CSE 379 Lab 3: Serial Communication with UART

Daniel Jin (djin3)
Motaz Khalifa (motazkha)

1) Purpose of the Program

a) Objective

The purpose of this program is to receive two numbers from the user and to output the largest of the two their difference. PuTTy is used for the user to input their two numbers and for the program to display outputs.

b) Debugging Steps

Whenever we ran into problems, we placed breakpoints wherever a subroutine is called. We stepped into every subroutine and looked at what's in memory and in registers. We continued to step into and resume, u
ter value into a second register before calling subroutines on the first register which might have very possibly overwritten the data that we need.

c) How To Run The Program

Go to Start, and type "cmd" and press Enter. Then type "mode" and press Enter. A window will pop up, and at the top of the window, there is a line that says "COMXX." The number, XX, will be important. To open PuTTy, go to Start → All Programs → PuTTy → PuTTy. The PuTTy Configuration window will pop up. Select Serial as the connection type. In the "Serial line" text field, write "COMXX" where XX is the number that you saw in the first step. Change the speed to 115200. To save these settings, type "Tiva-c" in the "Saved Sessions" text field and press "Save," located to the right of it. Then click "Open" at the bottom of the PuTTy Configuration window. Then open Code Composure Studio from the start menu. Now click on debug project, and click on resume. Enter the first number into PuTTY when prompted, click enter, enter the second number then click enter. You should see our program telling you which of your inputs is the largest, and what the difference between the two is.

d) Outside References

For reference, please refer to Texas Instruments' Tiva™ TM4C123GH6PM Microcontroller, Chapter 14. This contains information about UARTDR and UARTFR; the UARTDR and UARTFR are used in *read_character* and *output_character*.

In addition, please refer to ***Introduction to Microprocessor Based Systems Using the ARM Microprocessor***, Kris Schindler, Second Edition, Pearson, 2013, page 41. This contains information about the division algorithm that is implemented in the *div_and_mod* subroutine.

2) Division of Work

Daniel wrote *read_character*, and Motaz wrote *output_character*. The subroutines *read_character* and *output_character* were used to build *read_string* and *output_string* which they both developed together. Daniel wrote *convert_to_int* and Motaz wrote *convert_to_ASCII*. The thought process and the development of the other subroutines were mutually contributed by Daniel and Motaz.

3) Logic

a) Summary of Flowchart

First, the prompt to enter the first number is outputted to PuTTy. Then, as the user inputs their digit(s), the program stores these values in memory. Then, the prompt to enter the second number is outputted to PuTTy, and, again, the digit(s) are stored in memory. The program then finds and outputs the largest of the two numbers. Finally, the program computes the difference and outputs the difference. The program then loops back to the beginning, allowing the user to re-run the program without exiting.

b) Summary of Subroutines

i) *uart_int*

This subroutine initializes the board. The assembly code is written based on the *serial_init()* function in the C wrapper.

ii) *read_character*

When the user presses a key, the character is stored in UARTDR. This subroutine receives the character and stores it in the register r0.

iii) *output_character*

This subroutine stores whatever character/data is in r0 into UARTDR. By doing so, the character is outputted into PuTTy.

iv) *read_string*

When the user presses a key, the character is stored in UARTDR. This subroutine receives the character and stores it into memory in r4. It then proceeds to store any character pressed later after it in memory. If the key pressed is the Enter key, then a null character is stored instead and it terminates.

v) *output_string*

This subroutines starts at the address held in r4, and outputs the character at that address, traversing through r4. When the character in r4 is a null character, this subroutine stops.

vi) *output_newline*

This subroutine simply outputs a new line into PuTTy, by outputting the character represented by ASCII value 10 and 13, which are new line and carriage return, respectively.

vii) *convert_to_int*

When a key is pressed by the user, the computer reads it as the key's ASCII representation. This subroutine converts the key's ASCII representation into its integer representation by subtracting 48 and weighting in terms of 100's 10's and 1's.

viii) *Calculations*

This subroutine takes the two numbers, entered by the user, as integer representations, and finds the biggest of the two, and the difference. The biggest number and the difference are then moved into different registers for future use.

ix) *convert_to_ASCII*

This subroutine takes an integer stored in register, and starts storing it backwards into memory address by dividing by 10 three times and setting the remainder as the last value in memory slot.

