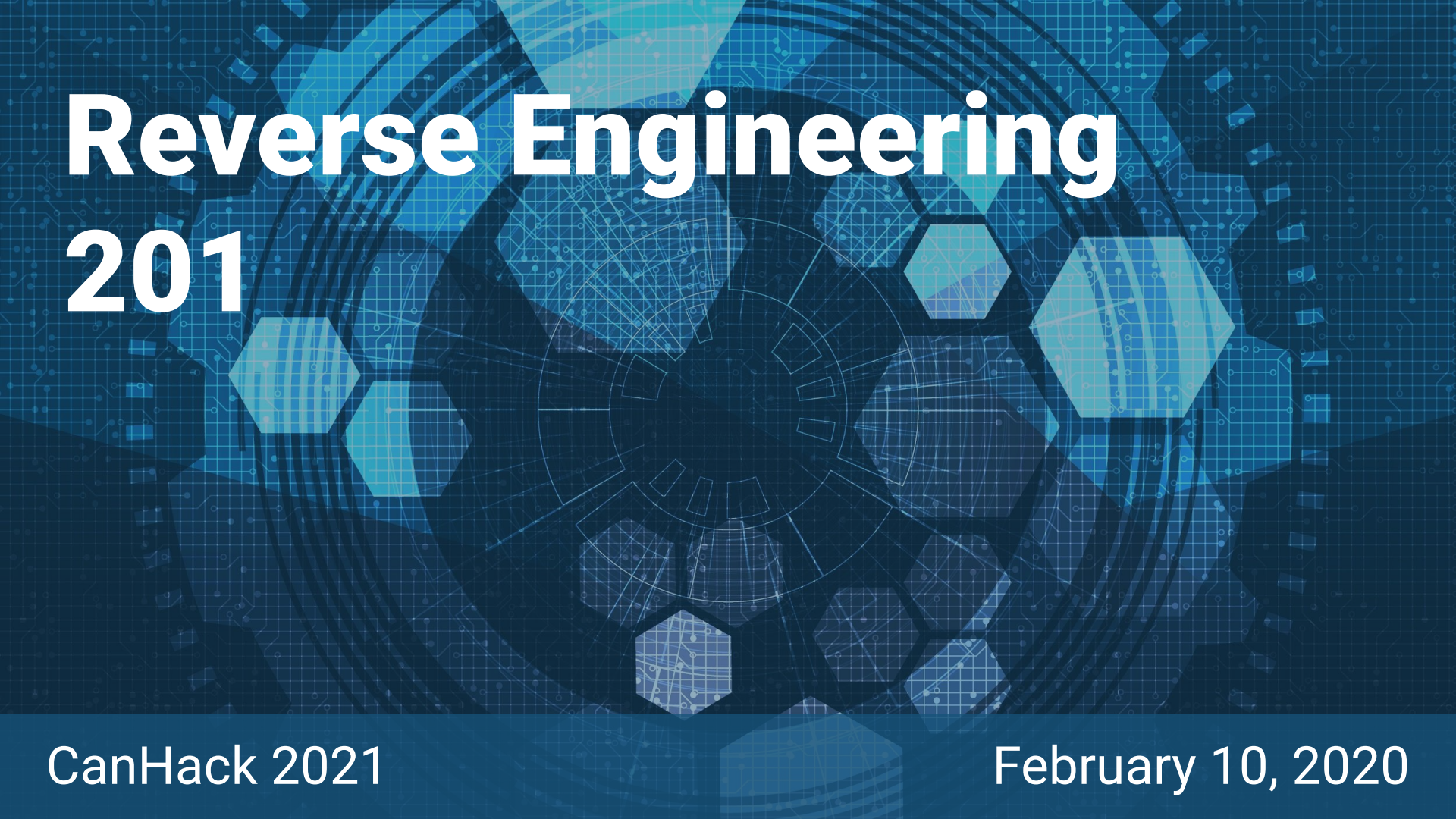

Reverse Engineering
201
CanHack 2021
February 10, 2020

# Digital Badging

- To provide students proof of knowledge they've gained during CanHack 2021
- Visually represent student's successful completion of the CanHack workshops
- Students can share them on their social media profiles, resumes and college/university applications.

# How to earn this digital badge

Participate in CanHack competition to receive this badge and get a passing score of 50% on the CanHack quiz to get a digital badge.

