

## **Summary of work**

I separated the operations for the program into these few steps,

1. Set the bits for sieve[p].length = N
2. Creating first for-loop
3. Creating second for-loop
4. Clearing sieve[k]
5. Division subroutine (UDIV)  
(taken from [CS1021](#))
6. Number to printed ASCII subroutine (PUTU)

Step 1 is completed with the branch

STRWORD,STRHWORD,STRBYTE,STRBITINI and STRBIT.

N was compared to 32,16 and 8.

0xFFFF, 32bits, STRWORD

0xFF, 16bits, STRHWORD

0xF, 8bits, STRBYTE

With each loops, the amount of bits set is taken off N.

When N is lower than 8, the byte was logically shifted left and added one for N times. By then, N bits was set.

Step 2 is completed with the branch FOR0INI and FOR0.

for (int p = 2; p <= N; p++) { if (sieve[p]) {

p was set, and was compared to N.

For each loop than was completed, p is incremented.

To check if sieve[p] is set, we first load the byte.

The address of the byte was calculated from p/8 (1byte = 8bits)

#1 was logically shifted to the position of the bit. (#1,LSL (p % 8))

The byte was then ANDed, and compared if it is equal to 0.

If it is equal, return to FOR0 loop as the bit was not set.

If not, then proceed to if (sieve[p]) statement.

Step 3 and 4 is completed with the branch FOR1INI and FOR1.

k was set, and was compared to N.

For each loop than was completed, k is added by p.

To clear sieve[k], we first find out the position of [k].

The address of the byte was calculated from  $p/8$  (1byte = 8bits)

#1 was logically shifted to the position of the bit. ( $\#1, \text{LSL } (p \% 8)$ )

The byte was loaded, and then BICed by the mask.

The branch loops back to FOR1 and exit when  $k > N$ .

Step 5 is completed with the subroutine UDIV.

UDIV was taken from CS1021.

R0 was taken as the number to be divided, while R1 was taken as the divisor.

R0 would contain the value of the quotient and R1, the remainder when the values are returned.

Step 6 is completed with the subroutine PUTU.

PUTU takes in a number.

Link register was pushed.

Then, the number was passed to UDIV with divisor = #10.

The remainder was then pushed to the stack.

If the quotient is larger than 0, PUTU branch and link back to PUTU.  
(recursive)

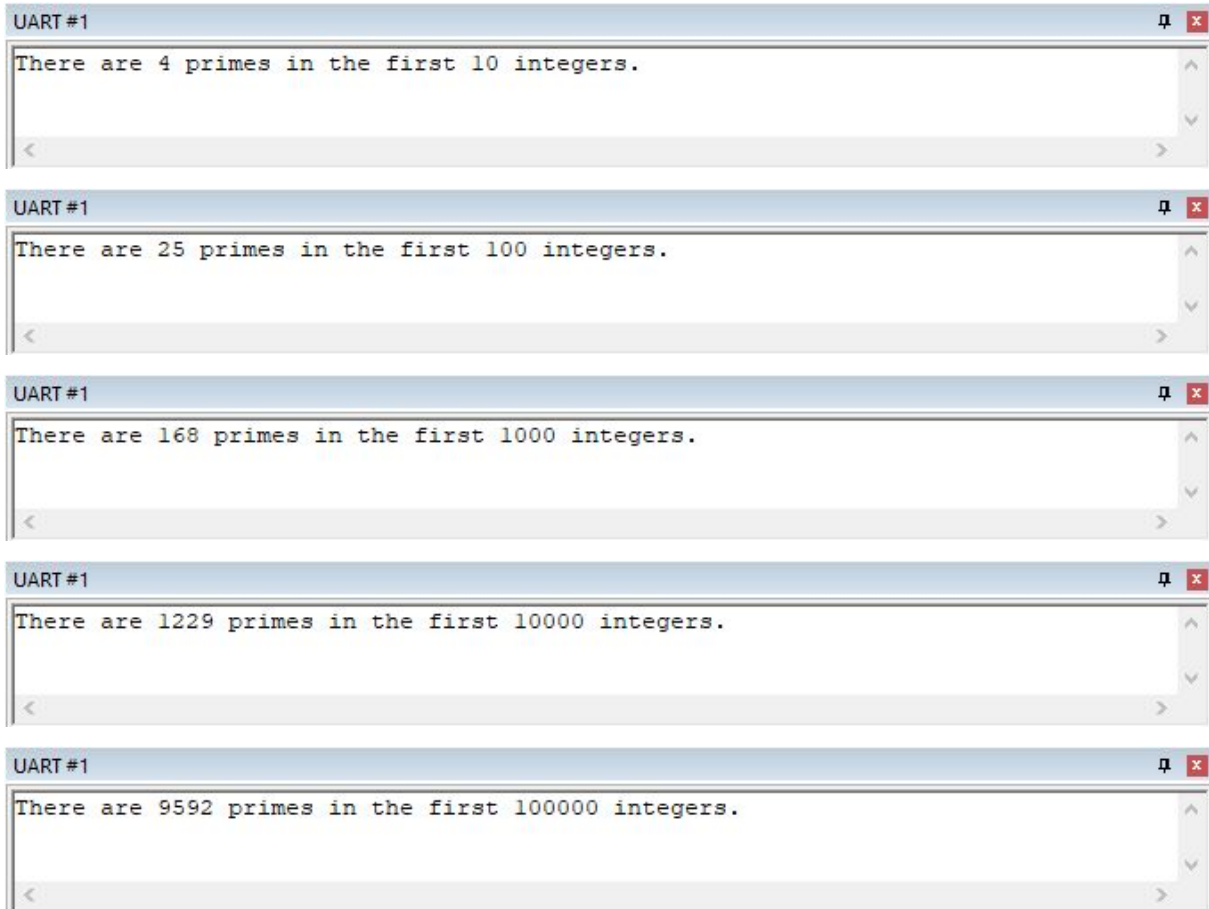
If not, branches to PUTE.

PUTE pops stack into R0.

#0x30 was added to R0 (conversion to ASCII) and PUT was called.

Program counter was popped and the subroutine ends.

Depending on the situation, after PUTE it will branch and link back to the previous instance of PUTU, popping stack and calling PUT till all the PUTU subroutine ended.

**Proof of work**

The image displays five sequential UART #1 terminal windows, each showing the output of a program that counts the number of prime numbers in the first n integers. The windows are arranged vertically, and each has a title bar labeled 'UART #1' with a bell icon and a close button. The output text in each window is as follows:

- UART #1: There are 4 primes in the first 10 integers.
- UART #1: There are 25 primes in the first 100 integers.
- UART #1: There are 168 primes in the first 1000 integers.
- UART #1: There are 1229 primes in the first 10000 integers.
- UART #1: There are 9592 primes in the first 100000 integers.