

1

Lab

**PHỤC VỤ MỤC ĐÍCH GIÁO DỤC**  
FOR EDUCATIONAL PURPOSE ONLY

# ÔN TẬP NGÔN NGỮ ASSEMBLY & CHÈN MÃ VÀO TẬP TIN PE

**Thực hành môn Cơ chế hoạt động của mã độc**

Tháng 3/2025

**Lưu hành nội bộ**

*<Nghiêm cấm đăng tải trên internet dưới mọi hình thức>*

# ÔN TẬP NGÔN NGỮ ASSEMBLY

## A. TỔNG QUAN

### 1. Mục tiêu

- Tìm hiểu ngôn ngữ assembly (hợp ngữ)
- Thực hiện một số chương trình mẫu trên IDE online

### 2. Kiến thức nền tảng

Hợp ngữ là một ngôn ngữ lập trình cấp thấp đối với một máy tính hoặc thiết bị lập trình khác đặc trưng cho một kiến trúc máy tính cụ thể trái ngược với hầu hết các ngôn ngữ lập trình cấp cao, nói chung là nhiều hệ thống trên di động. Hợp ngữ được chuyển thành mã máy thực thi một số trình biên dịch như NASM, MASM...

Một ví dụ về hợp ngữ chương trình xuất ra chữ "Hello, world":

```
section    .text

    global _start    ;must be declared for linker (ld)
_start:                ;tells linker entry point

    mov     edx,len    ;message length
    mov     ecx,msg    ;message to write
    mov     ebx,1      ;file descriptor (stdout)
    mov     eax,4      ;system call number (sys_write)
    int     0x80       ;call kernel

    mov     eax,1      ;system call number (sys_exit)
    int     0x80       ;call kernel

section    .data
msg db 'Hello, world!', 0xa ;our dear string
len equ $ - msg    ;length of our dear string
```

## Hợp ngữ là gì?

Mỗi máy tính cá nhân có một bộ vi xử lý để quản lý hoạt động số học, logic, và điều khiển của máy tính.

Mỗi họ của bộ vi xử lý đã thiết lập riêng của mình về hướng dẫn xử lý các hoạt động khác nhau chẳng hạn như nhận đầu vào từ bàn phím, hiển thị thông tin trên màn hình và thực hiện các công việc khác nhau. Những bộ lệnh(instruction) được gọi là lệnh ngôn ngữ máy”.

Một bộ xử lý hiểu chỉ lệnh ngôn ngữ máy, đó là những chuỗi của 1 và 0. Tuy nhiên, ngôn ngữ máy là quá mơ hồ và phức tạp để sử dụng trong phát triển phần mềm. Vì vậy, các ngôn ngữ lắp ráp ở mức độ thấp được thiết kế cho một gia đình cụ thể của bộ vi xử lý đại diện cho hướng dẫn khác nhau trong mã biểu tượng và một hình thức dễ hiểu hơn.

## Ưu điểm của hợp ngữ

Có một sự hiểu biết về ngôn ngữ lắp ráp làm cho người ta nhận thức được:

- Làm thế nào các chương trình giao diện với các hệ điều hành, bộ xử lý, và BIOS;
- Làm thế nào dữ liệu được đại diện trong bộ nhớ và các thiết bị bên ngoài khác;
- Làm thế nào các bộ vi xử lý truy cập và thực hiện các hướng dẫn;
- Làm thế nào hướng dẫn truy cập và xử lý dữ liệu;
- Làm thế nào một chương trình truy cập các thiết bị bên ngoài.

Những lợi ích khác của việc sử dụng ngôn ngữ lắp ráp là:

Nó đòi hỏi ít bộ nhớ và thời gian thực hiện;

- Nó cho phép phần cứng cụ thể công việc phức tạp một cách dễ dàng hơn;
- Nó phù hợp cho công việc thời gian quan trọng;
- Nó phù hợp nhất để viết các thủ tục dịch vụ ngắt và chương trình thường trú bộ nhớ khác.

## Các tính năng cơ bản của phần cứng máy

Các phần cứng nội bộ chính của một máy tính bao gồm bộ vi xử lý, bộ nhớ, và đăng ký. Đăng ký là thành phần xử lý mà giữ dữ liệu và địa chỉ. Để thực hiện một chương trình, các bản sao hệ thống từ các thiết bị bên ngoài vào bộ nhớ trong. Các bộ vi xử lý thực hiện các hướng dẫn của chương trình.

Các đơn vị cơ bản của lưu trữ máy tính là một chút; nó có thể là ON (1) hoặc OFF (0). Một nhóm chín bit liên quan làm cho một byte, trong đó tám bit được sử dụng cho dữ liệu và là người cuối cùng được sử dụng cho tính chẵn lẻ. Theo quy luật chẵn lẻ, số lượng bit được ON (1) trong mỗi byte nên luôn luôn là số lẻ.

Vì vậy, các bit chẵn lẻ được sử dụng để làm cho số bit trong một byte lẻ. Nếu tính chẵn lẻ là chẵn, hệ thống giả định rằng đã có một lỗi chẵn lẻ (dù hiếm), mà có thể đã được gây ra do lỗi phần cứng hoặc xáo trộn điện.