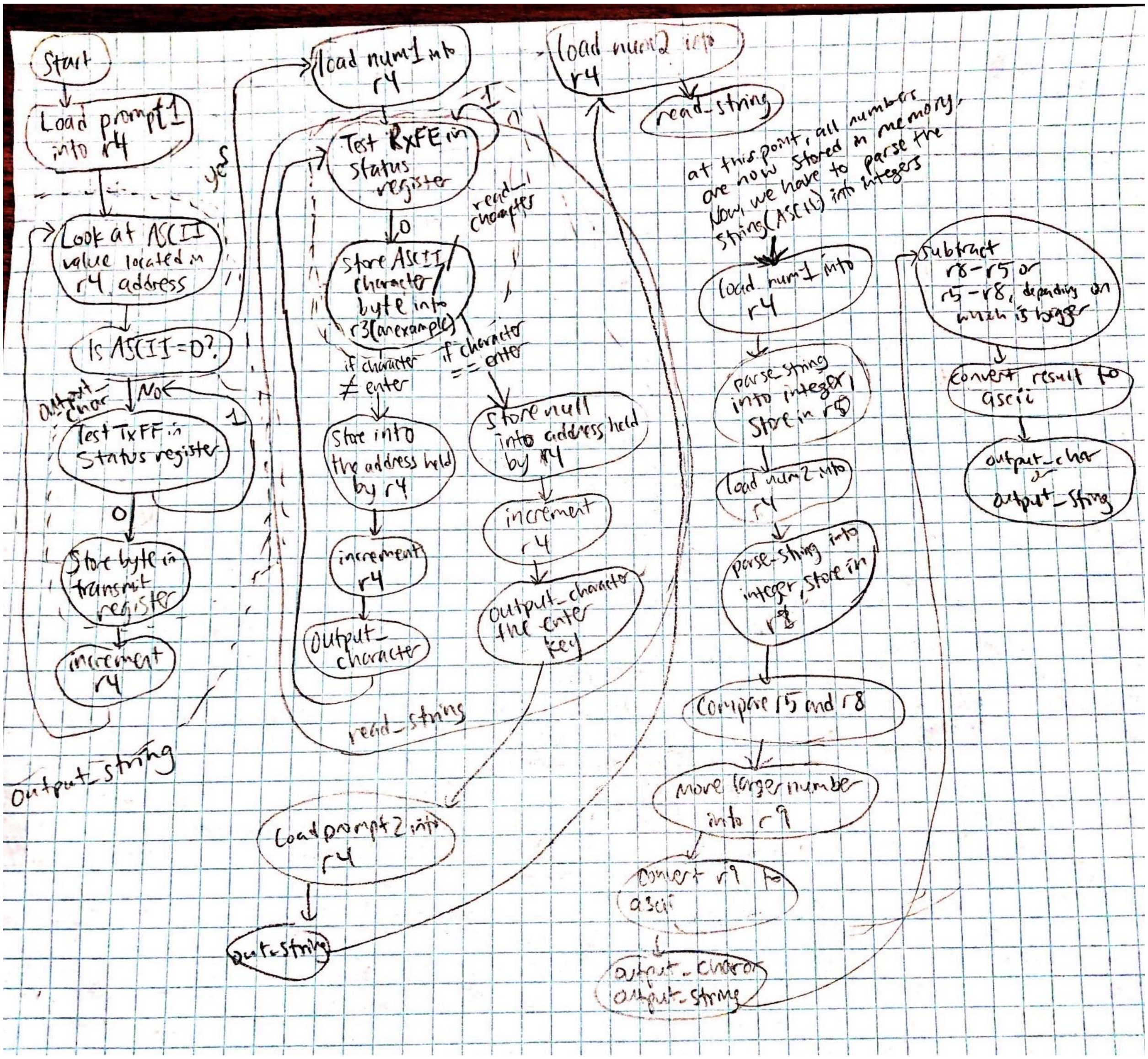
x) *div_and_mod*

Subroutine used from lab2, used to determine the quotient and remainder of an unsigned division.

