TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

——————— * ———————

BÁO CÁO HỌC PHẦN:
LẬP TRÌNH HỆ THỐNG

DỊCH NGƯỢC MÃ NGUỒN
(REVERSE ENGINEERING)

Nhóm sinh viên	: **Lê Đình Mạnh** - 20162644
	: **Vũ Tiến Hùng** – 20161996
	: **Trần Hữu Thúy** - 20163983
Lớp	: **111608**
Giáo viên hướng dẫn	: **TS. Đỗ Quốc Huy**

Hà Nội, tháng 12 năm 2019

MỤC LỤC

Lời mở đầu:

Dịch ngược mã nguồn là một trong những lĩnh vực quan trọng của công nghệ thông tin. Nó đòi hỏi rất nhiều kiến thức về hệ thống phần cứng, quy trình phát triển phần mềm, … Trong khuôn khổ của môn học và kiến thức hạn chế, chúng em xin trình bày những khái niệm cơ bản và những yếu tố cần thiết cho dịch ngược mã nguồn.

Em xin chân thành cảm ơn sự nhiệt tình giúp đỡ của thầy TS. Đỗ Quốc Huy đã giúp chúng em hoàn thiện đề tài này!

Hà Nội, tháng 12 năm 2019

1. Dịch ngược mã nguồn là gì?

- Reverse engineering: là quá trình tìm ra nguyên lý kỹ thuật của một phần mềm hay thiết bị phần cứng qua việc phân tích cấu trúc, chức năng và hoạt động của nó. Trong quá trình này, người ta thường phải phân tích chi tiết hoạt động theo từng phần của đối tượng để hiểu được thiết kế của nó.
- Dịch ngược mã nguồn: là quá trình phân tích nhằm nắm bắt được chương trình mà không có source code hoặc tài liệu mô tả, cố gắng khôi phục các chi tiết liên quan đến thiết kế và triển khai nó. Kỹ thuật dịch ngược mã nguồn đòi hỏi sự kết hợp các kỹ năng và sự hiểu biết thấu đáo về hệ thống máy tính, quy trình phát triển phần mềm, nó tích hợp một số kỹ nghệ về phá mã, giải câu đó, lập trình và phần tích logic.

2. Ứng dụng của dịch ngược mã nguồn

Đảo ngược mã nguồn được ứng dụng vào rất nhiều lĩnh vực khác nhau trong công nghệ thông tin, phổ biến nhất là ứng dụng liên quan đến bảo mật và phát triển các phần mềm liên quan.

2.1. Bảo mật

Với một số người, việc liên hệ giữa bảo mật và dịch ngược mã nguồn có thể không rõ ràng, nhưng trong thực tế, dịch ngược mã nguồn có liên quan đến nhiều khía cạnh khác nhau của bảo mật máy tính.

- Dịch ngược mã nguồn được sử dụng trong nghiên cứu mã hóa, các nhà nghiên cứu sản phẩm mã hóa và đánh mức độ bảo mật mà nó cung cấp. Nó giúp tìm ra thuật toán mã hóa để giải được mã khóa hay tìm ra được lỗ hổng trong thuật toán để trích xuất khóa hoặc bản rõ.
- Dịch ngược mã nguồn cũng được sử dụng nhiều khi nghiên cứu phần mềm độc hại, phía hacker ứng dụng nó để tìm ra lỗ hổng phần mềm hoặc hệ điều hành để tạo ra virus hay trojan gây hại. Về phía các chuyên gia bảo mật, họ ứng dụng để phát hiện lỗ hổng, mổ sẻ, phân tích mã độc nhằm tránh thiệt hại và tìm cách khắc phục.
- Trong bản quyền kỹ thuật số: để tránh việc sao chép bản quyền, các nhà phát hành thường nhúng các đoạn mã bổ sung nhằm cố gắng hoặc hạn chế người dùng sao chép được chương trình. Tuy nhiên, nó hoàn toàn có thể bị kỹ thuật dịch ngược mã nguồn phân tích và bẻ khóa (crack)

2.2. Trong phát triển phần mềm

Đảo ngược mã nguồn có thể giúp chúng ta khôi phục được phần nào source code của chương trình tuy nhiên trong thực tế không ứng dụng nó để tạo ra các sản phẩm cạnh tranh vì các phần mềm thường quá phức tạp cho kỹ thuật dịch ngược.

