## CS330 (261) with Prof. Basit Qureshi

## Assignment 1

Weight: 5%

**Learning Objectives:** After completing this assignment, students should be able to:

- Compile and execute C and Java programs using GCC and the Java compiler in a Linux environment.
- Differentiate between high-level language, assembly language, machine code, and Java bytecode.
- Use Linux development tools such as gcc, objdump, hexdump, javac, and javap to inspect program translation and execution.
- Identify the relationship between assembly instructions, machine-code bytes, and opcodes, particularly for an arithmetic operation.
- Explain the compilation and execution processes of C and Java, including the role of the JVM and JIT compiler.

The assignment is composed of two parts each requiring a task to be completed. To complete this assignment, you need a Linux VM to be able to execute some shell commands.

---

**Part I** As we know, the high level languages such as C etc, compile program so it can execute on hardware. This involves translation of the C code to assembly and machine languages.

Following is a C program called a.c.
In this part of the assignment, you will use a simple C program to add two values followed by displaying the results. This program would be compiled using the GNU C compiler, gcc to an executable.

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char **argv){
  int a = 10;
  int b = 5;
  int c = a + b;
  printf("Sum=%d\n", c);
}
```

Compiling a.c program
Compile the program using gcc

```
gcc a.c -o a
```

this compiles a.c to a executable a
To view the assembly language code for a.c, use the -S flag

```
gcc a.c -S
```

This generates the assembly code in a text file a.s. Use a text editor to read the assembly code.

You can use a disassembly tool to disassemble a executable binary such as

```
objdump -d executable_file
```

Use direction operator to write the output to a text file for editing:

```
objdump -d executable_file > textfile
```

Lets look at the underlying bytes of the assembly code to understand how it executes on the hardware.

```
hexdump -C textfile.txt
```

The -C option gives a convenient hex + ASCII display.

Use the direction operator to write the output to another textfile Hex.txt

**TASK1: Read the textfile and identify where the two numbers A and B are added C? You need to figure out the machine code bytes and assembly instructions. You previously learned about OPCODE, can you identify the opcode for addition?**

## Part II Write a similar program in java (a.java)

```
public class a{
    public static void main(String [] args) {
        int A = 10;
        int B = 5;
        int C = A + B;
        System.out.println("Sum = " + C);
    }
}
```

Use the java compiler to compile your code in bash shell.

```
javac a.java
```

This creates a class file (a.class). Run it

```
Java a
```

The class file contains bytecode and not machine code. Do this:

```
javap -c a
```

observe the output. How is this different from assembled code from c?

The bytecode executes on JVM/JIT compiler. Newer versions of JDK support:

```
java -XX:+PrintCompilation a
```

This tells you that the JVM's JIT compiler is compiling methods.

**TASK2:**

- **Explain when the C program is translated into native machine code and when the Java program is translated into native machine code.**
- **Draw a diagram showing the complete execution path of both programs, from source code to CPU.**

## Submission:

Assignment report would be hand-written. You are allowed to copy paste screen-shots to demonstrate the output of various commands.

## Grading:

Task 1 — 2 points
Locate the addition in assembly: 0.75
Identify corresponding machine-code bytes: 0.75
Identify/explain the opcode and relationship between instruction and bytes: 0.50

Task 2 — 3 points
Explain C compilation to native code: 0.75
Explain Java bytecode/JVM/JIT process: 0.75
Accurate C vs. Java comparison: 0.50
Complete execution-path diagram: 1.00