Floating Point

Project 1

Prepared for

CSE 410 Advanced Topics in Embedded System Design

By

Steven Collins (sac9)

September 15, 2024

Table of Contents

Section 1: Division of Work

I, Steven Collins, wrote the entirety of the floating point project. Some subroutines were used from CSE 379, on which my partner Frank Peretti and I worked together to implement.

Section 2: Program Overview

➔ Program Overview

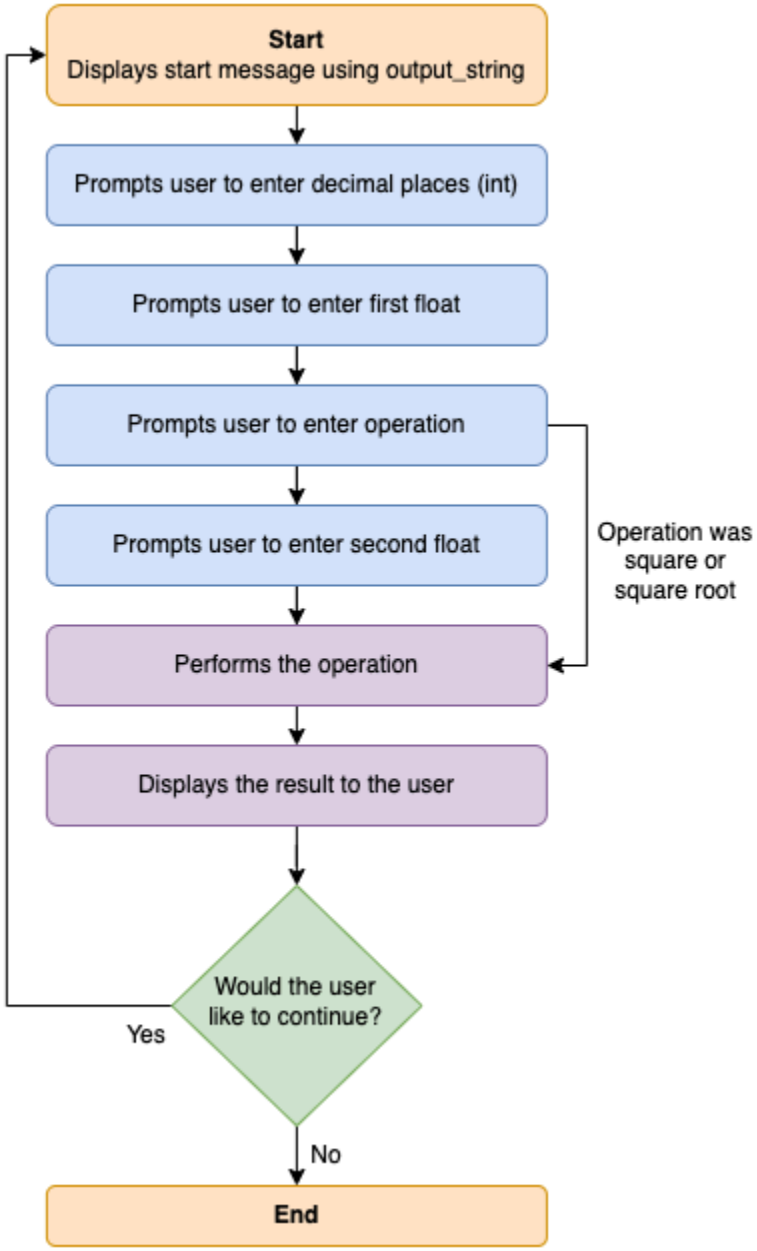
- Enter the maximum number of decimal places the result should have (it should be an integer 1 or larger, and should not exceed the decimal place values of the full result)
- Enter the first number as a float
- Enter the operation to perform
- Enter the second number as a float, if square root or square was not selected.
- If an invalid symbol was entered, the program will restart after prompting the second number
- The result will be displayed
    - Due to inaccuracies in storing floats in base 2, there may be some rounding errors
    - The more complex the number, the higher the chance of inaccuracy
- The user will be asked if they want to continue
    - Enter ‘c’ to rerun the program
    - Enter ‘q’ (or any other key) to quit