- Sử dụng dịch ngược để khám phá cách sử dụng phần mềm không có tài liệu hoặc chỉ có một phần tài liệu mô tả.
- Trích xuất các thông tin có giá trị từ sản phẩm của đối thủ cạnh tranh với mục đích cải thiện công nghệ riêng.
- Đánh giá chất lượng và tính năng mạnh mẽ của phần mềm.

3. Cơ sở thực hiện dịch ngược mã nguồn

Thông thường, các chương trình máy tính được dịch về các file thực thi (executable file – định dạng file đặc biệt chứa các thông tin cần thiết để hệ điều hành có thể đưa vào trong bộ nhớ và thực hiện) và khi thực hiện dịch ngược mã nguồn, kỹ thuật viên chỉ nắm được file thực thi đó để tiến hành dịch ngược. Có một số kỹ thuật và yêu cầu cần thiết cho việc dịch ngược, tuy nhiên trước hết ta sẽ đi tìm hiểu về cấu trúc và định dạng file thực thi để hiểu sâu hơn về thứ mà chúng ta đang xử lý.

3.1. Định dạng file thực thi

Định dạng file thực thi phụ thuộc và kiến trúc bộ xử lý và hệ điều hành đang sử dụng. Với hệ điều hành Windows sử đụng định dạng Portable Executable (PE), hệ điều hành Linux sử dụng Executable and Linkable Format (ELF). Trong khuôn khổ môn học em chỉ xin trình bày về định dạng PE trong Windows.

Định dạng file PE được tổ chức theo các dòng dữ liệu nối tiếp nhau. Nó bắt đầu với 1 MS-DOS header và chữ ký PE file. Ngay sau đó là PE file header và optional header và đến header các section và nội dung các section. Kết thúc file là một số thông tin như: thông tin tái định vị, bảng thông tin ký hiệu, thông tin về số dòng và dữ liệu bảng chuỗi.