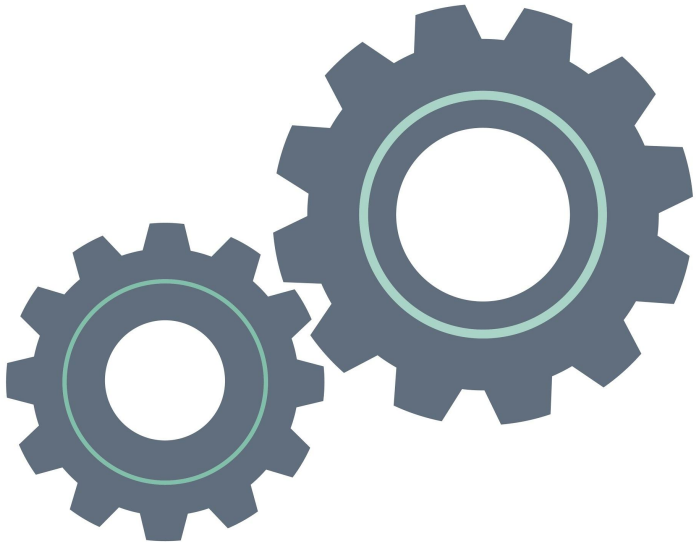
The competition registration opens on Tuesday, February 16th, 2021.
The quizzing for the badge begins on Monday, February 22nd, 2021 - March 30th, 2021
The competition takes place from March, 16th, 2021 - March 30th, 2021.

This badge is approved by The Royal Bank of Canada (RBC) and [Cylab Carnegie Mellon University's Security and Privacy Research Institute](#).

These badges are valued in the cybersecurity industry and will look great on internship, volunteering and college/university applications.

# Overview of Reverse Engineering

- What is Reverse Engineering
- Java
- Assembly Language
- What are registers and how do they work?
- Assembly Instructions



# What is Reverse Engineering?

Taking something apart and putting it back together again to understand how it works

**Uses:**

- Analyze malware and malicious programs to understand how they work to prevent against them
- Breaking down code to better understand the potential vulnerability of a software
- Understand how certain parts of the program work

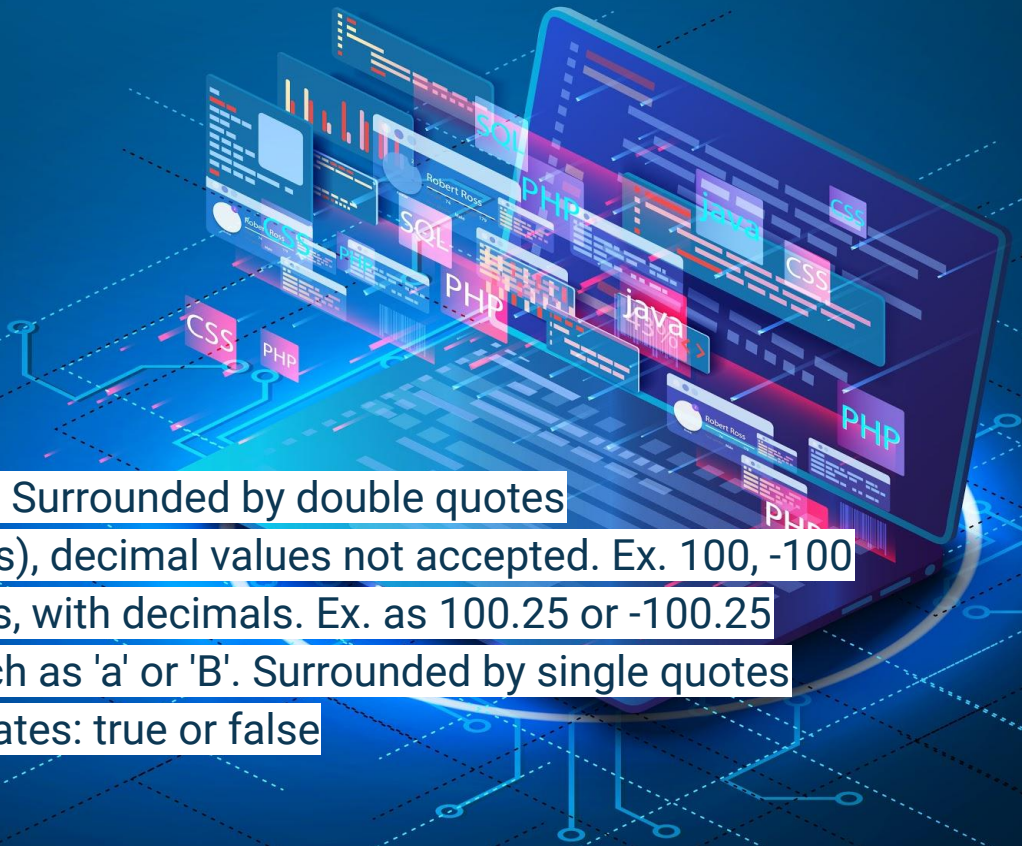
# What needs to be Reverse Engineered?