➔ Program Summary

This fully functional floating point calculator allows a user to perform six different mathematical operations. First the user will specify the number of decimal places they want the answer rounded to. Then they will enter the first number along side an operation. If the operation is square or square root, the program will display the result and prompt the user to either continue or quit. If it’s one of the other four operations, the user will enter another number, and the the final result and continue prompt will be displayed. The program is made possible through the utilization of UART on a Tiva-C Series, which is initialized through the uart_init subroutine.

➔ High Level Flowchart

- *An overview of the entire program*



Section 3: Subroutine Descriptions

Description of each subroutine's functionality, the arguments it passes through, and what the return values in each register are.

➔ float2string

- Functionality:
  - This subroutine converts a floating point value to a NULL-terminated ASCII string in memory
- Arguments:
  - r0 - address to store the NULL-terminated ASCII string
  - r1 - floating point value to convert to a string
- Return registers:
  - There are no return registers for the functionality of this subroutine.

➔ string2float

- Functionality:
  - This subroutine converts a NULL-terminated ASCII string into a single precision IEEE formatted floating point value.
- Arguments:
  - r0 - base address of NULL-terminated ASCII string
- Return registers:
  - r0 - returns the floating point value (converted from the string)

➔ output_character

- Functionality:
  - This subroutine transmits a character from the UART to PuTTy by storing a character into memory at the data register (0x4000C000).
- Arguments:
  - r0 - contains the character to display
- Return registers:
  - There are no return registers for the functionality of this subroutine.

➔ read_character
- Functionality:
  - This subroutine reads a character by the UART from PuTTy by loading a character into memory at the data register (0x4000C000).
- Arguments:
  - There are no argument registers for the functionality of this subroutine.
- Return registers:
  - r0 - contains the character read from the user

➔ output_string
- Functionality:
  - This subroutine displays a NULL-terminated ASCII string from a location in memory.
- Arguments:
  - r0 - base address of the NULL-terminated string to display
- Return registers:
  - There are no return registers for the functionality of this subroutine.

➔ read_string
- Functionality:
  - This subroutine reads a string from PuTTy and stores it as a NULL-terminated string in memory once the user hits *Enter*.
- Arguments:
  - r0 - the base address to store the user inputted string
- Return registers:
  - There are no return registers for the functionality of this subroutine.

➔ int2string
- Functionality:
  - This subroutine converts an integer to a NULL-terminated ASCII string in memory
- Arguments:
  - r0 - address to store the NULL-terminated ASCII string
  - r1 - integer to convert to a string
- Return registers:
  - There are no return registers for the functionality of this subroutine.

➔ string2int

- Functionality:
  - This subroutine converts a NULL-terminated ASCII string into an integer.
- Arguments:
  - r0 - base address of NULL-terminated ASCII string
- Return registers:
  - r0 - returns the integer (converted from the string)

➔ div_and_mod

- Functionality:
  - Takes two integers and returns the division result, the quotient and remainder. It accepts signed integers, quotients may be negative on result but remainders will always be positive.
- Arguments:
  - r0 - dividend to be divided by the divisor as an integer
  - r1 - divisor to divide the dividend as an integer
- Return registers:
  - r0 - quotient result as an integer
  - r1 - remainder result as an integer