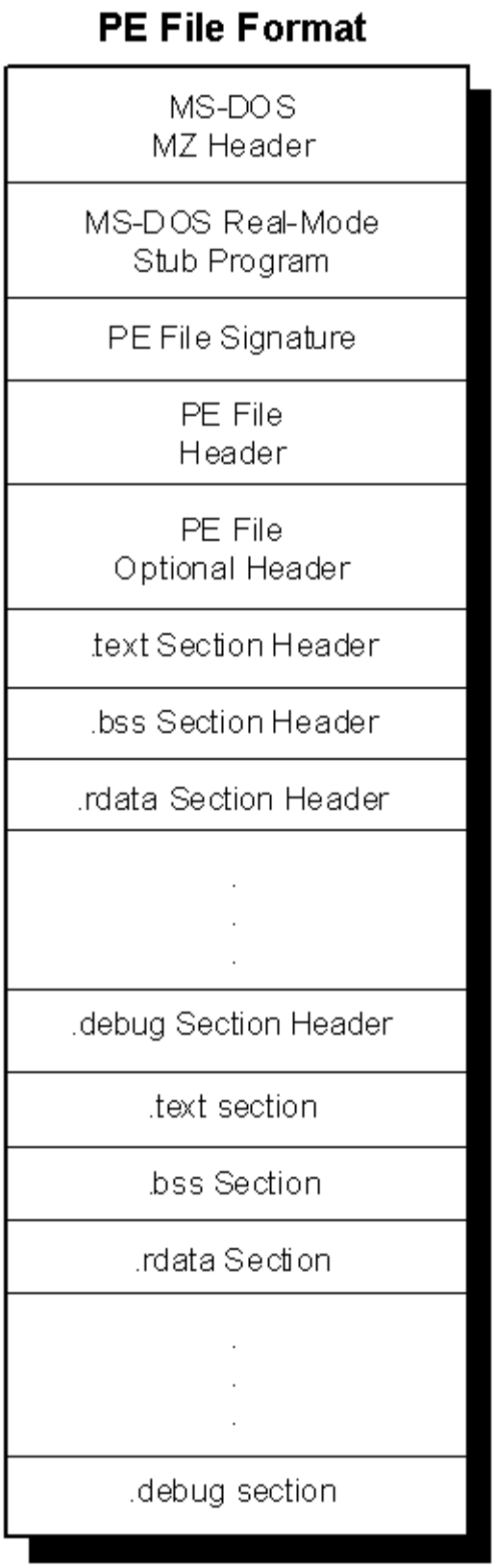


Figure 1: Cấu trúc file Portable Executable

- DOS MZ Header: Thành phần này tổ chức xem file của bạn có đúng là file định dạng PE hay không.Tất cả file đúng định dạng sẽ được bắt đầu với MZ.
- +MS-Dos Stub:Nếu chương trình không thể chạy trên windows, một chuỗi cảnh báo sẽ được hiển thị ra "This program cannot be run in DOS mode". Tại vị trí 0x3c, nó có file offset để đến PE signature.
- PE Header: được cấu trúc với tên IMAGE_NT_HEADER, bao gồm 3 thành phần chính
    - Signature: bắt đầu PE Header, nằm ngay sau MS-Dos Stub, 4 bytes
    - COFF File Header (Object and Image): được chỉ định với cấu trúc IMAGE_FILE_HEADER:

      IMAGE_FILE_HEADER struct {

      Machine: 2 bytes

      NumberOfSections: 2 bytes

      TimeDateStamp: 4 bytes

      PointerToSymbolTable: 4 bytes

      NumberOfSymbols: 4 bytes

      SizeOfOptionalHeader: 2 bytes

      Characteristics: 2 bytes

      }

      ~Machine: giá trị hằng số nguyên, chỉ định kiến trúc CPU. VD IMAGE_FILE_MACHINE_I386 (0x14c), Intel 386 or later processors and compatible processors

~NumberOfSections: số sections trong files, từ đó cho biết kích thước bằng các sections.

~Characteristics: tập các cờ chỉ định tập thuộc tính của file (exe, dll, ...)

- Optinal Header: 224 bytes, chứa thông tin sơ đồ logic của PE file.

+8 trường thuộc tính đầu tiên là các trường chuẩn được định định nghĩa mỗi khi thực thi COEF.

+ Magic: 2bytes, số nguyên không dấu,định nghĩa trạng thái của image file.VD 0x10B, định nghĩa file thực thi bình thường.

+MajorLinkerVersion: 1 byte

+MinorLinkerVersion: 1 byte

+SizeOfCode: 4bytes, kích thước của tất cả (text) section.

+SizeOfInitializedData: 4bytes, kích thước của các initialized data section

+SizeOfUninitializedData: 4bytes, kích thước của các uninitialized data section

+AddressOfEntryPoint: 4 bytes, liên quan đến đến image base khi file được nạp vào bộ nhớ.

+BaseOfCode: 4 bytes, địa chỉ này liên quan đến image base của beginning-of-code section khi nó được nạp trong bộ nhớ.

- Section Table: Số lượng section được lưu trữ ở thuộc tính NumberOfSections, mỗi hàng được tổ chức theo cấu trúc

IMAGE_SECTION_HEADER, đinh dạng thông tin của mỗi section tương ứng. Dưới đây là một vài section đặc biệt:

- SECTION.text: chứa mã code thực thi.
- SECTION.data: chứa dữ liệu đã được khởi tạo của chương trình.
- SECTION.bss: chứa dữ liệu chưa được khởi tạo của chương trình.
- SECTION.idata: chứa các data có thể đọc viết, API, …

