuối?

* Ta chỉ quan tâm tới các ô nhớ Register, phần nhớ toàn cục, phần nhớ chứa Instruction, Stack, còn Heap không quan tâm
* Khi mới đầu nhập lệnh chạy chương trình trong CMD, ta Pass các tham số cho chương trình, thì các tham số này sẽ được đẩy vào Stack, đồng thời địa chỉ của Instruction tiếp theo trong tác vụ của CMD, gọi là A, cũng được đẩy vào, đồng thời EIP là Register mang giá trị là địa chỉ của Instruction sẽ được thực hiện cũng sẽ mang giá trị là Instruction đầu tiên của hàm main, hơn nữa giá trị trước đó của EBP cũng được lưu trong Stack, gọi là B
* Instruction thời điểm này

|  |  |
| --- | --- |
| Địa chỉ | Dữ liệu |
| … | … |
| K + Z + 2 | Instruction Z + 2 hàm bar |
| K + Z + 1 | Instruction trả về hàm foo |
| K + Z | Instruction Z hàm foo |
| … | … |
| K + Y + 2 | Instruction Y + 2 hàm foo |
| K + Y + 1 | Instruction Y + 1 hàm foo |
| K + Y | Instruction Y hàm main |
| … | … |
| K + X + 3 | Instruction X + 3 hàm main |
| K + X + 2 | Instruction X + 2 hàm main |
| K + X + 1 | Instruction gọi hàm foo |
| K + X | Instruction X hàm main |
| … | … |
| K + 2 | Instruction 2 hàm main |
| K + 1 | Instruction 1 hàm main |

* Stack thời điểm này

|  |  |  |
| --- | --- | --- |
| Địa chỉ | Dữ liệu | Tên |
| N + M | Tham số M hàm main | Đáy Stack |
| … | … |  |
| N + 2 | Tham số 2 hàm main |  |
| N + 1 | Tham số 1 hàm main |  |
| N | A |  |
| N – 1 | B | Đỉnh Stack |
| … | … | … |
| H | ? | Giới hạn Stack |

* Register thời điểm này

|  |  |
| --- | --- |
| Dữ liệu | Tên |
| K + 1 | EIP |
| N – 1 | ESP |
| N – 1 | EBP |
| ? | EAX |
| ? | EBX |
| … | … |

* Phần nhớ toàn cục thời điểm này

|  |  |
| --- | --- |
| Địa chỉ | Dữ liệu |
| F + G | Biến toàn cục G |
| … | … |
| F + 2 | Biến toàn cục 2 |
| F + 1 | Biến toàn cục 1 |

* EIP là Register mang giá trị là địa chỉ của Instruction đang thực hiện, tự động tăng thêm 1 khi xong 1 Instruction hoặc nhảy tới Instruction nào đấy khi gặp lệnh nhảy
* ESP là Register mang giá trị là địa chỉ của đỉnh Stack hiện tại
* EBP là Register mang giá trị là địa chỉ của ô nhớ lưu giá trị EBP cũ
* Cứ mỗi khi biến được tạo ra trong hàm main, thì nó sẽ được đẩy vào Stack, đồng thời ESP giảm 1
* Sau thời điểm này, Instruction 1 sẽ được thực hiện, sau đó đến 2, …, cho đến khi EIP mang giá trị là K + X + 1, giả sử thời điểm đó bộ nhớ có dạng sau
* Stack

|  |  |  |
| --- | --- | --- |
| Địa chỉ | Dữ liệu | Tên |
| N + M | Tham số M hàm main | Đáy Stack |
| … | … |  |
| N + 2 | Tham số 2 hàm main |  |
| N + 1 | Tham số 1 hàm main |  |
| N | A |  |
| N – 1 | B |  |
| N – 2 | Biến cục bộ 1 hàm main |  |
| N – 3 | Biến cục bộ 2 hàm main |  |
| … | … |  |
| N – L | Biến cục bộ L – 1 hàm main | Đỉnh Stack |
| … | … |  |
| H | ? | Giới hạn Stack |

* Register

|  |  |
| --- | --- |
| Dữ liệu | Tên |
| K + X + 1 | EIP |
| N – L | ESP |
| N – 1 | EBP |
| ? | EAX |
| ? | EBX |
| … | … |