➔ uart_init

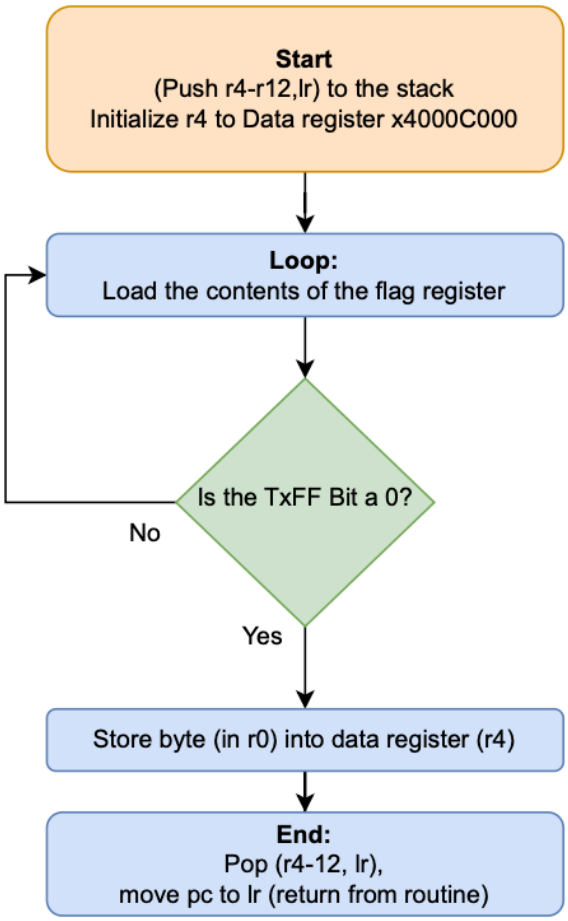
- Functionality:
  - This subroutine initializes the UART for use. This copies the C code used in part one, but written in assembly language.
- Arguments:
  - None
- Return registers:
  - None

➔ project1

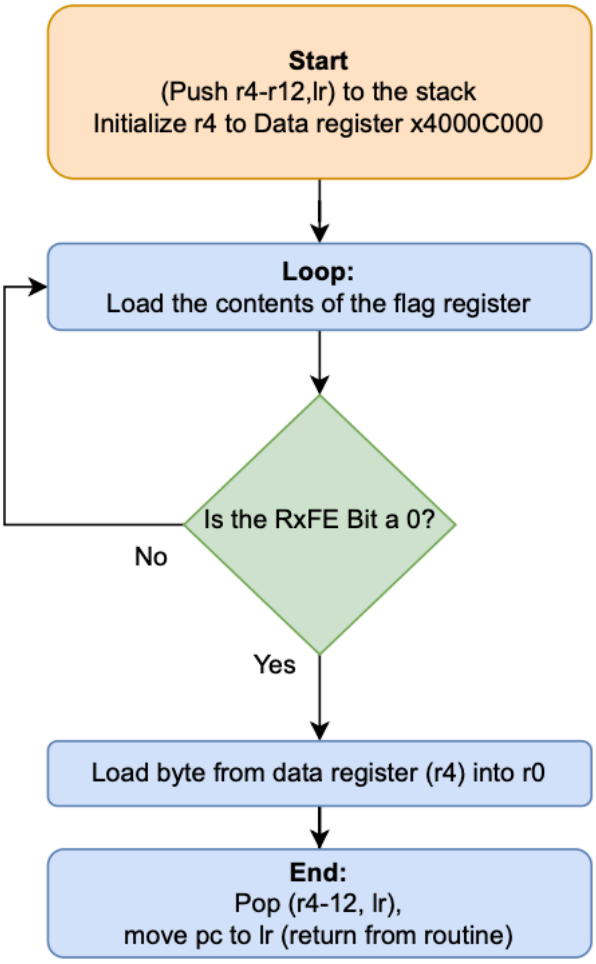
- Functionality:
  - This subroutine controls the overall program and menu which is displayed on PuTTy. It branches and links to all of the subroutines above to display prompts to enter floating point values, decimal lengths, and display the results of a specified computation.
- Arguments:
  - None
- Return registers:
  - None