### Các bộ xử lý hỗ trợ các kích thước dữ liệu sau:

- Word: một mục dữ liệu 2-byte
- Doubleword: a 4-byte (32 bit) mục dữ liệu
- Quadword: một byte 8 (64 bit) mục dữ liệu
- Đoạn: 16-byte (128 bit) khu vực
- Kilobyte: 1024 bytes
- Megabyte: 1.048.576 byte

## B. CHUẨN BỊ MÔI TRƯỜNG

*\*Lựa chọn một trong hai cách sau*

### 1. Chạy code assembly trên IDE online

1. Truy cập [http://www.tutorialspoint.com/compile\\_assembly\\_online.php](http://www.tutorialspoint.com/compile_assembly_online.php)
2. Nhấn vào Excute để chạy tập tin thực thi:

```

1 section .text
2 global _start           ;must be declared for using gcc
3 _start:                 ;tell linker entry point
4 mov edx, len            ;message length
5 mov ecx, msg            ;message to write
6 mov ebx, 1              ;file descriptor (stdout)
7 mov eax, 4              ;system call number (sys_write)
8 int 0x80                ;call kernel
9 mov eax, 1              ;system call number (sys_exit)
10 int 0x80               ;call kernel
11
12 section .data
13
14 msg db 'Hello, world!',0xa ;our dear string
15 len equ $ - msg         ;length of our dear string

```

```

$ nasm -f elf *.asm; ld -m elf_i386 -s -o demo *.o

$demo

Hello, world!

```

## 2. Chạy code assembly trên Linux NASM

### 1. Tạo tập tin nguồn *test.asm*

```

khoanh@khoanh-inseclab:~$ cat test.asm
section .text
    global _start           ;must be declared for using gcc
_start:
    mov     edx, len        ;message length
    mov     ecx, msg        ;message to write
    mov     ebx, 1          ;file descriptor (stdout)
    mov     eax, 4          ;system call number (sys_write)
    int     0x80            ;call kernel
    mov     eax, 1          ;system call number (sys_exit)
    int     0x80            ;call kernel

section .data

msg     db     'Hello, world!',0xa    ;our dear string
len     equ     $ - msg                ;length of our dear string

```

### 2. Chạy lệnh **nasm -f elf test.asm** để biên dịch code ra tập tin *test.o*

### 3. Tạo tập tin thực thi **ld -m elf\_i386 test.o -o test**

### 4. Chạy tập tin thực thi **./test**

## C. THỰC HÀNH

## 1. Cấu trúc cơ bản của một chương trình

```
1 section .text
2     global _start           ;must be declared for using gcc
3     _start:                 ;tell linker entry point
4     mov edx, len            ;message length
5     mov ecx, msg            ;message to write
6     mov ebx, 1              ;file descriptor (stdout)
7     mov eax, 4              ;system call number (sys_write)
8     int 0x80                ;call kernel
9     mov eax, 1              ;system call number (sys_exit)
10    int 0x80                 ;call kernel
11
12 section .data
13
14 msg db 'Cau truc co ban cua mot chuong trinh',0xa ;our dear string
15 len equ $ - msg           ;length of our dear string
16
```

1. Phần **.text** là phần chính chạy chương trình.
2. Phần **.data** là phần khai báo biến và hằng của chương trình ở đây chúng ta khai báo hai biến `msg` và `len`.
3. Gõ và thực thi chương trình trên IDE Online hoặc NASM và quan sát kết quả trả về.

```
khoanh@khoanh-inseclab:~$ nasm -f elf *.asm; ld -m elf_i386 -s -o demo *.o
khoanh@khoanh-inseclab:~$ ./demo
Cau truc co ban cua mot chuong trinh
khoanh@khoanh-inseclab:~$
```

## 2. Địa chỉ thanh ghi, khai báo biến và gọi hàm hệ thống

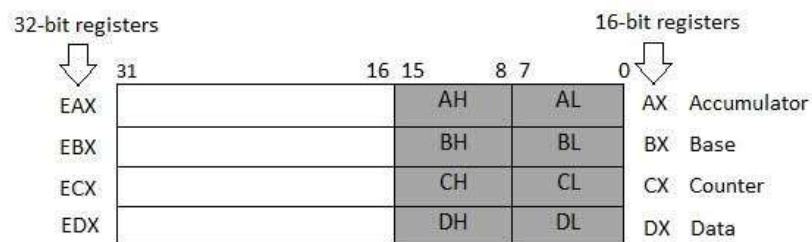
1. Thực thi chương trình sau và quan sát kết quả.

```

1 section .text
2     global _start      ;must be declared for linker (gcc)
3
4 _start:
5     mov     edx,len     ;message length
6     mov     ecx,msg     ;message to write
7     mov     ebx,1       ;file descriptor (stdout)
8     mov     eax,4       ;system call number (sys_write)
9     int     0x80        ;call kernel
10
11     mov     edx,5       ;message length
12     mov     ecx,s2      ;message to write
13     mov     ebx,1       ;file descriptor (stdout)
14     mov     eax,4       ;system call number (sys_write)
15     int     0x80        ;call kernel
16
17     mov     eax,1       ;system call number (sys_exit)
18     int     0x80        ;call kernel
19
20 section .data
21 msg db 'Displaying 5 stars', 0xa ;a message
22 len equ $ - msg         ;length of message
23 s2 times 5 db '*'

```

2. Trong ví dụ sử dụng các vùng địa chỉ trên thanh ghi như sau: **edx**, **ecx**, **ebx**, **eax** để lưu các giá trị sau đó gọi hàm hệ thống tương ứng.