* Sau thời điểm này, Instruction gọi hàm foo được thực hiện, khi sẽ tái diễn lại những gì đã xảy ra khi hàm main được chạy
* Trạng thái bộ nhớ ngay sau khi gọi hàm foo
* Stack

|  |  |  |
| --- | --- | --- |
| Địa chỉ | Dữ liệu | Tên |
| N + M | Tham số M hàm main | Đáy Stack |
| … | … |  |
| N + 2 | Tham số 2 hàm main |  |
| N + 1 | Tham số 1 hàm main |  |
| N | A |  |
| N – 1 | B |  |
| N – 2 | Biến cục bộ 1 hàm main |  |
| N – 3 | Biến cục bộ 2 hàm main |  |
| … | … |  |
| N – L | Biến cục bộ L – 1 hàm main |  |
| N – L – 1 | Tham số W hàm foo |  |
| … | … |  |
| N – L – W + 1 | Tham số 2 hàm foo |  |
| N – L – W | Tham số 1 hàm foo |  |
| N – L – W – 1 | K + X + 2 |  |
| N – L – W – 2 | N – 1 | Đỉnh Stack |
| … | … |  |
| H | ? | Giới hạn Stack |

* Register

|  |  |
| --- | --- |
| Dữ liệu | Tên |
| K + Y + 1 | EIP |
| N – L – W – 2 | ESP |
| N – L – W – 2 | EBP |
| ? | EAX |
| ? | EBX |
| … | … |

* Sau thời điểm này, Instruction Y + 1 sẽ được thực hiện, sau đó đến Y + 2, …, cho đến khi EIP mang giá trị là K + Z + 1, giả sử thời điểm đó bộ nhớ có dạng sau
* Stack

|  |  |  |
| --- | --- | --- |
| Địa chỉ | Dữ liệu | Tên |
| N + M | Tham số M hàm main | Đáy Stack |
| … | … |  |
| N + 2 | Tham số 2 hàm main |  |
| N + 1 | Tham số 1 hàm main |  |
| N | A |  |
| N – 1 | B |  |
| N – 2 | Biến cục bộ 1 hàm main |  |
| N – 3 | Biến cục bộ 2 hàm main |  |
| … | … |  |
| N – L | Biến cục bộ L – 1 hàm main |  |
| N – L – 1 | Tham số W hàm foo |  |
| … | … |  |
| N – L – W + 1 | Tham số 2 hàm foo |  |
| N – L – W | Tham số 1 hàm foo |  |
| N – L – W – 1 | K + X + 2 |  |
| N – L – W – 2 | N – 1 |  |
| N – L – W – 3 | Biến cục bộ 1 hàm foo |  |
| N – L – W – 4 | Biến cục bộ 2 hàm foo |  |
| … | … |  |
| N – L – W – Q | Biến cục bộ Q – 2 hàm foo | Đỉnh Stack |
| … | … |  |
| H | ? | Giới hạn Stack |

* Register

|  |  |
| --- | --- |
| Dữ liệu | Tên |
| K + Z + 1 | EIP |
| N – L – W – Q | ESP |
| N – L – W – 2 | EBP |
| ? | EAX |
| ? | EBX |
| … | … |

* Sau thời điểm này, lệnh trả về sẽ được thực hiện, khi đó giá trị trả về sẽ lưu ở EAX, ESP được gán = EBP, sau đó nhìn vào giá trị EBP, nó là N – L – W – 2, tại địa chỉ này có giá trị N – 1, gán N – 1 cho EBP, đồng thời ESP được + 1
* Sau đó EIP sẽ được gán = giá trị tại địa chỉ mà địa chỉ này là giá trị của ESP, sau đó trong hàm foo do cũng có lưu số tham số Pass vào là W, nên gán ESP = ESP + W + 1
* Sơ đồ bộ nhớ sau thời điểm này
* Stack

|  |  |  |
| --- | --- | --- |
| Địa chỉ | Dữ liệu | Tên |
| N + M | Tham số M hàm main | Đáy Stack |
| … | … |  |
| N + 2 | Tham số 2 hàm main |  |
| N + 1 | Tham số 1 hàm main |  |
| N | A |  |
| N – 1 | B |  |
| N – 2 | Biến cục bộ 1 hàm main |  |
| N – 3 | Biến cục bộ 2 hàm main |  |
| … | … |  |
| N – L | Biến cục bộ L – 1 hàm main | Đỉnh Stack |
| … | … |  |
| H | ? | Giới hạn Stack |

* Register

|  |  |
| --- | --- |
| Dữ liệu | Tên |
| K + X + 2 | EIP |
| N – L | ESP |
| N – 1 | EBP |
| Giá trị hàm foo trả về | EAX |
| ? | EBX |
| … | … |

* Như vậy trạng thái các bộ nhớ trở về như khi chưa gọi hàm foo
* Ta có thể lưu giá trị các Register trước khi hàm foo được gọi vào phần biến cục bộ, để đề phòng trường hợp Register bị thay đổi khi gọi hàm foo, sau khi hàm foo kết thúc thì gán lại