Section 4: Subroutine Flowcharts

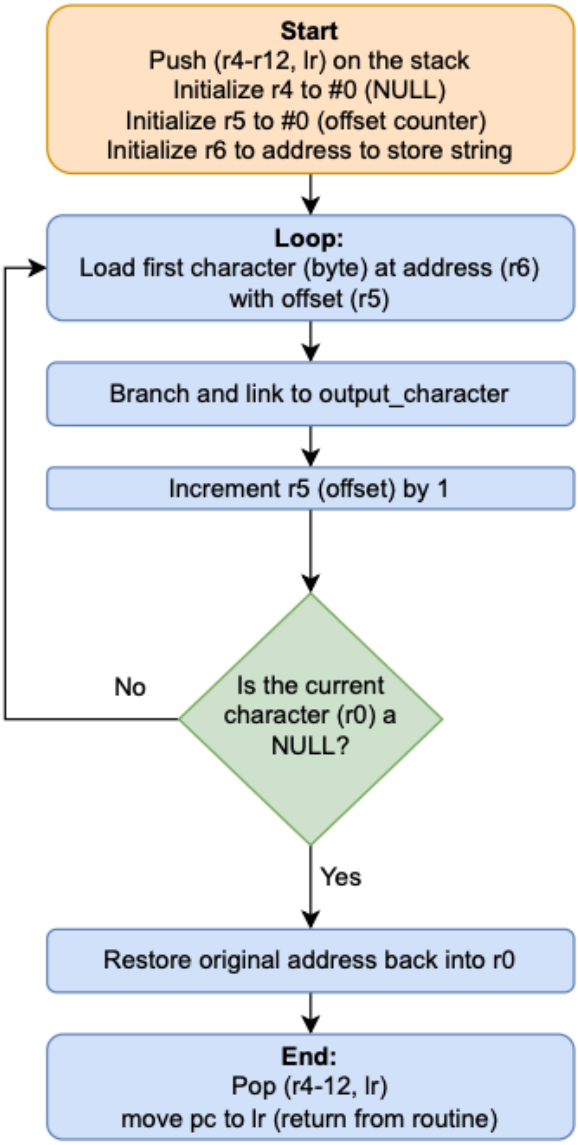
➔ output_character Flowchart