3. Các hàm hệ thống được gọi: **sys\_write(4)**, **sys\_exit(1)**.

| %eax | Name      | %ebx           | %ecx         | %edx   | %esx | %edi |
|------|-----------|----------------|--------------|--------|------|------|
| 1    | sys_exit  | int            | -            | -      | -    | -    |
| 2    | sys_fork  | struct pt_regs | -            | -      | -    | -    |
| 3    | sys_read  | unsigned int   | char *       | size_t | -    | -    |
| 4    | sys_write | unsigned int   | const char * | size_t | -    | -    |
| 5    | sys_open  | const char *   | int          | int    | -    | -    |
| 6    | sys_close | unsigned int   | -            | -      | -    | -    |

#### 4. Các kiểu biến sử dụng trong assembly.

| Directive | Purpose           | Storage Space      |
|-----------|-------------------|--------------------|
| DB        | Define Byte       | allocates 1 byte   |
| DW        | Define Word       | allocates 2 bytes  |
| DD        | Define Doubleword | allocates 4 bytes  |
| DQ        | Define Quadword   | allocates 8 bytes  |
| DT        | Define Ten Bytes  | allocates 10 bytes |

Quan sát dòng **code 22**, ta có dòng khai báo **equ** đây là cách **khai báo hằng** trong assembly. Dấu \$ trong trường hợp này là vùng địa chỉ(byte) ngay sau vùng địa chỉ khai báo biên **msg**.

### 3. Instruction

#### 1. Hiện thực chương trình và quan sát kết quả.

```

SYS_EXIT equ 1
SYS_READ equ 3
SYS_WRITE equ 4
STDIN equ 0
STDOUT equ 1

segment .data

msg1 db "Enter a digit ", 0xA,0xD
len1 equ $- msg1

msg2 db "Please enter a second digit", 0xA,0xD
len2 equ $- msg2

msg3 db "The sum is: "
len3 equ $- msg3

```



```
msg4 db 0xA
len4 equ $- msg4

segment .bss

num1 resb 2
num2 resb 2
res resb 1

section .text
global _start ;must be declared for using gcc

_start: ;tell linker entry point
mov eax, SYS_WRITE
mov ebx, STDOUT
mov ecx, msg1
mov edx, len1
int 0x80

mov eax, SYS_READ
mov ebx, STDIN
mov ecx, num1
mov edx, 2
int 0x80

mov eax, SYS_WRITE
mov ebx, STDOUT
mov ecx, msg2
mov edx, len2
int 0x80

mov eax, SYS_READ
mov ebx, STDIN
mov ecx, num2
mov edx, 2
int 0x80

mov eax, SYS_WRITE
mov ebx, STDOUT
mov ecx, msg3
mov edx, len3
int 0x80

; moving the first number to eax register and second number to ebx
; and subtracting ascii '0' to convert it into a decimal number

mov eax, [num1]
sub eax, '0'

mov ebx, [num2]
sub ebx, '0'
```

```
; add eax and ebx
add eax, ebx
; add '0' to to convert the sum from decimal to ASCII
add eax, '0'

; storing the sum in memory location res
mov [res], eax

; print the sum
mov eax, SYS_WRITE
mov ebx, STDOUT
mov ecx, res
mov edx, 1
int 0x80

; new line
mov eax, SYS_WRITE
mov ebx, STDOUT
mov ecx, msg4
mov edx, len4
int 0x80

exit:

mov eax, SYS_EXIT
xor ebx, ebx
int 0x80
```

2. Chương trình sử dụng lệnh (instruction) add để cộng hai giá trị ở vùng địa chỉ: **eax** và **ebx** lại, sau đó chuyển đổi từ số(decimal) sang chuỗi ASCII bằng cách cộng thêm ký tự '0'

```
; add eax and ebx
add eax, ebx
; add '0' to to convert the sum from decimal to ASCII
add eax, '0'
```

Ngoài ra còn có một số lệnh(instruction) như sau:

- **SUB**: trừ hai vùng địa chỉ
- **MUL**: nhân hai vùng địa chỉ
- **IMUL**: chia hai vùng địa chỉ
- **AND**: phép logic AND giữa 2 vùng địa chỉ
- **OR**: Phép logic OR giữa 2 vùng địa chỉ
- **XOR**: Phép logic XOR giữa 2 vùng địa chỉ

- **TEST:** Phép AND có biến nhớ(flag)
  - **NOT:** Phép phủ định
3. **MOV:** Phép gán dữ liệu từ một vùng này sang một vùng khác hoặc giá trị vào một vùng địa chỉ.

```
MOV register, register
MOV register, immediate
MOV memory, immediate
MOV register, memory
MOV memory, register
```