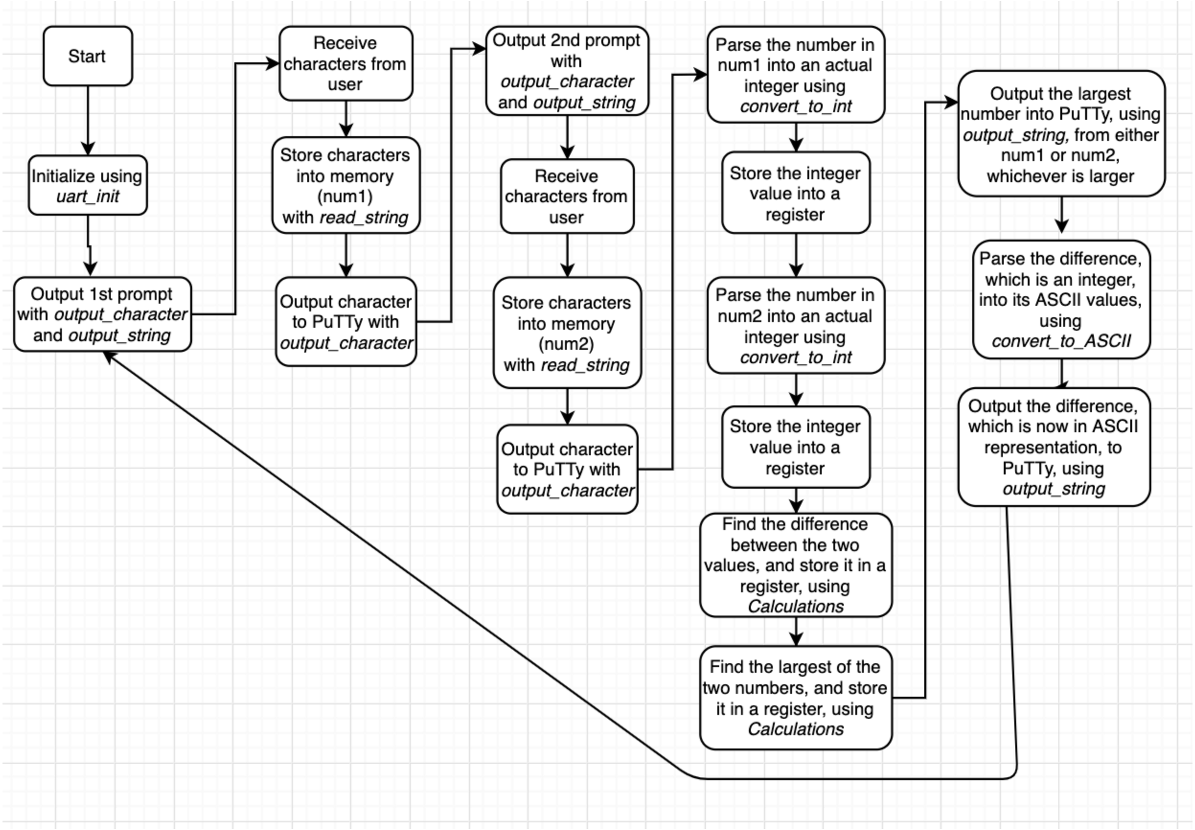
4) Flowcharts

All flowcharts are generated in draw.io.

