# Q1.Assembly Language Programming

Definition: In this question, you are asked to implement an assembly application in RISC-V ISA. Your application first asks a positive integer value; N, from the console. Then, it prints a NxN right triangle to the screen by using \* s. For input 7, your output should be as in the example:

\*

\* \*

\* \* \*

\* \* \* \*

\* \* \* \* \*

\* \* \* \* \* \*

\* \* \* \* \* \* \*

# Q2.Assembly Language, Programming a Recursive Method

Definition: In this question, you are asked to implement a recursive code assembly in RISC-V ISA.

Your application should ask an N value from the console. Then it calculates the F(N) according to the given definition of f(x).

F(x) =

# Q3.Assembly Language, Programming with Arrays

Definition: In this question, you are asked to implement an assembly code in RISC-V, which operates on arrays.

Your application first asks an integer value for the number of elements in an array. Then, that many times it gets an integer value from the console for the elements of the array. Then it sorts the array and prints the sorted array to the screen.

Example:

Size of the array?: 5

1: 13

2: 25

3: 2

4: 17

5: 21

Your sorted array is: 2 13 17 21 25

Rules:

* Your code will be tested in RARS simulator. It is written in java and you can get the jar file from the following link: <https://github.com/TheThirdOne/rars/releases/tag/continuous> Also you can find many other tools to simulate/emulate your RISC-V code. You are allowed to use any of them.
* You must do all work individually. All submitted studies will be passed through similarity check.
* Finding an online code or using converter tools will not help you learn about computer assembly. Also, there might be other students in the class using that specific resource.

Hints:

* To be able to read from the console and write to console, you need to use system calls. RARS provides a small set of OS–like services through the system call (ecall) instruction. To request a service, a program loads the system call code into register a7 and arguments into registers $a0–$a3. System calls that return values put their results in register a0. You can see the code pieces for reading/printing string/integer values below.

.global \_start

\_start:

la a0, helloworld # prepare to print string

li a7, 4 # print string

ecall

li a0, 0 # File descriptor, 0 for STDIN

la a1, str1 # Address of buffer to store string

li a2, 255 # Maximum number of chars to store

li a7, 63 # System call code for read string

ecall

la a0, str1 # prepare to print string

li a7, 4 # print string

ecall

li a0, 0 # File descriptor, 0 for STDIN

li a7, 5 # System call code for read integer. The value will be in a0

ecall

li a0, 42

li a7, 1 # Print integer. (the value is taken from a0)

ecall

addi a7, zero, 93 #Exit process

addi a0, zero, 13

ecall

.data

helloworld:

.ascii "Merhaba\n"

str1:

.space 255