- Code
- Binary Files
- Assembly Instructions
- Malware
- Applications
- Programs written in Java, Python, C

# Java

Java is a programming language

Used to build applications(desktop, mobile, web), games, and more




## Variables

- string - stores text, such as "Hello". Surrounded by double quotes
- int - stores integers (whole numbers), decimal values not accepted. Ex. 100, -100
- float - stores floating point numbers, with decimals. Ex. as 100.25 or -100.25
- char - stores single characters, such as 'a' or 'B'. Surrounded by single quotes
- boolean - stores values with two states: true or false

# Vault Door Training

```
import java.util.*;

class VaultDoorTraining {
    public static void main(String args[]) {
        VaultDoorTraining vaultDoor = new VaultDoorTraining();
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter vault password: ");
        String userInput = scanner.next();
                String input = userInput.substring("picoCTF{".length(),userInput.length()-1);
                if (vaultDoor.checkPassword(input)) {
                    System.out.println("Access granted.");
                } else {
                    System.out.println("Access denied!");
                }
    }

    // The password is below. Is it safe to put the password in the source code?
    // What if somebody stole our source code? Then they would know what our
    // password is. Hmm... I will think of some ways to improve the security
    // on the other doors.
    //
    // -Minion #9567
    public boolean checkPassword(String password) {
        return password.equals("w4rm1ng_Up_w1tH_jAv4_be8d9806f18");
    }
}
```

```
import java.util.*;

class VaultDoorTraining {
```

Every line of code that runs in Java must be inside a class.

The name of the java file must match the class name.

```
import java.util.*;

class VaultDoorTraining {
    public static void main(String args[]) {
```

The main() method is required and you will see it in every Java program

```
import java.util.*;

class VaultDoorTraining {
    public static void main(String args[]) {
        VaultDoorTraining vaultDoor = new VaultDoorTraining();
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter vault password: ");
        String userInput = scanner.next();
```

The Scanner class is used to get user input, and it is found in the java.util package. System.in tells the java compiler that system input will be provided through console(keyboard).

Asks the user for “Enter vault password:”

Get user input

```
import java.util.*;

class VaultDoorTraining {
    public static void main(String args[]) {
        VaultDoorTraining vaultDoor = new VaultDoorTraining();
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter vault password: ");
        String userInput = scanner.next();
            String input = userInput.substring("picoCTF{".length(),userInput.length()-1);
            if (vaultDoor.checkPassword(input)) {
                System.out.println("Access granted.");
            } else {
                System.out.println("Access denied!");
            }

}

    // The password is below. Is it safe to put the password in the source code?
    // What if somebody stole our source code? Then they would know what our
    // password is. Hmm... I will think of some ways to improve the security
    // on the other doors.
    //
    // -Minion #9567
    public boolean checkPassword(String password) {
        return password.equals("w4rm1ng_Up_w1tH_jAv4_be8d9806f18");
    }
}
```

Check the user.Input but the string “picoCTF{” and the length-1 which will be “}” this character is not checked

Compare if userinput = the string provided

If user input matches then print “Access granted” and if not then “Access denied”

# Let’s take a look at some additional Java programs

# Assembly Language

An assembly language is a low-level programming language designed for a specific type of processor.

Also called assembly or ASM

**The instruction below tells an x86/IA-32 processor to move an immediate 8-bit value into a register:**

```
10110000 01100001
```

**This binary computer code can be made more human-readable by expressing it in hexadecimal:**

```
B0 61
```

B0 means 'Move a copy of the following value into AL, and 61 is a hexadecimal representation of the value 01100001, which is 97 in decimal.