a) Rough Draft

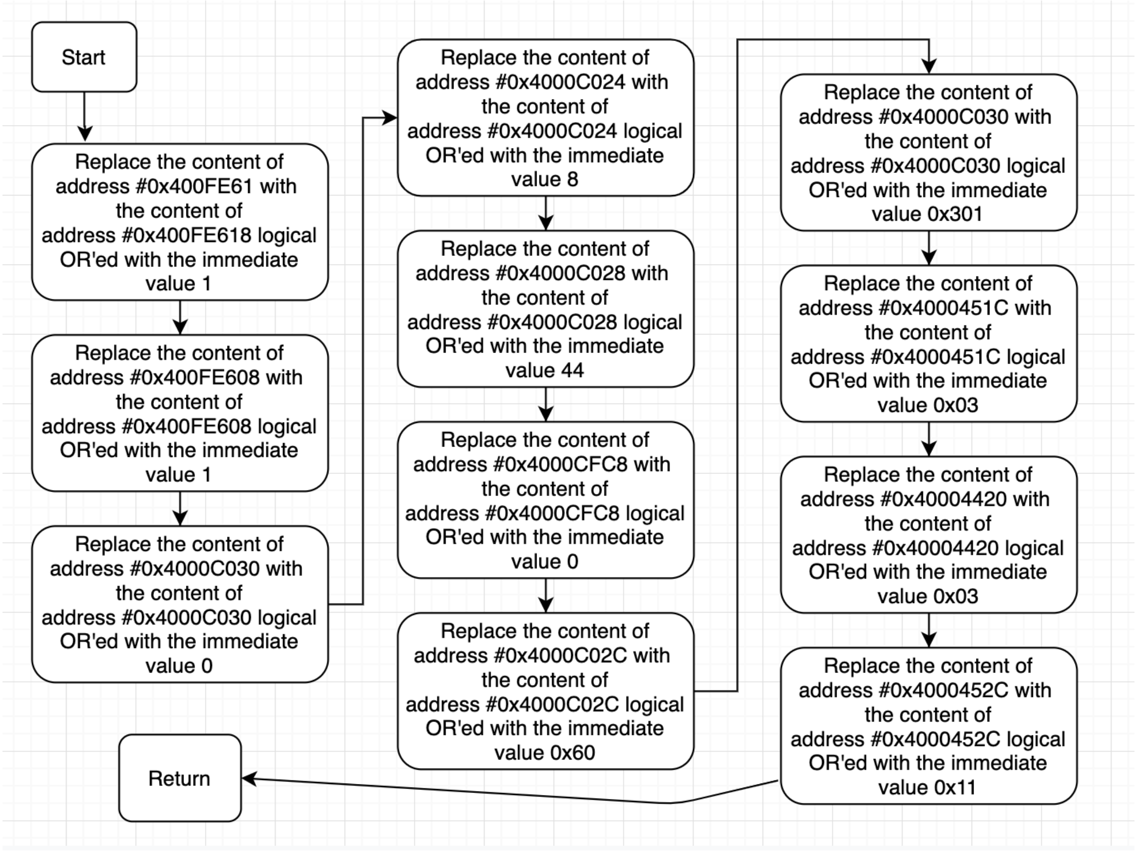

Start
Load prompt1 into r4
yes
Look at ASCII value located in r4 address
Is ASCII = 0?
No
output_char
Test TxFF in Status register
1
0
Store byte in transmit register
increment r4
output_string
load num1 into r4
1
Test RxFE in Status register
0
read_character
Store ASCII character/byte into r3(an example)
if character ≠ enter
if character == enter
Store into the address held by r4
increment r4
Output_character
Store null into address held by r4
increment r4
output_character the enter key
read_string
Load prompt2 into r4
out_string
load num2 into r4
read_string
at this point, all numbers are now stored in memory. Now, we have to parse the string(ASCII) into integers
load num1 into r4
parse_string into integer, Store in r8
load num2 into r4
parse_string into integer, Store in r8
Compare r5 and r8
Move larger number into r9
Convert r9 to ascii
output_char or output_string
Subtract r8-r5 or r5-r8, depending on which is bigger
Convert result to ascii
output_char or output_string

b) Main Program Final Flowchart

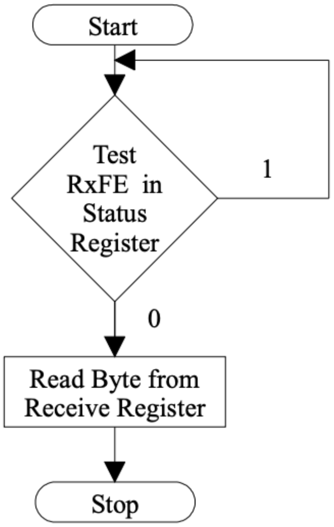

Start
Initialize using *uart_init*
Output 1st prompt with *output_character* and *output_string*
Receive characters from user
Store characters into memory (num1) with *read_string*
Output character to PuTTy with *output_character*
Output 2nd prompt with *output_character* and *output_string*
Receive characters from user
Store characters into memory (num2) with *read_string*
Output character to PuTTy with *output_character*
Parse the number in num1 into an actual integer using *convert_to_int*
Store the integer value into a register
Parse the number in num2 into an actual integer using *convert_to_int*
Store the integer value into a register
Find the difference between the two values, and store it in a register, using *Calculations*
Find the largest of the two numbers, and store it in a register, using *Calculations*
Output the largest number into PuTTy, using *output_string,* from either num1 or num2, whichever is larger
Parse the difference, which is an integer, into its ASCII values, using *convert_to_ASCII*
Output the difference, which is now in ASCII representation, to PuTTy, using *output_string*