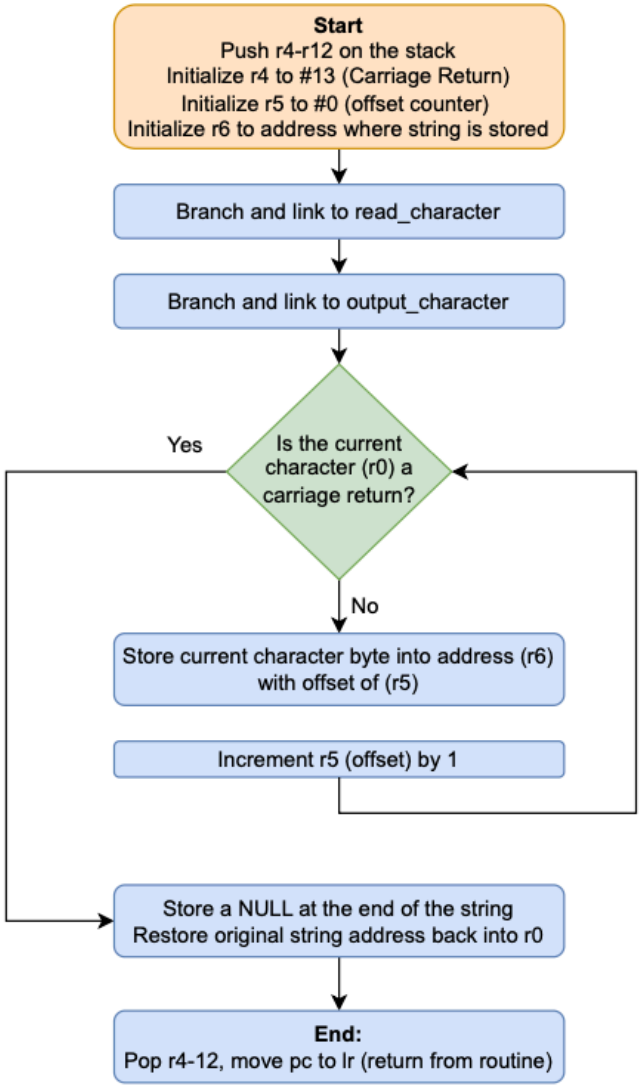
➔ read_character Flowchart



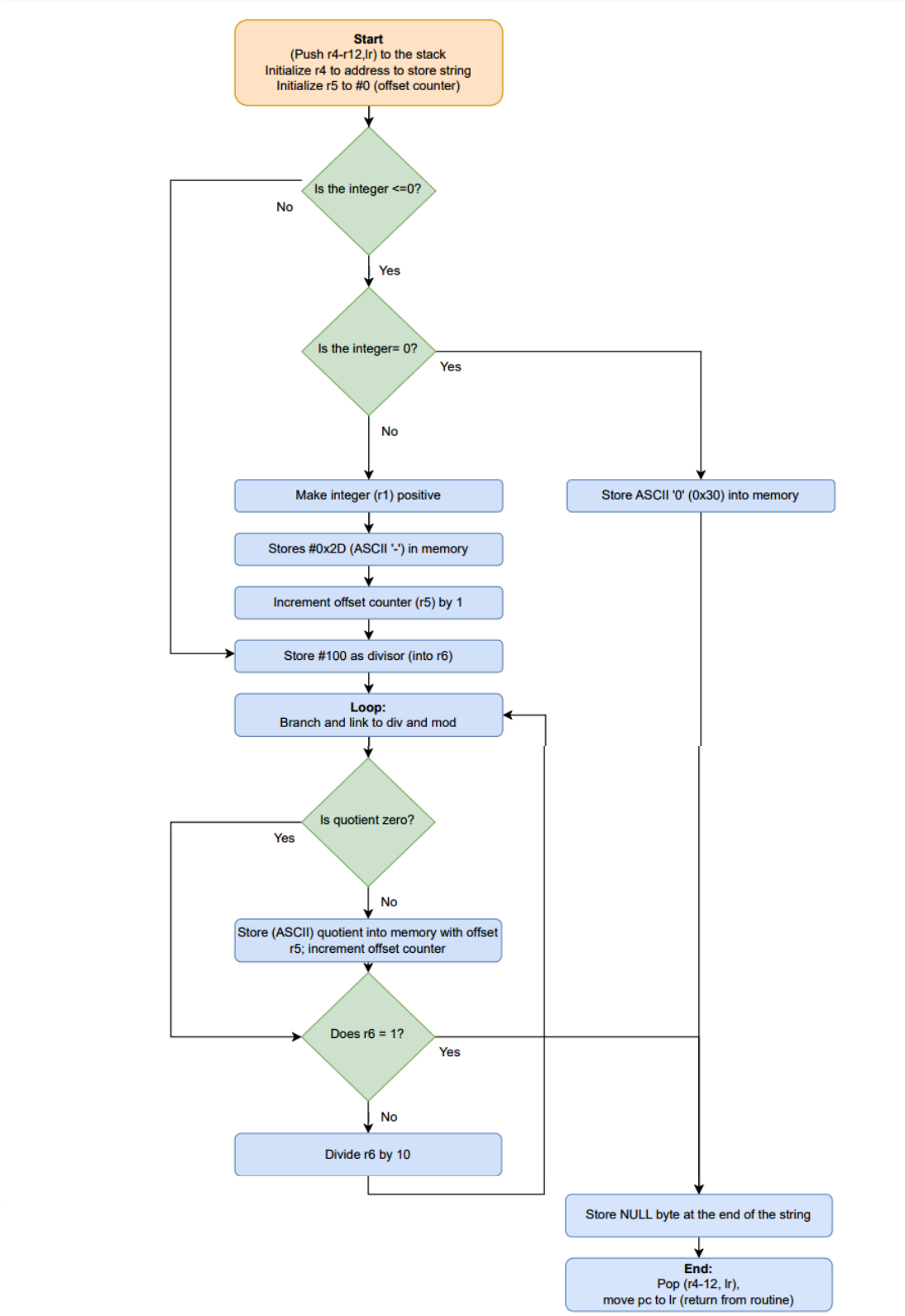