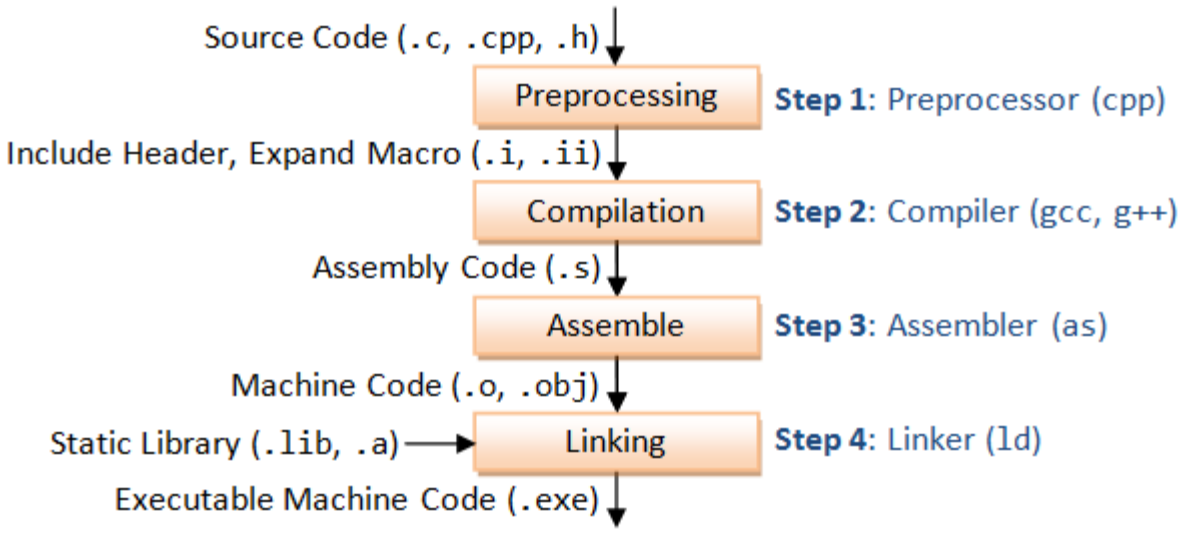
Khi thực hiện dịch ngược mã nguồn ta thường chỉ cần quan tâm đến chỉ định của kiến trúc CPU và các trường thông tin trong section.

Lưu ý khi xử lý các file PE là các địa chỉ trong file PE là địa chỉ tương đối so với đầu file mỗi lần nó sẽ được tải với 1 địa chỉ ảo khác nhau. Mỗi khi khởi tạo 1 module thì địa chỉ cơ sở của nó sẽ được khởi tạo, khi thực thi thì trình loader sẽ tiến hành thay hết những địa chỉ tương đối sang địa chỉ tuyệt đối.

3.2. Quy trình biên dịch và thực thi chương trình

3.2.1. Quá trình biên dịch (compile)

Là quá trình chuyển đổi từ ngôn ngữ bậc cao (NNBC) (C, C++, …) sang ngôn ngữ máy (mã nhị phân) mà máy tính đọc được và thực thi. Quá trình biên dịch được chia ra làm 4 giai đoạn chính: Giai đoạn tiền xử lý, giai đoạn dịch ngôn ngữ bậc cao sang mã assembly, giai đoạn dịch mã assembly sang mã máy, giai đoạn liên kết.



- Giai đoạn tiền xử lý:
    - Nhận mã nguồn, kiểm tra cú pháp.
    - Xóa bỏ tất cả chú thích, comment của chương trình
    - Xử lý các chỉ thị tiền xử lý (Ví dụ #include cho phép ghép thêm mã chương trình của một tệp tiêu đề và mã nguồn cần dịch, các hằng số được định nghĩa bằng #define sẽ được thay thế bằng các giá trị cụ thể tại mỗi nơi sử dụng chương trình).
- Dịch ngôn ngữ bậc cao sang assembly:
    - Phân tích cú pháp của mã nguồn, chuyển chúng sang dạng mã assembly.
    - Giai đoạn này còn tích hợp thêm trình tối ưu hóa nhằm tối ưu hóa hiệu năng và bộ nhớ sử dụng của chương trình. Cấu trúc chương trình trên mã assembly có thể không giống như mã nguồn ngôn ngữ bậc cao (dù chức năng thực hiện tương tự). Ví dụ các vòng lặp có sử dụng lệnh nhảy và điều kiện có thể thay (hoặc thay thế một phần) bằng các lệnh tuần tự giống nhau thực hiện nhiều lần. Mã assembly thường sẽ rất khó đọc và khó có thể khôi phục lại được.