c) Subroutine Flowcharts

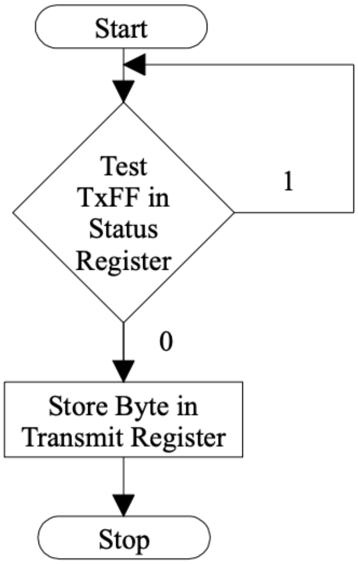
i) uart_init


Start
Replace the content of address #0x400FE61 with the content of address #0x400FE618 logical OR'ed with the immediate value 1
Replace the content of address #0x400FE608 with the content of address #0x400FE608 logical OR'ed with the immediate value 1
Replace the content of address #0x4000C030 with the content of address #0x4000C030 logical OR'ed with the immediate value 0
Replace the content of address #0x4000C024 with the content of address #0x4000C024 logical OR'ed with the immediate value 8
Replace the content of address #0x4000C028 with the content of address #0x4000C028 logical OR'ed with the immediate value 44
Replace the content of address #0x4000CFC8 with the content of address #0x4000CFC8 logical OR'ed with the immediate value 0
Replace the content of address #0x4000C02C with the content of address #0x4000C02C logical OR'ed with the immediate value 0x60
Replace the content of address #0x4000C030 with the content of address #0x4000C030 logical OR'ed with the immediate value 0x301
Replace the content of address #0x4000451C with the content of address #0x4000451C logical OR'ed with the immediate value 0x03
Replace the content of address #0x40004420 with the content of address #0x40004420 logical OR'ed with the immediate value 0x03
Replace the content of address #0x4000452C with the content of address #0x4000452C logical OR'ed with the immediate value 0x11
Return