➔ output_string Flowchart



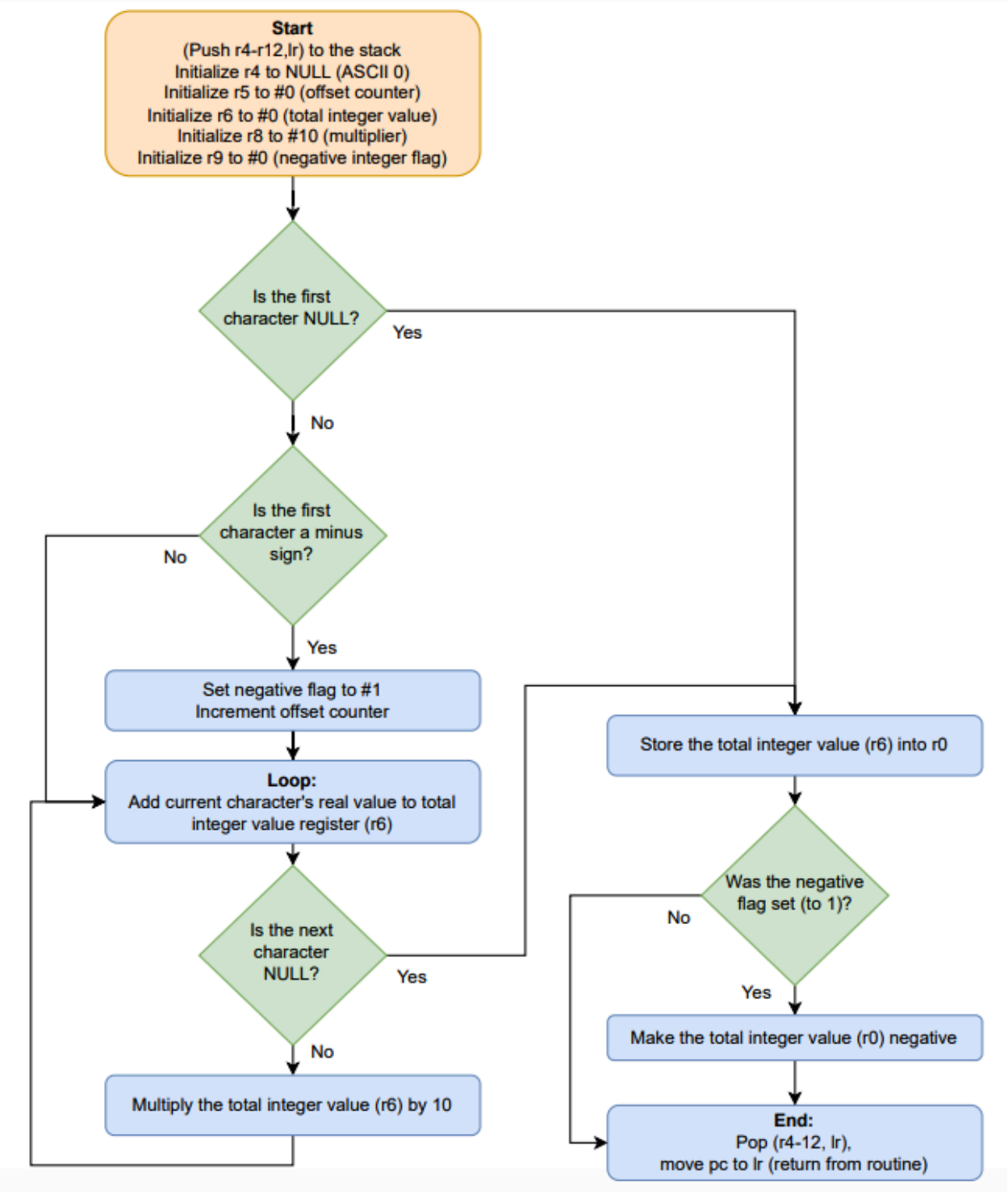
➔ read_string Flowchart



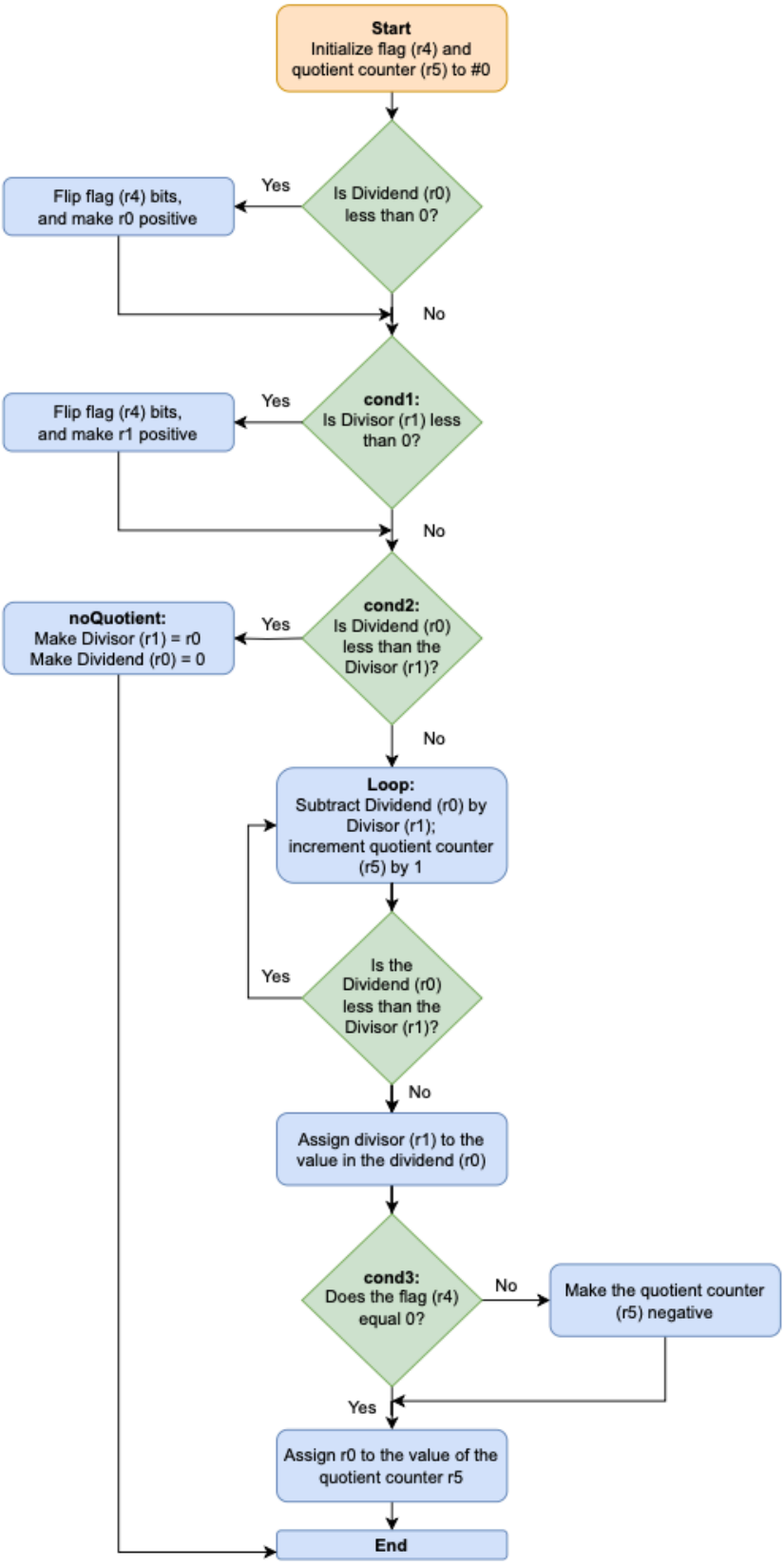
➔ int2string Flowchart



➔ string2int Flowchart



➔ div_and_mod Flowchart



➔ uart_init Flowchart