#### 4. Điều kiện rẽ nhánh (Condition)

1. Hiện thực chương trình sau và quan sát kết quả.

```
section .text
global _start ;must be declared for using gcc

_start: ;tell linker entry point
    mov ecx, [num1]
    cmp ecx, [num2]
    jg check_third_num
    mov ecx, [num2]

check_third_num:

    cmp ecx, [num3]
    jg _exit
    mov ecx, [num3]

_exit:

    mov [largest], ecx
    mov ecx, msg
    mov edx, len
    mov ebx, 1 ;file descriptor (stdout)
    mov eax, 4 ;system call number (sys_write)
    int 0x80 ;call kernel

    mov ecx, largest
    mov edx, 2
    mov ebx, 1 ;file descriptor (stdout)
    mov eax, 4 ;system call number (sys_write)
    int 0x80 ;call kernel

    mov eax, 1
    int 80h
```

```

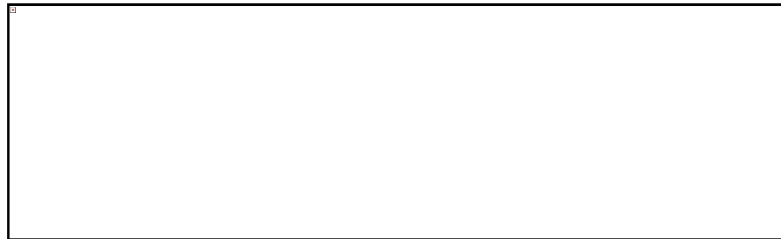
section .data

msg db "The largest digit is: ", 0xA,0xD
len equ $- msg
num1 dd '7'
num2 dd '2'
num3 dd '1'

segment .bss
largest resb 2

```

Kết quả:



2. Chương trình trên tìm ra giá trị lớn nhất trong 3 số truyền vào (num1, num2, num3). Trong chương trình này sử dụng lệnh *jump* có điều kiện kết hợp với lệnh *cmp*:

```
jg check_third_num
```

- **jg**(Jump Greater) : lệnh jump được thực thi khi giá trị so sánh của lệnh *cmp* lớn hơn.
  - **check\_third\_num**: là một nhãn địa chỉ để lệnh *jump* hướng tới thực thi.
3. Ban đầu chương trình thực hiện so sánh hai số num1 và num2.

```

mov ecx, [num1]
cmp ecx, [num2]
jg check_third_num
mov ecx, [num2]

```

4. Sau đó **jg** được thực thi để so sánh với num3.

```

check_third_num:

cmp ecx, [num3]
jg _exit
mov ecx, [num3]

```

Lúc này ecx sẽ đang là num1, ta so sánh tiếp với num3 thông qua lệnh: **cmp ecx, [num3]**.

Nếu **ecx(num1)** lớn hơn **num3** thì ta **\_exit**, tức là **ecx(num1)** là giá trị lớn nhất được tìm thấy.

Trong trường hợp **num1** nhỏ hơn **num2** thì lệnh **check\_third\_num** sẽ không được thực thi, thay vào đó chương trình tiếp tục gán **num2** cho **ecx** để tiếp tục so sánh, nhấn **check\_third\_num** được thực thi.

Nếu **ecx** lớn hơn **num3** thì jg được thực thi tới nhấn **\_exit**, tức là **ecx(num2)** là giá trị lớn nhất được tìm thấy => dừng chương trình. Ngược lại, **num3** được gán cho **ecx**, chương trình tiếp tục thực thi tới nhấn **\_exit**. Lúc này **ecx(num3)** sẽ được xem là giá trị lớn nhất được tìm thấy.

**Bài thực hành 1:** Viết một đoạn chương trình tìm số nhỏ nhất trong 3 số (1 chữ số) **a,b,c** cho trước.

**Bài thực hành 2:** Viết chương trình chuyển đổi một số (number) 123 thành chuỗi '123' Sau đó thực hiện in ra màn hình số 123.

**Bài thực hành 3:** Cải tiến chương trình yêu cầu 1 sao cho tìm số nhỏ nhất trong 3 số bất kỳ (nhiều hơn 1 chữ số)

### \*Gợi ý yêu cầu 2

```
%assign SYS_EXIT    1
%assign SYS_WRITE    4
%assign STDOUT      1

;;; -----
;;; data section
;;; -----
section .data
x    db    123

msgX  db    "x = "

;;; -----
;;; code section
;;; -----
section .text
global _start
_start:

;;; display x
    mov     ecx, msgX
    mov     edx, 4
    call    _printString
```

```

mov     eax, 0
mov     al, byte[x]
call    _printDec
;;; ; exit
mov     ebx, 0
mov     eax, 1
int     0x80

_printString:
    push    eax
    push    ebx

    mov     eax, SYS_WRITE
    mov     ebx, STDOUT
    int     0x80

    pop     ebx
    pop     eax
    ret

;;; -----
;;; ; _println      put the cursor on the next line.
;;; ;
;;; ;
;;; ; Example:
;;; ;
;;; ;     call    _println
;;; ;
;;; ;
;;; ; REGISTERS MODIFIED: NONE
;;; ;
;;; ; -----
_println:
    section .data
.nl      db      10

    section .text
    push    ecx
    push    edx

    mov     ecx, .nl
    mov     edx, 1
    call    _printString

    pop     edx
    pop     ecx
    ret

_printDec:
;;; saves all the registers so that they are not changed by the function

section    .bss
.decstr    resb      10
.ct1       resd      1 ; to keep track of the size of the string