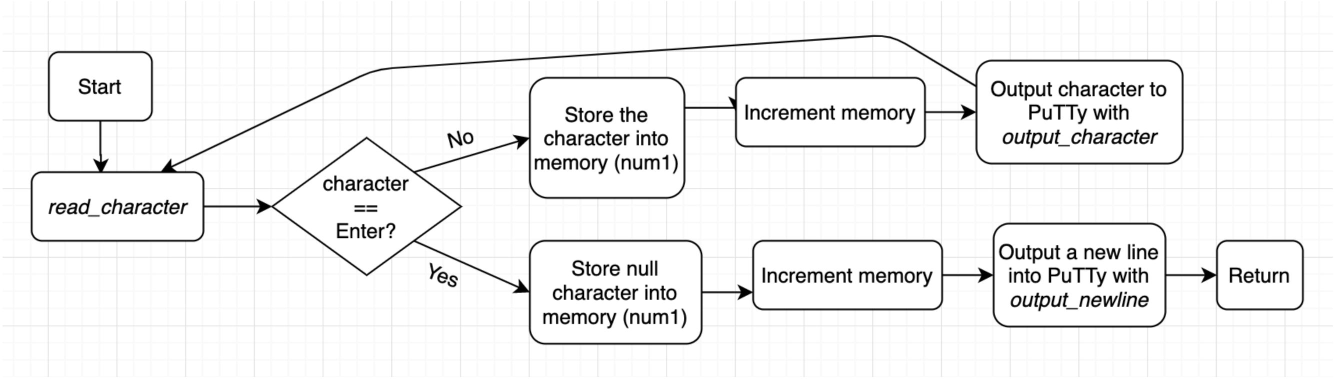
ii) read_character



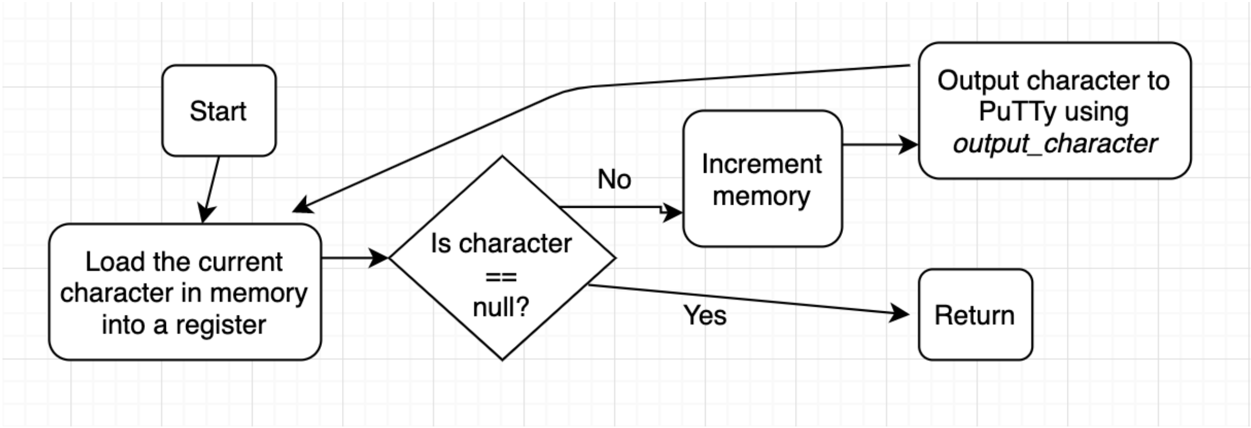
iii) output_character



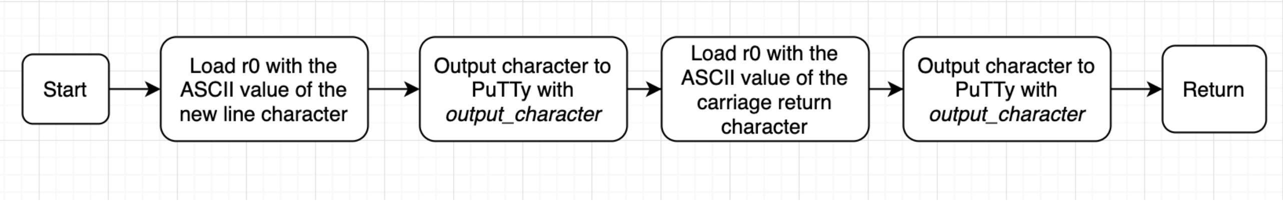
iv) read_string


Start
read_character
character == Enter?
No
Store the character into memory (num1)
Increment memory
Output character to PuTTy with output_character
Yes
Store null character into memory (num1)
Increment memory
Output a new line into PuTTy with output_newline
Return

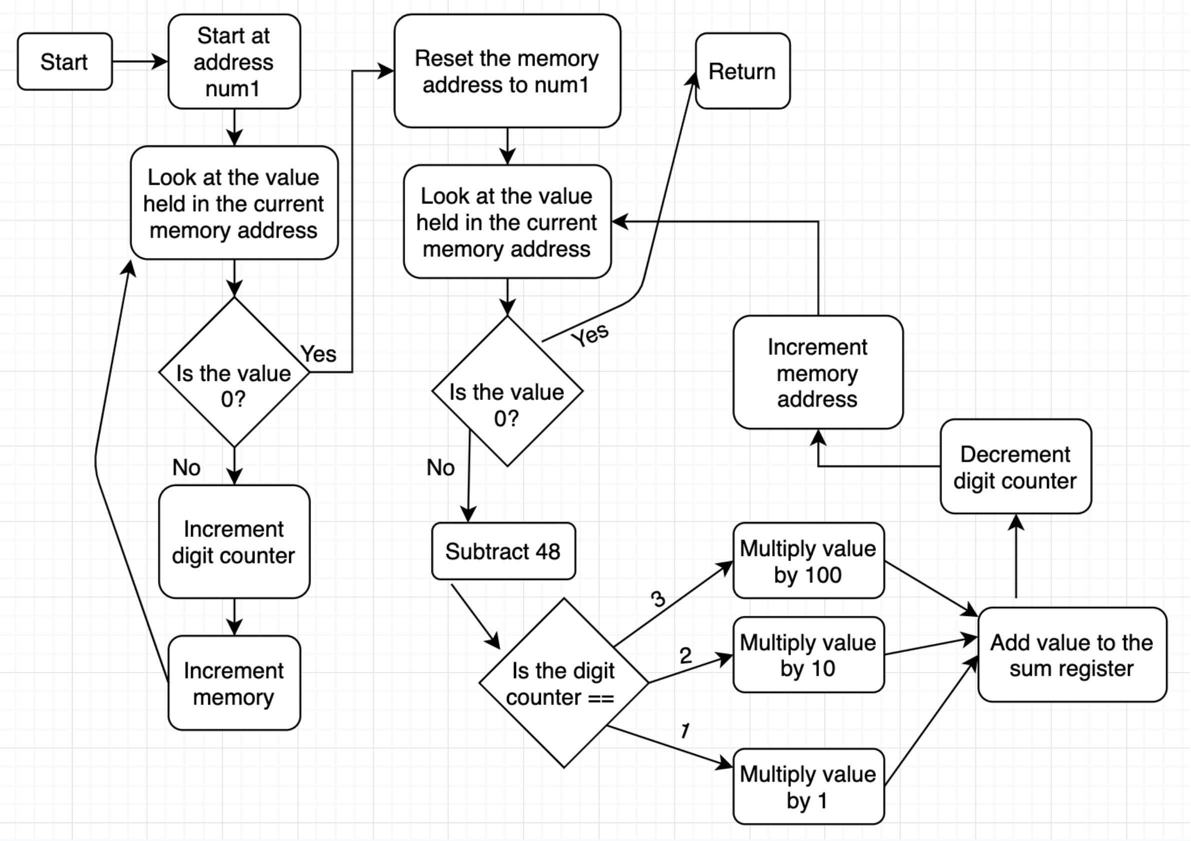
v) output_string


Start
Load the current character in memory into a register
Is character == null?
No
Increment memory
Output character to PuTTy using output_character
Yes
Return