**Assembly language for the 8086 family provides the mnemonic MOV (an abbreviation of move) for instructions such as this, so the machine code above can be written as follows in assembly language, complete with an explanatory comment if required, after the semicolon. This is much easier to read and to remember.**

```
MOV AL, 61h      ; Load AL with 97 decimal (61 hex)
```

Source: Wikipedia

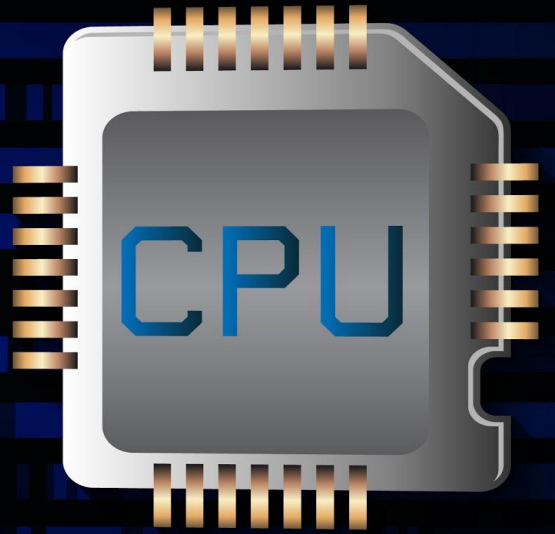
# Central Processing Unit (CPU)

CPU comprises of:

- The Arithmetic Logic Unit (ALU)
  - The ALU consists of the Arithmetic Unit (responsible for mathematical functions) and Logic Unit (responsible for logical operations)

- The Control Unit (CU)
  - Controls and directs the main memory, (ALU), input and output devices, and is also responsible for the instructions that are sent to the CPU

- Registers
  - Small, extremely high-speed CPU storage locations where data can be efficiently read or manipulated, hold data temporarily



# Registers

What do registers do?

- Holds temporary data that is is needed by the CPU to execute instructions
- Perform operations
- Store resulting data

Only hold a small amount of data, 64-bit architecture CPU’s hold 64 bits of data/register and 32-bit architecture CPU’s hold 32 bits of data/register

The CPU Architecture determines the design of the processor, instructions that are supported, size of registers and other factors.