1. Thân Chương Trình?

.code

<Nhãn>:

<Làm Gì Đó>

end <Nhãn>

* <Nhãn> là tên bất kì, coi như hàm main

1. Comment?

* Đặt dấu chấm phẩy đằng trước để Comment tất cả những gì đằng sau nó cho đến cuối dòng
* Ví dụ

foo:;thanglondumamay

end foo

* Tương đương

foo:

end foo

1. Return?

ret

* Rút 4 Byte trên cùng của Stack nạp vào Register EIP, ví dụ Stack có 10 Byte thì sau lệnh này còn 6 Byte

1. Gọi Hàm?

call <Nhãn>

* Đẩy địa chỉ của Instruction tiếp theo sau lệnh này vào Stack, cho EIP nhảy đến <Nhãn>

1. Gọi Hàm Hoàn Chỉnh?

invoke <Tên Hàm>, <Các Tham Số>

* Ví dụ

invoke foo, 1, 2

* Hàm foo được gọi với tham số đầu vào lần lượt là 1 và 2
* Cách gọi hàm này là chuẩn đúng theo như mục 1

1. Nhãn?

* Trong hàm main, ta có thể đặt các nhãn, khi này, ta có thể dùng lệnh nhảy để nhảy đến vị trí nhãn đó và tiếp tục thực hiện đoạn Code phía dưới nhãn đó, giống nhãn trong Batch

<Nhãn>:

<Làm Gì Đó>

* Ví dụ

.code

foo:

bar:

invoke thangcho, 0

invoke thanglon, 1

bab:

invoke thangngu, 0

jmp bar

end foo

* Lệnh jmp sẽ làm cho chương trình nhảy lại về vị trí nhãn bar, do đó tạo vòng lặp vô hạn

1. PE (Portable Executable)?

* Là định dạng 1 tệp tên thực thi có thể chạy trên bất kì chỗ nào, được EXE tuân theo

Register:

1. Các Register Trong CPU?

|  |  |  |
| --- | --- | --- |
| Tên Register | Kích thước | Chức năng |
| e<Index>x | 32 Bit | Lưu giá trị  <Index> = a thì còn sẽ lưu giá trị của mọi phép nhân  <Index> = c thì là biến đếm trong các Instruction lặp |
| <Index>x | 16 Bit | Là 2 Byte LSB của e<Index>x, thay đổi nó cũng thay đổi e<Index>x và ngược lại |
| <Index>l | 8 Bit | Là Byte LSB của e<Index>x, thay đổi nó cũng thay đổi e<Index>x, <Index>x và ngược lại |
| <Index>h | 8 Bit | Là Byte thứ 2, tức là Byte kế Byte LSB của e<Index>x, thay đổi nó cũng thay đổi e<Index>x, <Index>x và ngược lại |
| edi, esi | 32 Bit | Lưu giá trị, giá trị của chúng được xem lần lượt là địa chỉ của đích và nguồn trong các String Instruction |
| di, si | 16 Bit | Lần lượt là 2 Byte LSB của edi và esi |
| esp | 32 Bit | Lưu giá trị, giá trị của nó được xem là địa chỉ của đỉnh Stack |
| ebp | 32 Bit | Lưu giá trị, giá trị của nó được xem là địa chỉ của điểm ngăn cách giữa phần tham số và biến cục bộ trong khung hàm |
|  |  |  |

* <Index> chỉ có thể là a, b, c hoặc d, ví dụ eax

Operator – Toán Tử:

1. Gán Giá Trị Cho Register?

mov <Register>, <Giá Trị>

* Ví dụ

mov ecx, 6

1. Gán Giá Trị Cho Biến Toàn Cục = Giá Trị Của Register?

mov [<Biến Toàn Cục>], <Register>

* Ví dụ

mov [foo], eax

1. Gán Giá Trị Cho Register Đầu = Giá Trị Tại Địa Chỉ Mà Register Sau Chĩa Vào?

mov <Register Đầu>, [<Register Sau>]

* Ví dụ

mov eax, [esp]

1. Gán Giá Trị Của Register Này Cho Register Kia?

mov <Register Kia>, <Register Này>

* 2 Register phải có cùng kích thước
* Ví dụ

mov eax, edx

1. Gán Giá Trị Của Biến Toàn Cục Cho Register?

mov <Register>, <Biến Toàn Cục>

* Kể từ ô Byte đầu tiên của <Biến Toàn Cục>, đi theo hướng địa chỉ tăng dần, Copy từng Byte 1 vào <Register>, từ LSB đến MSB
* Kích thước của <Register> phải = kích thước kiểu dữ liệu <Biến Toàn Cục>