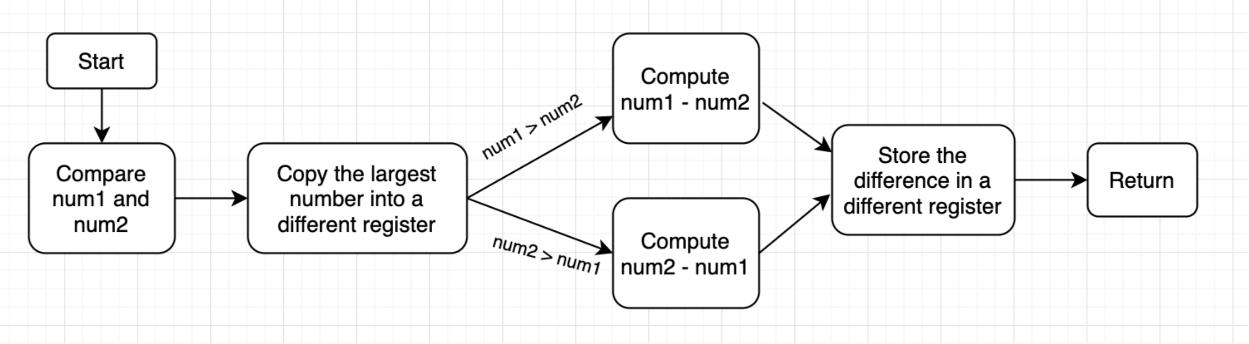
vi) output_newline


Start
Load r0 with the ASCII value of the new line character
Output character to PuTTy with *output_character*
Load r0 with the ASCII value of the carriage return character
Output character to PuTTy with *output_character*
Return