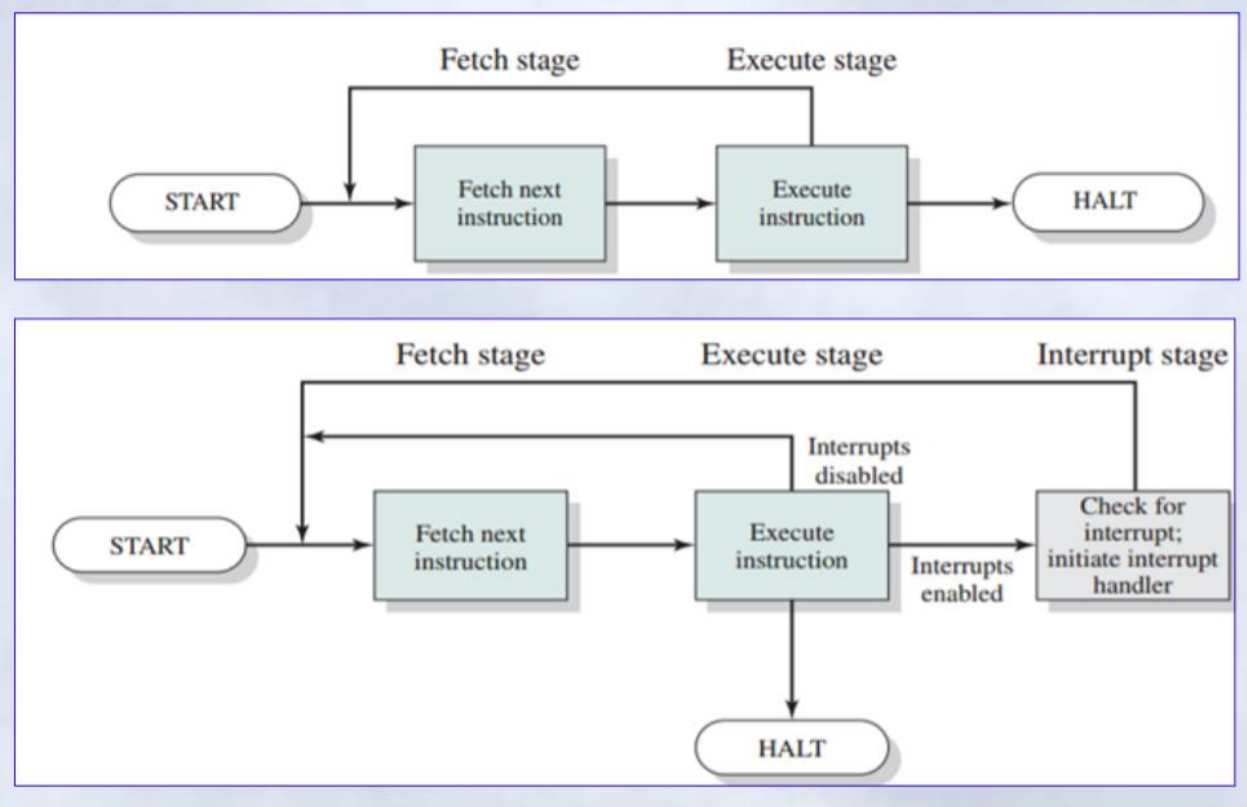
- Dịch mã assembly sang mã nhị phân:
    - Các lệnh assembly có mỗi quan hệ 1-1 với các lệnh mã nhị phân
    - Tùy thuộc vào kiến trúc bộ xử lý, hệ thống sẽ chuyển đổi mã assembly thành các mã nhị phân tương ứng.
- Giai đoạn Linker
    - Trong giai đoạn này mã máy của một chương trình dịch từ nhiều nguồn (file .c hoặc file thư viện liên kết tĩnh .lib) được liên kết với nhau tạo thành chương trình đích duy nhất.
    - Mã máy của các hàm thư viện gọi trong chương trình cũng được đưa vào chương trình cuối trong giai đoạn này.
    - Chính vì vậy các lỗi liên quan đến việc gọi hàm hay sử dụng biến tổng thể mà không tồn tại sẽ bị phát hiện.

Kết thúc quá trình này tất cả các đối tượng được liên kết lại với nhau thành một file executable thống nhất.

3.2.2. Quá trình thực thi

Loader sẽ được sử dụng để thực thi một file thực thi trong ngữ cảnh tiến trình.