1. Gán Địa Chỉ Của Biến Toàn Cục Cho Register?

lea <Register>, <Biến Toàn Cục>

* Ví dụ

lea ax, foo

* Giả sử foo có địa chỉ ABCDEF123E theo thứ tự từ MSB đến LSB, thì ax sẽ có giá trị là 123E theo thứ tự từ MSB đến LSB, nghĩa là cắt phần LSB của địa chỉ bỏ vào Register

1. Làm Phép Bitwise XOR Trên 2 Register Rồi Trả Về Giá Trị Cho Register Đầu?

xor <Register Đầu>, <Register Sau>

* Ví dụ

xor ecx, eax

1. Cộng 2 Register Rồi Gán Kết Quả Cho Register Đầu?

add <Register Đầu>, <Register Sau>

1. Lấy Register Đầu Trừ Register Sau Rồi Gán Kết Quả Cho Register Đầu?

sub <Register Đầu>, <Register Sau>

1. Giảm Giá Trị Register Đi 1?

dec <Register>

1. Tăng Giá Trị Register Thêm 1?

inc <Register>

1. Nhảy Đến Nhãn Nào Đó?

jmp <Nhãn>

1. Nhảy Đến Nhãn Nào Đó Nếu Kết Quả Của Phép Toán Trước Đó Khác 0?

jnz <Nhãn>

* Hoặc

jne <Nhãn>

* Ví dụ nếu lệnh trước là

sub eax, eax

* Thì sẽ không nhảy do eax – eax = 0

1. Lấy Register Đầu Trừ Register Sau Nhưng Không Gán Kết Quả Cho Thứ Gì Hết?

cmp <Register Đầu>, <Register Sau>

1. Nhảy Đến Nhãn Nào Đó Nếu Kết Quả Của Phép Toán Trước Đó = 0?

jz <Nhãn>

1. Nhân 1 Register Nào Đó Với Register EAX, Rồi Gán Kết Quả Cho Register EAX?

mul <Register Nào Đó>

1. Dịch Trái Phải Register?

* Quy ước giá trị trong Register từ trái sang phải là MSB tới LSB, đồng thời khi dịch, MSB nối vào LSB
* Để dịch trái

rol <Register>, <Số Bit Dịch>

* Để dịch phải

ror <Register>, <Số Bit Dịch>

* Ví dụ

rol eax, 5

1. Chuyển Số Nguyên Có Dấu 8 Bit Sang 16 Bit?

cbw

* Lệnh trên sẽ xét Register AL, nếu MSB của nó = 1 thì ghi đè AH Full 1, nếu MSB của nó = 0 thì ghi đè AH Full 0

1. Chuyển Số Nguyên Có Dấu 16 Bit Sang 32 Bit?

cwde

* Lệnh trên sẽ xét Register AX, nếu MSB của nó = 1 thì ghi đè phần còn lại của EAX Full 1, ngược lại Full 0

1. Thay Đổi Giá Trị Directional Flag?

* Để cho nó = 0

cld

* Để cho nó = 1

Std

1. Copy Giá Trị Trong Ô Byte Này Sang Ô Byte Khác?

movsb

* Lệnh trên sẽ Copy giá trị trong ô Byte mà Register ESI chĩa vào và gán nó vào giá trị trong ô Byte mà Register EDI chĩa vào, đồng thời ESI và EDI cùng + thêm 1 khi Directional Flag = 0, cùng trừ đi 1 khi Directional Flag = 1

lodsb

* Lệnh trên sẽ Copy giá trị trong ô Byte mà Register ESI chĩa vào và gán nó cho Register AL, đồng thời ESI + thêm 1 khi Directional Flag = 0, trừ đi 1 khi Directional Flag = 1

stosb

* Lệnh trên sẽ Copy Register AL vào ô Byte mà Register EDI chĩa vào, đồng thời EDI + thêm 1 khi Directional Flag = 0, trừ đi 1 khi Directional Flag = 1

1. Lặp Instruction?

rep <Instruction>

* <Instruction> sẽ được thực hiện k lần, k = giá trị của Register ECX, sau đó ECX bị Reset về 0
* <Instruction> chỉ có thể là movsb, lodsb

1. Đẩy Giá Trị Vào Stack?