Common architecture for processors is x86 developed by Intel

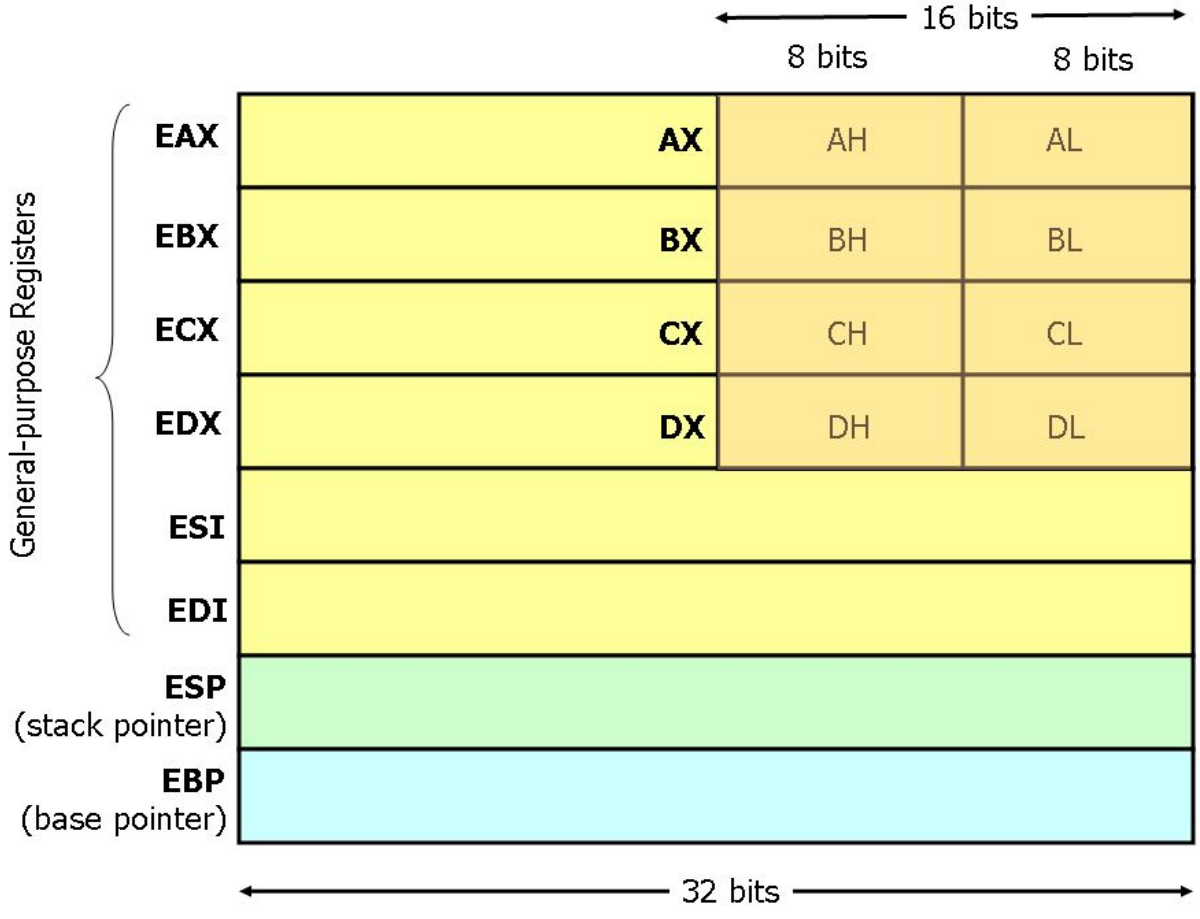



Image Source: University of Virginia Computer Science

**EAX** - Accumulator Register - used for storing operands and result data

**EBX-** Base register - Points to data

**ECX** - Counter Register - Loop operations

**EDX**- Data register. Input/output operations.

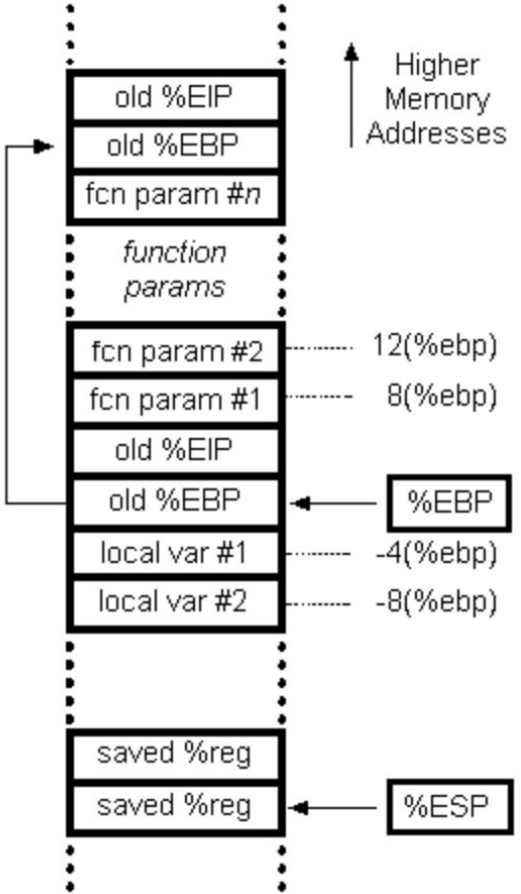
**ESI/EDI**- Source/Destination index for string operations.

**ESP**- Current position of data or address within the program stack, which changes automatically based on the operation

**EBP**- Frame pointer, contains the base address of the function's frame.

# x86 architecture

- All x86 architectures use a stack as a temporary storage area in RAM that allows the processor to quickly store and retrieve data in memory
- Higher memory addresses are at the top of the stack
- LIFO (Last In First Out) Method is used, items that are “pushed” on top of the stack are “popped” first