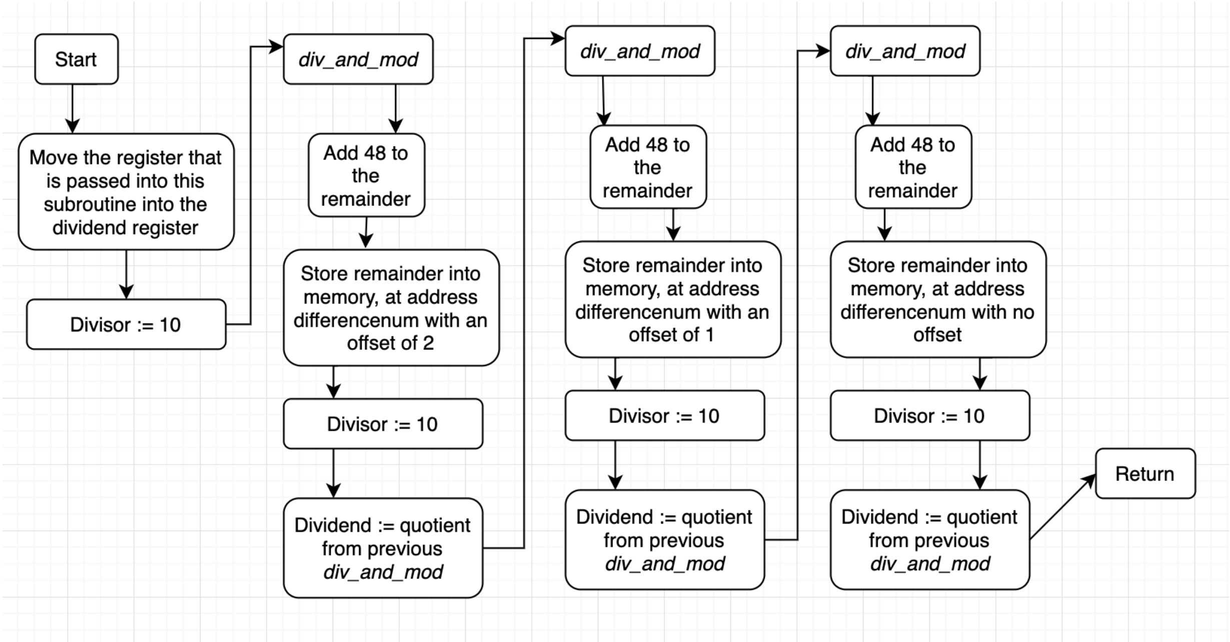
vii) convert_to_int



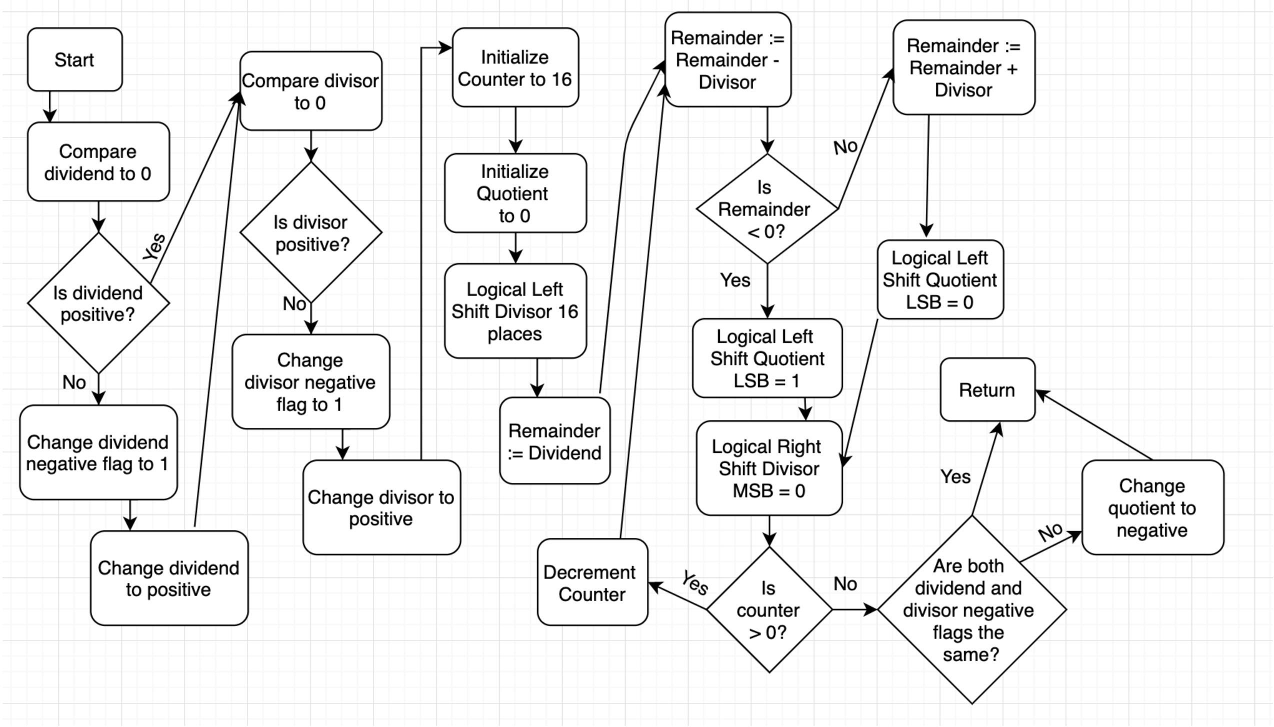
viii) Calculations


Start
Compare num1 and num2
Copy the largest number into a different register
num1 > num2
num2 > num1
Compute num1 - num2
Compute num2 - num1
Store the difference in a different register
Return

ix) convert_to_ASCII


Start
Move the register that is passed into this subroutine into the dividend register
Divisor := 10
div_and_mod
Add 48 to the remainder
Store remainder into memory, at address differencenum with an offset of 2
Divisor := 10
Dividend := quotient from previous *div_and_mod*
div_and_mod
Add 48 to the remainder
Store remainder into memory, at address differencenum with an offset of 1
Divisor := 10
Dividend := quotient from previous *div_and_mod*
div_and_mod
Add 48 to the remainder
Store remainder into memory, at address differencenum with no offset
Divisor := 10
Dividend := quotient from previous *div_and_mod*
Return

x) div_and_mod


Start
Compare dividend to 0
Is dividend positive?
Yes
No
Change dividend negative flag to 1
Change dividend to positive
Compare divisor to 0
Is divisor positive?
No
Change divisor negative flag to 1
Change divisor to positive
Initialize Counter to 16
Initialize Quotient to 0
Logical Left Shift Divisor 16 places
Remainder := Dividend
Remainder := Remainder - Divisor
Is Remainder < 0?
No
Yes
Remainder := Remainder + Divisor
Logical Left Shift Quotient LSB = 0
Logical Left Shift Quotient LSB = 1
Logical Right Shift Divisor MSB = 0
Is counter > 0?
Yes
Decrement Counter
No
Are both dividend and divisor negative flags the same?
Yes
Return
No
Change quotient to negative