- Loader cung cấp bộ nhớ cho các phần của chương trình (Text, Data, bss, …)
- Nạp vào bộ nhớ thư viện dùng chung (nếu thư viện đó chưa được nạp)
- Sau khi hoàn tất, loader nhảy tới hàm _start(). Hàm này gọi tới hàm init() của các thư viện, sau đó gọi tới hàm main() để thực hiện chương trình.



3.3. Dịch ngược mã nguồn

Mã nguồn (ở dạng ngôn ngữ bậc cao), thường được dịch về mã assembly rồi từ mã assembly dịch sang mã máy, ta cần đảo ngược cả 2 quá trình này.

Để dịch ngược được mã nhị phân sang mã assembly ta thường sử dụng trình Disassembler quá trình này không quá phức tạp và có thể thực hiện được hiện nay có rất nhiều tool hỗ trợ.

Từ mã assembly dịch ngược sang ngôn ngữ bậc cao thì thường rất khó khăn vì như đã trình bày tùy thuộc vào trình biên dịch, kiến trúc thực thi, hệ điều hành khác nhau ta sẽ thu được mã assembly khác nhau. Những mã assembly này đã được tối ưu hóa sẽ rất khó đọc và không còn giữ được cấu trúc ban đầu của chương trình, ngoài ra tên biến, hằng số cũng bị ẩn đi chỉ còn lại các giá trị địa chỉ, … Tuy nhiên,

với một số ngôn ngữ không biên dịch ra mã máy mà biên dịch kiểu bytecode thì có thể thực hiện dịch ngược được gần như toàn bộ chương trình.

Các tool hỗ trợ là điều không thể thiếu trong quá trình dịch ngược, phân tích mã nguồn. Hiện nay có nhiều chương trình miễn phí hoặc trả phí: OllyDBG, IDA, … Tuy nhiên thay vì trình bày cách sử dụng những tool này, ta tìm hiểu về cơ sở để những tool này có thể thực hiện việc dịch ngược mã nguồn.

Trong cấu trúc file PE có section .text chứa toàn bộ mã code thực thi của chương trình. Bên dưới là nội dung phần .text section chương trình Hello world viết bằng C. File PE được đọc bằng tool PE Viewer.

