RE #1

Task:

Figure out what the code does.

Procedure:

I tackled this in two parts, first the arguments and then the loop.

A close-up of a white background

Description automatically generated  
The description says there are four arguments but initially it looks like there are three.

According to the ABI manual arguments 1, 2, and 3 are stored in rdi, rsi, and rdx respectively with argument 4 going to rcx (keep this in mind for later).

rdi is moved to r8

rbx is pushed to the stack

rsi is moved to rdi

rdx is moved to rbx

r8 is moved to rsi

and rdx is initialized to 0 by XORING it with itself.

So now lets move onto the second part which is a loop

A close-up of a white background

Description automatically generated

Here we can see the code loops a number of times equal to rcx (our fourth argument)

A value is loaded into rax from rsi and increments the memory pointer.

rax is divided by rbx and the remainder goes to rdx and the quotient is stored in rax

We then store rax in rsi and increment the register

Then we go back to the start of the loop

Once finished, we pop rbx move rd to rax and return

So, the code is performing a division operation repeatedly. I wasn’t immediately sure what this division operation was for. Refreshing on the lods and stos instructions they are usually used for array operations. So, we can immediately see that the result of the division operation is being stored in an array and the divisor is being loaded from another array and the remainder of the final operation is returned. Since the stos and lods instructions are using an operand of 64 bits we are then dividing multiple 64 bit dividend by a 64 bit divisor.

Based on it only returning the final remainder, I would guess that this code is doing division for a number larger than 64 bits.