```

```
section .text
pushad                ; save all registers

mov     dword[ct1],0   ; assume initially 0
mov     edi,decstr     ; edi points to decstring
add     edi,9          ; moved to the last element of string
xor     edx,edx        ; clear edx for 64-bit division
whileNotZero:
mov     ebx,10         ; get ready to divide by 10
div     ebx            ; divide by 10
add     edx,'0'        ; converts to ascii char
mov     byte[edi],dl    ; put it in string
dec     edi            ; mov to next char in string
inc     dword[ct1]     ; increment char counter
xor     edx,edx        ; clear edx
cmp     eax,0          ; is remainder of division 0?
jne     .whileNotZero  ; no, keep on looping

inc     edi            ; conversion, finish, bring edi
mov     ecx,edi        ; back to beg of string. make ecx
mov     edx,[ct1]      ; point to it, and edx gets # chars
mov     eax,SYS_WRITE  ; and print!
mov     ebx,STDOUT
int     0x80

popad                ; restore all registers

ret
```

# CHÈN MÃ VÀO TẬP TIN PE

## D. TỔNG QUAN

### 1. Mục tiêu

- Hiểu được cơ chế chèn mã vào file PE của viruses.
- Thực hiện chèn một đoạn mã đơn giản để hiển thị một message box khi mở file NOTEPAD.exe

### 2. Chuẩn bị môi trường

- Cài đặt [CFF Explorer](#) để xem và chỉnh sửa các tham số trong file PE.
- Cài đặt [HxD](#) để đọc và sửa nội dung file PE.
- Cài đặt **IDA Pro** để kiểm tra mã hợp ngữ của chương trình muốn chèn.

*(Vui lòng sử dụng máy ảo Windows để thực hành – liên hệ GVHD để lấy các công cụ liên quan)*

## E. THỰC HÀNH

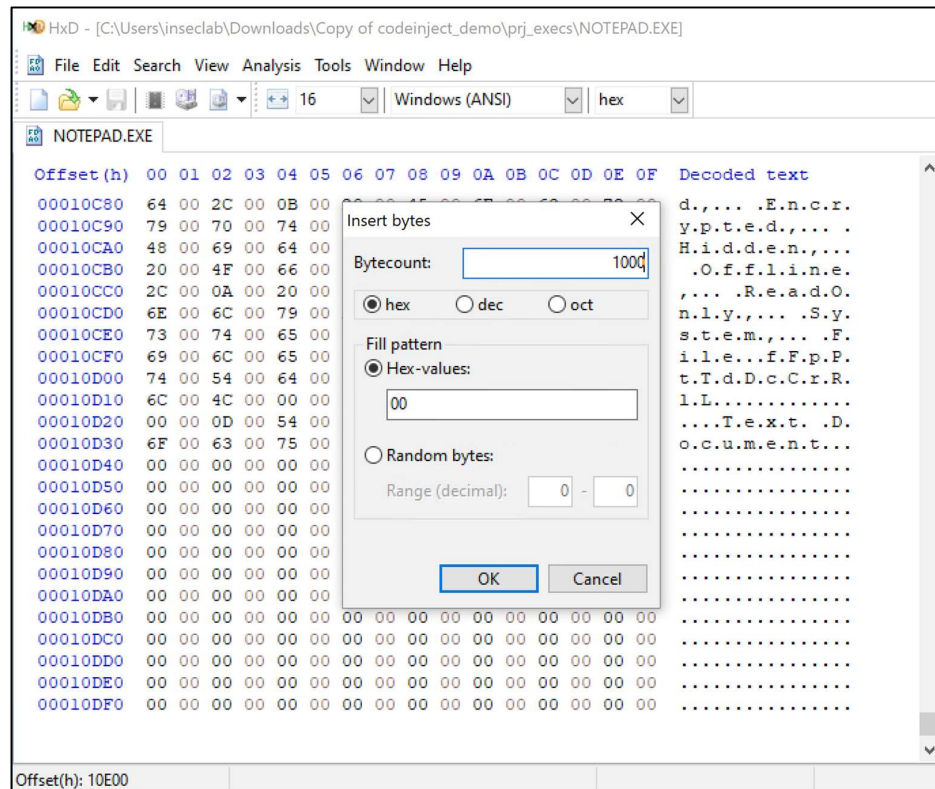
Mở CFF Explorer để đọc nội dung của Notepad.exe. Nhìn vào các nội dung được hiển thị.