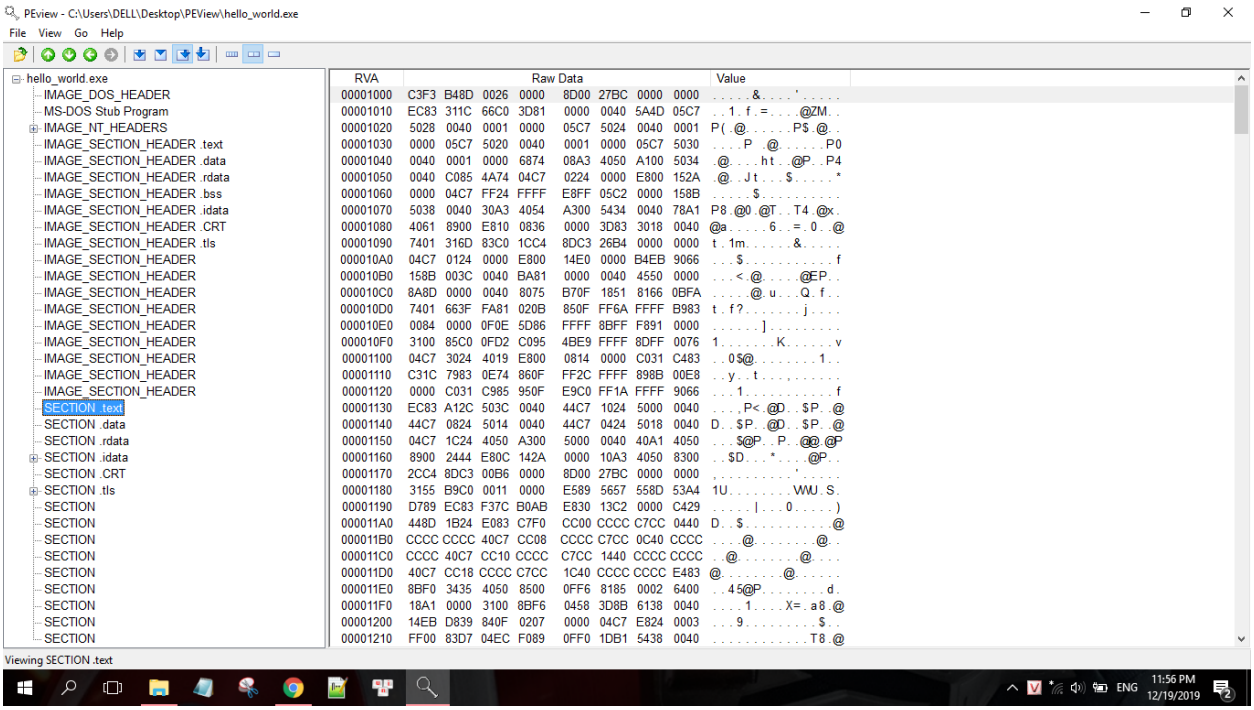


Figure 2: Mã máy chương trình Hello World

Mỗi lệnh assembly được ánh xạ 1-1 với mã nhị phân. Mỗi mã assembly sẽ được xác định bằng 1 opcode, tiếp đó là các toán hạng hoặc thành phần tương ứng của lệnh đó. Mỗi lệnh assembly trong kiến trúc X86 có thể có độ dài từ 1-15 bytes khi dịch sang mã nhị phân. Vì vậy ta cần nắm kỹ được kiến trúc tập lệnh: mã opcode, độ dài lệnh, số toán hạng, …

Hai cách cơ bản để dịch ngược mã nhị phân sang mã assembly:

- Quét tuyến tính: Quét từ đầu chương trình cho đến khi nhận được 1 mã opcode nào đó, nếu đúng mã opcode và định dạng lệnh thì ta tiếp tục quét.
- Traversal đệ quy: Quét N-bytes cho đến khi nhận được lệnh nhảy, lưu trữ vị trí và thực hiện lệnh nhảy, tiến hành xác định các lệnh sau lệnh nhảy dựa vào mã opcode cho đến khi nhận được 1 mã opcode không hợp lệ, ta quay lại vị trí đã lưu trữ trước đó và xác định tiếp các lệnh tiếp theo cho đến hết chương trình.

Vấn đề đặt ra là:

- Ta cần xác định được chương trình bắt đầu từ đâu.
- Cấu trúc của chương trình.

Do đó ta thường sử dụng thêm trình debugger, trong quá trình thực hiện dịch ngược ta cần từng bước xác định được cấu trúc chương trình, dòng chảy dữ liệu, loại dữ liệu, …

Khi thực hiện dịch ngược mã nguồn, không nên cố gắng chuyển toàn bộ mã nhị phân sang mã assembly, ta cần xác định được những hàm quan tâm, những chức năng chính của chương trình và tập trung xử lý đoạn code tương ứng đó.

Quá trình dịch ngược từ mã assembly sẽ phức tạp và đòi hỏi nhiều công sức hơn nữa.

Trên đây là những kiến thức cơ bản để hiểu thêm về dịch ngược mã nguồn, nhìn có vẻ đơn giản nhưng khi thực hiện thì vô cùng khó khăn kể cả với các công cụ hỗ trợ.

Phần demo em xin trình bày khi thuyết trình

4. Tài liệu tham khảo:

- Cuốn sách Reversing Secret of Reverse Engineering – Eldad Eilam
- Bài giảng Lập trình hệ thống – TS. Đỗ Quốc Huy
- https://docs.microsoft.com/en-us/windows/win32/debug/pe-format
- https://en.wikipedia.org/wiki/Portable_Executable
- https://blog.kowalczyk.info/articles/pefileformat.html
- https://tapit.vn/qua-trinh-bien-dich-mot-chuong-trinh-cc/
- https://en.m.wikibooks.org/wiki/X86_Assembly/Machine_Language_Conversion?fbclid=IwAR1gX6qQZgW5AJvUlQqcuT7oKqv_2Gbd-0vnrz7bcbKvHkKe5XhxSMi2R9g