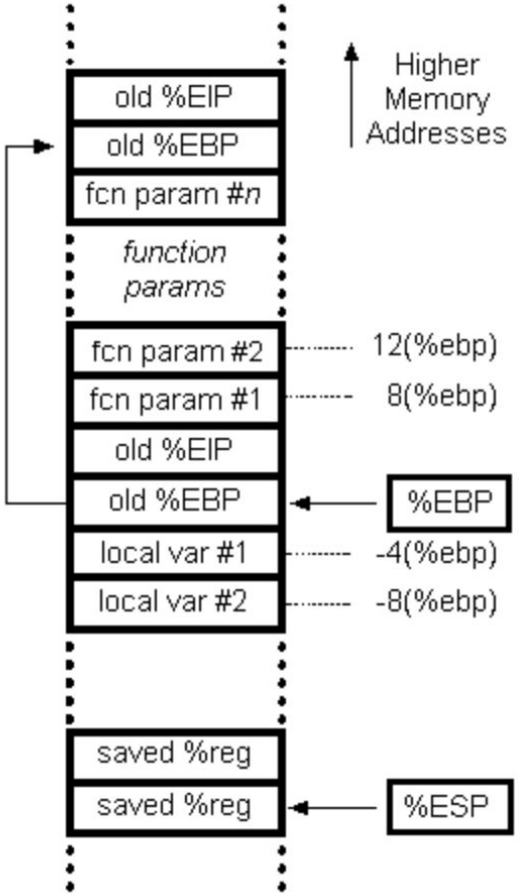


# x86 architecture

- Data is stored using the Little Endian method
- 0x12345678 , it would be entered as 78, 56, 34, 12 into the stack
- In 32-bit registers, memory addresses of registers are 4 bytes apart
- By using a base pointer the return address will always be at ebp+4, the first parameter will always be at ebp+8, and the first local variable will always be at ebp-4

Note*

Two's complement is the standard way of representing negative integers in binary.



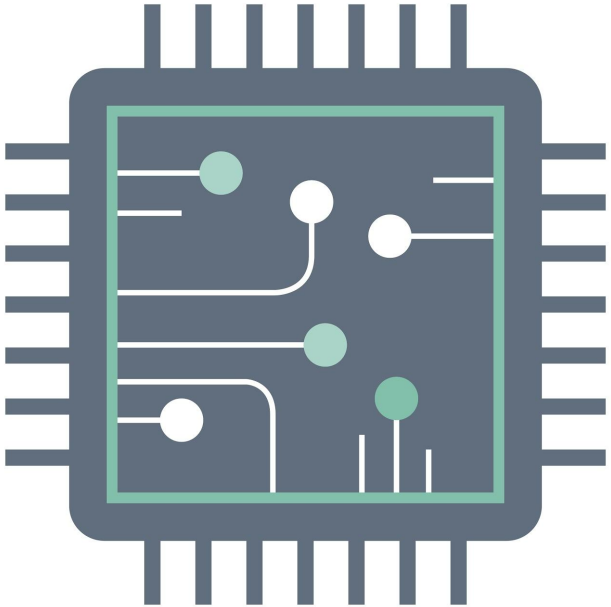
# x86 assembly instructions

Assembly instructions represent a single operation for the CPU to perform.

**Examples of Assembly instructions:**

| | |
|---|---|
| **mov D, S** | Move source to destination |
| **add S, D** | Add source to destination |
| **je/jne** | Jump when equal / Jump when not equal |
| **jg** | Jump when greater than |

| | | |
|---|---|---|
| BYTE | 00 | 1 byte/8 bits |
| WORD | 00 00 | 2 bytes/ 16 bits |
| DWORD | 00 00 00 00 | 4 bytes/ 32 bits |



Additional Instructions

# Prologue for x86 architecture

The function prologue prepares the stack and registers for use within the function

A function prologue typically looks like this:

- Push current base pointer onto the stack, so it can be restored later.
- Assigns the value of base pointer to the address of stack pointer (which is pointed to the top of the stack) so that the base pointer will pointed to the top of the stack.
- Moves the stack pointer further by decreasing its value to make room for function's local variables.

```
push    ebp
mov     ebp, esp
sub     esp, N
```

| | |
|---|---|
| Data 2 from Function | [ebp + 0xc] |
| Data 1 from Function | [ebp + 0x8] |
| Return address [ebp + 0x4] | |
| Old ebp | |
| Local variable 1 | [ebp - 0x4] |
| Local variable 2 | [ebp - 0x8] |
| | [ebp - 0xc] |
| | [ebp - 0x10] |