- Chọn Optional Header, tìm AddressOfEntryPoint trong bảng bên phải chứa giá trị 0x0000739D. Cũng trong bảng bên phải, tìm ImageBase chứa giá trị 0x01000000. Cộng 2 giá trị này lại ta được 0x0100739D. Đây chính là địa chỉ ảo (virtual address) xác định vị trí thực thi của chương trình.
- Chọn Section Header, ta thấy 3 sections: text, data và rsrc. Để đơn giản, chúng ta sẽ mở rộng rsrc và chèn code vào đây. Mỗi section có chứa 4 thông tin:
  - VirtualSize: kích thước của section khi được load vào bộ nhớ.
  - VirtualAddress(VA): địa chỉ của section khi được load vào bộ nhớ.
  - RawSize: kích thước của section trong PE file.
  - RawAddress(RA): địa chỉ của section trong PE file.



## 1. Tạo vùng nhớ trong tập tin PE

Sử dụng HxD để mở *Notepad.exe*. **Đặt trỏ chuột vào cuối file**, chọn *Edit -> Insert Bytes*, nhập giá trị *0x1000* (có thể chèn nhiều hơn đối với các đoạn mã phức tạp, tuy nhiên sẽ làm tăng kích thước của PE file) và nhấn OK.



## 2. Tạo chương trình cần chèn

1. Ta có một chương trình hiển thị MessageBox như sau:

```
#include <windows.h>
int main(int argc, char * argv[])
{
    MessageBox(NULL, L"Info", L"Code injected", MB_OK);
    return 0;
}
```

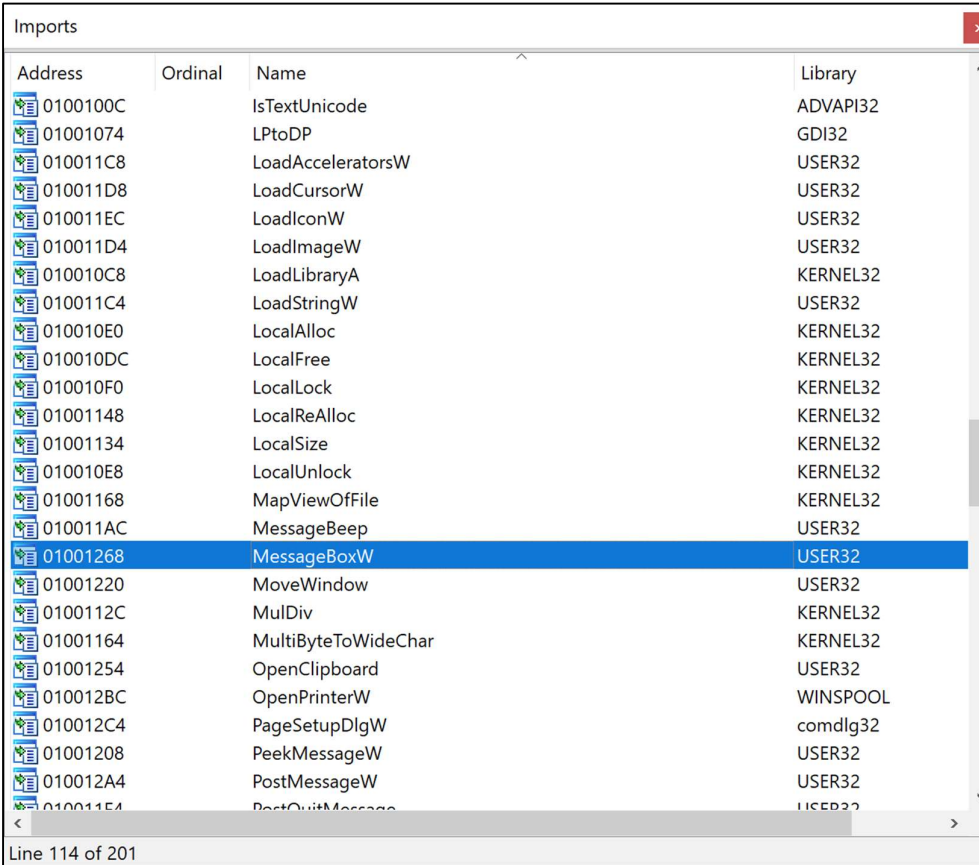
2. Biên dịch chương trình dưới chế độ *Release, Not Using Precompiled Headers*. Sử dụng IDA Pro để mở file PE và xem mã hợp ngữ của chương trình vừa biên dịch.

3. Về cơ bản, chương trình gồm 5 dòng lệnh (sử dụng chức năng hexview để xem mã hex của từng lệnh).

```
push 0 ; 6a 00
push Caption ; 68 X
push Text ; 68 Y
push 0 ; 6a 00
call [MessageBoxW] ; ff15 Z
```

Để chèn đoạn code này vào Notepad.exe, ta phải đi tìm các giá trị (X, Y, Z) phù hợp.

4. Giá trị **Z** chính là địa chỉ của hàm *MessageBoxW* được import từ thư viện USER32.dll. Trong IDA Pro, mở *Notepad.exe*, chọn **View -> Open Subviews -> Imports** và ta thấy địa chỉ của hàm *MessageBoxW* chính là 01001268.

