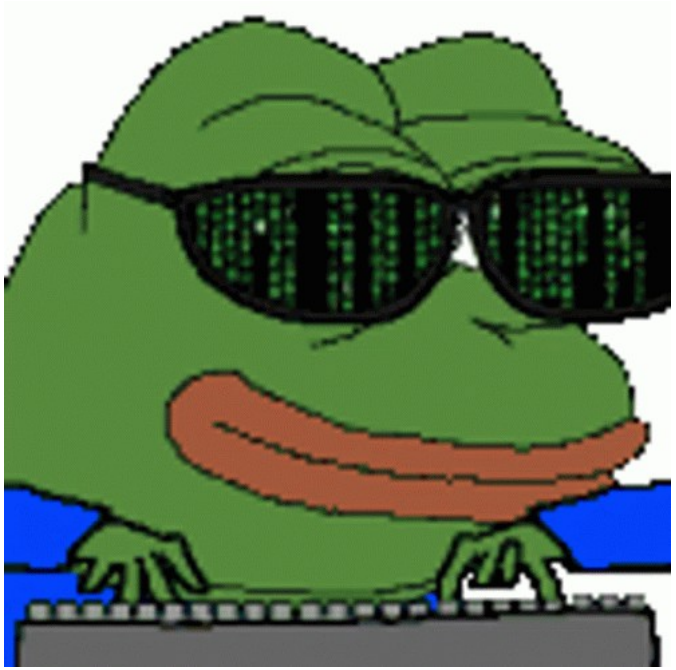
HackerFrogs Afterschool
x86 Assembly Basics: Part 1

Class:
Binary Exploitation

Workshop Number:
AS-BIN-00

Document Version:
1.0

Special Requirements:
None



x86 Assembly NASM (Linux)

We'll be learning how to code in x86 Assembly language in preparation for learning binary exploitation

The specific assembler we'll use is NASM (Netwide Assembler)

x86 Assembly NASM (Linux)

We'll be learning how to code in x86 Assembly language in preparation for learning binary exploitation

x86 Assembly NASM (Linux)

The specific assembler we'll use is NASM (Netwide Assembler). An assembler is a program which converts code into a working binary executable

Memory Registers (x86)

In order to code in Assembly, we need to understand the concept of memory registers, which are storage locations inside the compter processor, and contain data required for process execution

Memory Registers (x86)

`EAX  EBX  ECX  EDX`

We won't be going in-depth with memory registers today, but we do need to understand that our Assembly program will write data to these four registers

Memory Registers (x86)

`EAX  EBX  ECX  EDX`

For the Hello World program, we should understand that we need to write information into these registers before executing a system call which prints the message (Hello World) to the console

Anatomy of an Assembly Code File

```
section .data
 section .bss
section .text
```

There are three sections to Assembly code files, the .data, .bss, and .text sections

.data Section

```
SECTION .data
msg     db      'Hello World!', 0Ah
```

The .data section stores initialized data, such as constants and variables. In the Hello World program, it stores the string to be printed

.data Section

```
SECTION .data
msg     db      'Hello World!', 0Ah
```

The `msg` in the code acts as a reference (variable name) for this string. The `0Ah` represents the newline character in ASCII, which creates a new line after the message is printed out

.data Section

```
SECTION .data
msg     db      'Hello World!', 0Ah
```

The `db` stands for **define byte**, which reserves a sequence of bytes in memory

.bss Section

There is no .bss section in the Hello World code, but we'll talk about it here for the sake of completeness. It's used to store uninitialized global or static values, which could be variables that are defined during program runtime

.text Section

```
SECTION .text
global  _start

_start:

    mov     edx, 13
    mov     ecx, msg
```

The `.text` section of assembly code contains the program instructions

.text Section

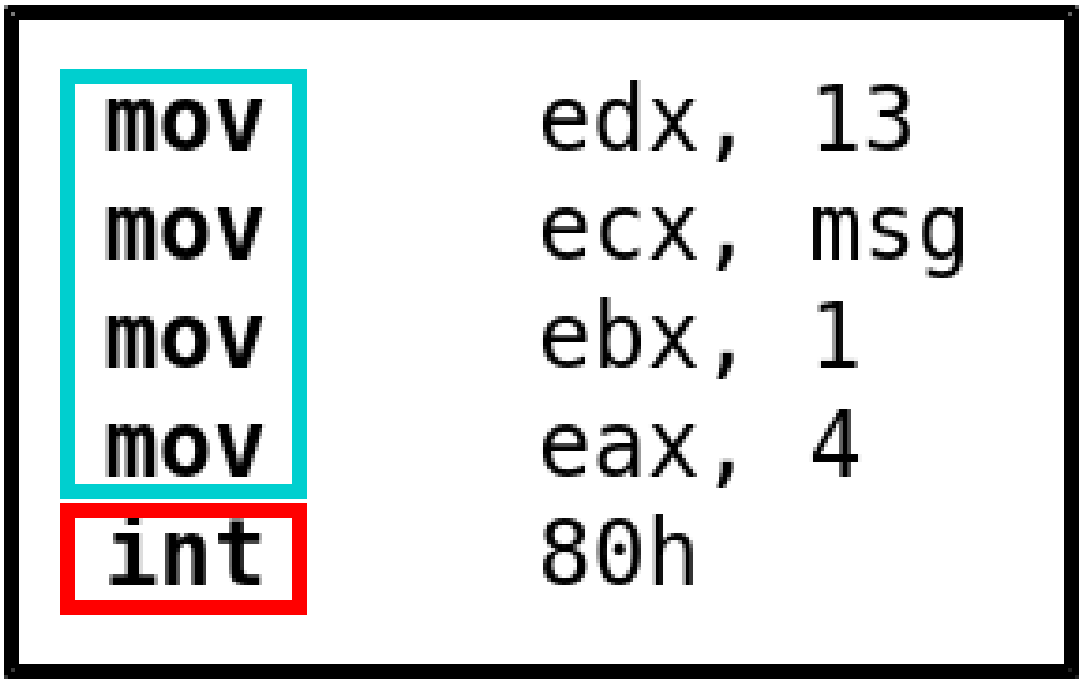
```
SECTION .text
global  _start

_start:

    mov     edx, 13
    mov     ecx, msg
```

The `global` directive in Assembly identifies a label as global symbol, and here it defines `_start` as the entry point for the program

The MOV and INT Instructions



There are two instructions in this code, the MOV and INT instructions

The MOV Instruction

```
mov destination, source
```

```
mov edx, 13
```

When using `mov`, we specify the destination first, then the value to be moved there

The INT Instruction

```
INT interrupt_number
```

```
INT 80h
```

The `INT` instruction is used to trigger software interrupts, allowing the program to request services from the operating system

The INT Instruction

```
INT interrupt_number
```

```
INT 80h
```

In the Hello World code, `INT 80h` performs a Linux system call (syscall). Because the value 4 was written to the EAX register, Linux performs the write syscall, which prints to the console