# Lets try reading some assembly instructions

```
asm1:
        <+0>:push   ebp
        <+1>:mov    ebp,esp
        <+3>:cmp    DWORD PTR [ebp+0x8],0x3fb
        <+10>:      jg     0x512 <asm1+37>
        <+12>:      cmp    DWORD PTR [ebp+0x8],0x280
        <+19>:      jne    0x50a <asm1+29>
        <+21>:      mov    eax,DWORD PTR [ebp+0x8]
        <+24>:      add    eax,0xa
        <+27>:      jmp    0x529 <asm1+60>
        <+29>:      mov    eax,DWORD PTR [ebp+0x8]
        <+32>:      sub    eax,0xa
        <+35>:      jmp    0x529 <asm1+60>
        <+37>:      cmp    DWORD PTR [ebp+0x8],0x559
        <+44>:      jne    0x523 <asm1+54>
        <+46>:      mov    eax,DWORD PTR [ebp+0x8]
        <+49>:      sub    eax,0xa
        <+52>:      jmp    0x529 <asm1+60>
        <+54>:      mov    eax,DWORD PTR [ebp+0x8]
        <+57>:      add    eax,0xa
        <+60>:      pop    ebp
        <+61>:      ret
```

2e0

Compare 2e0 with 3fb
2e0 is smaller than 3fb so don't jump

Compare 2e0 with 280
Jump if not equal to +29

Move 2e0 into eax → **eax = 2e0**

2e0 - 0xa= *2D6*

| | |
|---|---|
| 2e0 | [ebp + 0x8] |
| Return address | [ebp + 0x4] |
| Old ebp | |

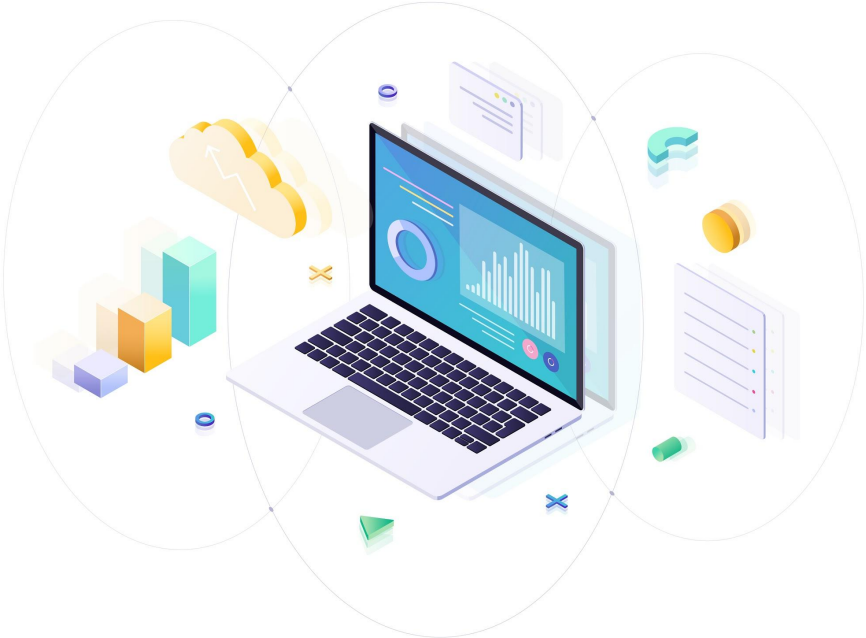
# Resources

**Java:**

https://www.w3schools.com/java/default.asp

**Assembly Language:**

https://www.secjuice.com/guide-to-x86-assembly/
https://www.cs.virginia.edu/~evans/cs216/guides/x86.html
http://unixwiz.net/techtips/win32-callconv-asm.html
Hex Calculator for mathematical operations
Two complements calculator for Hex



# Hacking the All-Female Prize: CanHack Cybersecurity Webinar for Girls

Weds. February 24, 2021, 6-7:00pm EST

- Explore cybersecurity, learn tips and tricks to succeed in CanHack, and explore future opportunities in this exciting field!
- This is a safe space for girls to meet other like-minded girls and ask questions about the contest and the cyber sector at large.
- The most important thing to remember is you don't need to be a cybersecurity expert to join... and you might even be the FIRST CanHack team to win the All-Female Prize of $2,000!

# Thank you!

Questions?

See you next week for our last workshop on
Binary Exploitation 201!

Registration opens next week for the competition