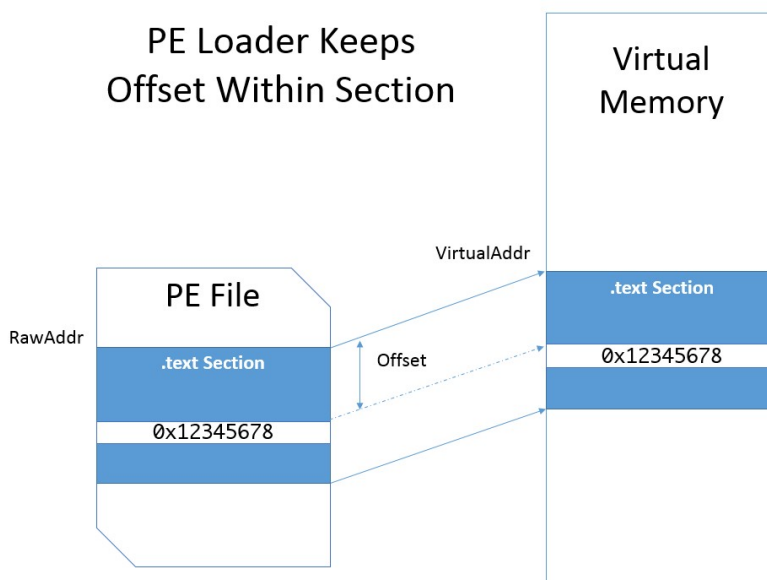


| Address  | Ordinal | Name                | Library  |
|----------|---------|---------------------|----------|
| 0100100C |         | IsTextUnicode       | ADVAPI32 |
| 01001074 |         | LPtoDP              | GDI32    |
| 010011C8 |         | LoadAcceleratorsW   | USER32   |
| 010011D8 |         | LoadCursorW         | USER32   |
| 010011EC |         | LoadIconW           | USER32   |
| 010011D4 |         | LoadImageW          | USER32   |
| 010010C8 |         | LoadLibraryA        | KERNEL32 |
| 010011C4 |         | LoadStringW         | USER32   |
| 010010E0 |         | LocalAlloc          | KERNEL32 |
| 010010DC |         | LocalFree           | KERNEL32 |
| 010010F0 |         | LocalLock           | KERNEL32 |
| 01001148 |         | LocalReAlloc        | KERNEL32 |
| 01001134 |         | LocalSize           | KERNEL32 |
| 010010E8 |         | LocalUnlock         | KERNEL32 |
| 01001168 |         | MapViewOfFile       | KERNEL32 |
| 010011AC |         | MessageBeep         | USER32   |
| 01001268 |         | MessageBoxW         | USER32   |
| 01001220 |         | MoveWindow          | USER32   |
| 0100112C |         | MulDiv              | KERNEL32 |
| 01001164 |         | MultiByteToWideChar | KERNEL32 |
| 01001254 |         | OpenClipboard       | USER32   |
| 010012BC |         | OpenPrinterW        | WINSPOOL |
| 010012C4 |         | PageSetupDlgW       | comdlg32 |
| 01001208 |         | PeekMessageW        | USER32   |
| 010012A4 |         | PostMessageW        | USER32   |
| 010011E4 |         | PostQuitMessage     | USER32   |

5. Trong HxD, ta chọn địa chỉ 0x00011000 trong vùng nhớ đã được mở rộng (bước C1) để lưu trữ mã hợp ngữ, 0x00011040 để lưu trữ Caption và 0x00011060 để lưu trữ Text. Ta có thể tùy chọn những vị trí khác tùy thích.

Đối với mỗi section, loader sẽ copy section tại RA trong PE file sang bộ nhớ ảo tại VA trong khi vẫn giữ đúng offset bên trong section đó.

$$\text{Offset} = \text{RA} - \text{Section RA} = \text{VA} - \text{Section VA} \quad (1)$$



Giá trị X có thể được tìm dựa vào công thức (1)

$$0x00011040 - 0x00008400 = X - 0x000B000$$

$$X = 0x00013C40$$

Cộng thêm ImageBase, suy ra  $X = 0x01013C40$ . Tương tự,  $Y = 0x01013C60$ .

Như vậy, đoạn code này thực hiện chức năng như mong đợi và có địa chỉ mới là:

$$\text{new\_entry\_point} = 0x00011000 - 0x00008400 + 0x000B000 = 0x00013C00$$

### 3. Thiết lập lệnh quay về AddressOfEntryPoint ban đầu

- Để chương trình *Notepad.exe* tiếp tục được thực thi sau khi đã chạy đoạn code trên, ta cần chèn dòng lệnh quay về *AddressOfEntryPoint* cũ ngay sau đoạn code ở bước 2. **jmp relative\_VA**
- Đối với lệnh jmp, đích đến (*old\_entry\_point*) sẽ được tính bằng cách cộng giá trị *relative\_VA* vào thanh ghi PC khi lệnh được thực thi. Bởi vì PC luôn trở đến vị trí đầu của câu lệnh kế tiếp, cho nên cần phải tính 5 bytes của câu lệnh jmp nữa. Ta có công thức sau:

$$\text{old\_entry\_point} = \text{jmp\_instruction\_VA} + 5 + \text{relative\_VA} \quad (2)$$

Nếu đặt lệnh jmp sau 5 câu lệnh ở bước 2 thì  $\text{jmp\_instruction\_VA} = 0x01013C14$ .

old\_entry\_point = 0x0100739D chính là giá trị AddressOfEntryPoint ban đầu đã cộng ImageBase.