* Để đẩy Register vào Stack

push <Register>

* Giả sử kích thước <Register> là k, dịch vị trí hiện tại của ESP xuống k đơn vị, từ vị trí của ESP lúc này, điền lần lượt Byte LSB đến MSB của <Register> vào Stack theo chiều tăng địa chỉ, k phải = 2 hoặc 4
* Để đẩy biến toàn cục vào Stack

push <Biến Toàn Cục>

* Tương tự cơ chế đẩy Register, từ địa chỉ của ô Byte đầu tiên, k = kích thước kiểu dữ liệu của <Biến Toàn Cục>, lấy k ô Byte, k phải = 2 hoặc 4
* Để đẩy số nguyên vào Stack

push <Số Nguyên>

* <Số Nguyên> không được vượt quá 4 Byte, tự động được chuyển thành 4 Byte
* <Số Nguyên> mà âm thì sẽ chuyển sang dạng bù 2, 4 Byte
* Cơ chế đẩy tương tự Register
* Để đẩy kí tự ASCII vào Stack

push <Chuỗi Kí Tự>

* <Chuỗi Kí Tự> không được vượt quá 4 kí tự, tự động được chèn thêm các kí tự \0 đằng sau cho đủ 4
* Kí tự bên trái hơn sẽ có địa chỉ thấp hơn

1. Cắt Phần Cao Nhất Của Stack Và Đẩy Nó Vào 1 Biến Toàn Cục Hoặc Register?

pop <Biến Toàn Cục>

* Giả sử kích thước kiểu dữ liệu của <Biến Toàn Cục> là k Byte, thì ESP sẽ đi lần lượt từ vị trí hiện tại tới vị trí + k đơn vị, đi đến đâu, nó nhả ô Byte ở đó cho ô Byte từ LSB đến MSB trong <Biến Toàn Cục>, kể từ vị trí ô Byte đầu tiên, giá trị các ô Byte trong Stack không thay đổi, chỉ có ESP thay đổi, k phải = 2 hoặc 4
* Tương tự với Register

pop <Register>

1. Viết Số Nhị Phân?

* Đặt b đằng sau
* Ví dụ

mov eax, 1001b

1. Viết Số Thập Lục Phân?

* Đặt 0 đằng trước h đằng sau
* Ví dụ

mov eax, 0FFABh

1. Gọi Hàm Interrupt?

int <Mã Hàm>

* <Mã Hàm> nằm trong khoảng từ 0 tới 255

|  |  |
| --- | --- |
| <Mã Hàm> | Chức năng |
| 3 | Tạm dừng chương trình, chuyển sang chế độ Debug nếu có |

Kernel 32:

1. Cách Import?

include C:\masm32\include\kernel32.inc

includelib C:\masm32\lib\kernel32.lib

1. Kết Thúc Chương Trình Ngay Và Trả Về Giá Trị Nào Đó?

invoke ExitProcess, <Giá Trị Trả Về>

MASM 32:

1. Cách Import?

include C:\masm32\include\masm32.inc

includelib C:\masm32\lib\masm32.lib

1. In Ra Màn Hình Console?

invoke StdOut, offset <Biến Toàn Cục>

* Đứng tại địa chỉ của <Biến Toàn Cục>, nó đi lên địa chỉ cao hơn cho đến khi gặp phải ô Byte có giá trị \0 thì dừng lại, trả về chuỗi các ô Byte nó đi qua, dưới dạng kí tự ASCII, ô Byte có địa chỉ thấp hơn thì kí tự tương ứng nằm bên trái hơn

Windows:

1. Cách Import?

include C:\masm32\include\windows.inc

1. Một Số Giá Trị Đặc Biệt?

NULL

MB\_OK

User 32:

1. Cách Import?

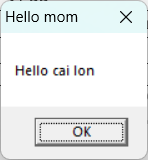
include C:\masm32\include\user32.inc

includelib C:\masm32\lib\user32.lib

1. Tạm Dừng Chương Trình Và Hiện 1 Hộp Thoại?

invoke MessageBox, NULL, <Địa Chỉ Nội Dung>, <Địa Chỉ Tiêu Đề>, MB\_OK

* <Địa Chỉ Nội Dung> và <Địa Chỉ Tiêu Đề> có cách hoạt động giống khi in ra màn hình Console
* Hộp thoại này có dạng sau



* Khi đóng hộp thoại này lại, chương trình sẽ tiếp tục

Quick Code – Mã Nhanh:

1. Hello World?

.386

.model flat, stdcall

option casemap :none

include C:\masm32\include\kernel32.inc

include C:\masm32\include\masm32.inc

includelib C:\masm32\lib\kernel32.lib

includelib C:\masm32\lib\masm32.lib

.data

message db "Hello World!", 0

.code

main:

invoke StdOut, addr message

invoke ExitProcess, 0

end main