Suy ra, relative\_VA = 0x0100739D - 5 - 0x01013C14 = 0xFFFF3784.

- Đến đây, ta đã có một đoạn mã hợp ngữ hoàn chỉnh để chèn vào Notepad.exe. Các địa chỉ được biểu diễn theo thứ tự little endian (x86).

```
push 0          ; 6a 00
push Caption    ; 68   403C0101
push Text       ; 68   603C0101
push 0          ; 6a 00
call [MessageBoxW] ; ff15 68120001
jmp Origanl_Entry_Point ; e9 8437FFFF
```

#### 4. Chèn vào Notepad.exe

Sử dụng HxD để chèn đoạn mã cùng với giá trị Caption và Text vào Notepad.exe. Lưu lại file.

| Offset(h) | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 0A | 0B | 0C | 0D | 0E | 0F |                  |
|-----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------------------|
| 00010FF0  | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | .....            |
| 00011000  | 6A | 00 | 68 | 60 | 3C | 01 | 01 | 68 | 40 | 3C | 01 | 01 | 6A | 00 | FF | 15 | j.h'<..h@<..j.ý. |
| 00011010  | 68 | 12 | 00 | 01 | E9 | 84 | 37 | FF | FF | 00 | 00 | 00 | 00 | 00 | 00 | 00 | h...é„7ýý.....   |
| 00011020  | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | .....            |
| 00011030  | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | .....            |
| 00011040  | 43 | 00 | 6F | 00 | 64 | 00 | 65 | 00 | 20 | 00 | 69 | 00 | 6E | 00 | 6A | 00 | C.o.d.e. .i.n.j. |
| 00011050  | 65 | 00 | 63 | 00 | 74 | 00 | 65 | 00 | 64 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | e.c.t.e.d.....   |
| 00011060  | 49 | 00 | 6E | 00 | 66 | 00 | 6F | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | I.n.f.o...ð..... |
| 00011070  | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | .....            |
| 00011080  | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | .....            |
| 00011090  | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | .....            |
| 000110A0  | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | .....            |

#### 5. Hiệu chỉnh các tham số trong PE header

- Sử dụng CFF Explorer để thay đổi các giá trị sau:

- Trong Section Headers, thay đổi .rsrc Section Header.

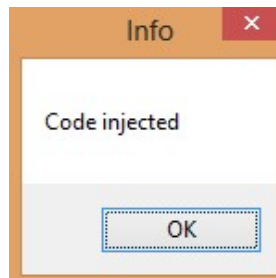
| Name    | Virtual Size | Virtual Address | Raw Size | Raw Address |
|---------|--------------|-----------------|----------|-------------|
|         |              |                 |          |             |
| Byte[8] | Dword        | Dword           | Dword    | Dword       |
| .text   | 00007748     | 00001000        | 00007800 | 00000400    |
| .data   | 00001BA8     | 00009000        | 00000800 | 00007C00    |
| .rsrc   | 00009958     | 0000B000        | 00009A00 | 00008400    |

- Trong Optional Headers, tăng SizeOfImage lên 0x1000.
- Trong Optional Headers, chỉnh sửa AddressOfEntryPoint thành 0x00013C00.

2. Lưu lại file.

## 6. Kiểm tra kết quả

Khi thực thi Notepad.exe, một cửa sổ xuất hiện như sau:



**Bài thực hành 4:** Thực hiện lại các bước trên thay đổi phần Text là MSSV.

**Bài thực hành 5:** Bằng cách không tạo thêm vùng nhớ mở rộng vào tập tin PE, tận dụng vùng nhớ trống để chèn chương trình cần chèn trên tập tin Notepad và calc.

## F. YÊU CẦU & ĐÁNH GIÁ

- Sinh viên tìm hiểu và thực hành theo hướng dẫn, thực hiện **theo nhóm đã đăng ký**.
- Nộp báo cáo kết quả gồm **Code, File được export** và chi tiết những việc (**Report**) mà nhóm đã thực hiện, quan sát thấy và kèm ảnh chụp màn hình kết quả (nếu có); giải thích cho quan sát (nếu có).

### Báo cáo:

- File **.PDF**. Tập trung vào nội dung, không mô tả lý thuyết.
- Đặt tên theo định dạng: **[Mã lớp]-LabX\_MSSV1-MSSV2-MSSV3**.
- Nếu báo cáo có nhiều file, nén tất cả file vào file **.ZIP** với cùng tên file báo cáo.
- Nộp file báo cáo trên theo thời gian đã thống nhất tại [courses.uit.edu.vn](https://courses.uit.edu.vn).

*Bài sao chép, trộm, ... sẽ được xử lý tùy mức độ vi phạm.*

## HẾT

*Chúc các bạn hoàn thành tốt!*