CSE 379 Lab 4: General Purpose I/O

Daniel Jin (djin3)
Motaz Khalifa (motazkha)

---

1) Purpose of the Program

a) Objective

The objective of this program is to allow the user to choose between four options, each of which has its own function. One option asks the user to input a 4 digit binary value, and lights up the LEDs on the board corresponding to the binary input. Another option asks the user to use the push buttons, and outputs to PuTTy the hexadecimal representation of the push buttons; the push buttons represents a 4 digit binary input, i.e. the left most push button is the MSB and the right most push button is the LSB. Another option is to ask the user to press a key (0 - 9) on the keypad, and PuTTy will display the key pressed. The last option is to ask the user to choose a color to be displayed on the colored LED on the board.

b) Debugging Steps

Since this program involves a lot of loading and storing to registers, we made sure to set breakpoints at any lines that deal with direction, digital, and data registers, if we ran into problems. We monitored the values in certain memory addresses, and carefully stepped over line by line. Such memory addresses we monitored are Port A's, Port B's, Port D's, and Port F's direction, digital, and data registers. Other steps we took were putting breakpoints at lines where we believe the program is not working as intended. Using this method, we can slowly but surely narrow down the exact line where the bug exists.

c) How To Run The Program

Open PuTTy using the proper COM number and speed. In Code Composure Studios, click Debug → Resume. In PuTTy, you will see a menu listing out the 4 options, and a 5th option to exit the program. After selecting an option, another line of instruction will tell you — the user — how to use that particular function. After that function finishes, the program will bring you back to the main menu, where you can again select an option.

2) Division of Work

Daniel wrote the subroutines, *illuminate_LEDs*, *illuminate_RGB_LED*, and *read_from_push_btns*. Subroutines used to implement these 3 subroutines were also written by Daniel. These subroutines are *initiate_push_btns*, *initiate_LED*, *initiate_RGB_LED*, *turn_blue_on*, *turn_blue_off*, *turn_red_on*, *turn_red_off*, *turn_green_on*, *turn_green_off*. Motaz wrote the main menu, and *read_from_keypad*, which is the most complex and comprehensive subroutine of the four.

3) Logic

a) Summary of Main Program Flowchart

The main menu is outputted to PuTTy. The next subroutine called is dependent on what option the user selects. After every subroutine finishes its functions, the program then returns to the main menu, outputting it to PuTTy. At this point, the user can select from any of the options again.

b) Summary of Subroutine Flowcharts

i) illuminate_LEDs

This subroutine initializes the correct port and properly configures the port's direction and digital register, using another subroutine, *initiate_LED*. Then, this subroutine reads the user input via *read_string*, and sets the port's data register corresponding to the user input; effectively, this lights up the LEDs wherever the input has a 1.

ii) initiate_LED

This subroutine enables Port B in the clock register, sets pins 3, 2, 1, and 0 in Port B's direction register to output, sets pins 3, 2, 1, and 0 in Port B's digital register to digital.

iii) read_from_push_btns

This subroutine reads the data register for which push button was pressed. The subroutine then takes that value, parses it into its hexadecimal representation, and outputs it to PuTTy.

iv) initiate_push_btns

This subroutine enables Port D in the clock register, sets pins 3, 2, 1, and 0 in Port B's direction register to input, sets pins 3, 2, 1, and 0 in Port D's digital register to digital.

v) initiate_read_keypad

This subroutine starts by enabling port D in GPIO, setting direction for 0,1,2,3 as output (1), and digital enabling pins 0,1,2,3. It then proceeds to initialize all data inside to 1 to supply power to the keypad. It then enables port A in GPIO, sets direction for pins 2,3,4,5 as input and digitally enables pins 2,3,4,5 and also initializes all data to 0.

vi) read_keypad

This subroutine loops through until data inside port A is no longer zero. By comparing the value inside the data register it could then decide which column (A Pin) is being used. By shutting down port D pins one by one consecutively and checking whether the data in A was erased or not, it could know which pin in port D is supplying power for the button press. By storing the column number and row number together in one value the subroutine then proceeds to print the ASCII equivalent of the key equivalent of the column/row number using output_character.

vii) illuminate_RGB_LED

This subroutines reads the color choice inputted into PuTTy by the user, and lights up certain colors and keeps other colors off in order to make the LED light up the desired color for the user.

viii) initiate_RGB_LED

This subroutine enables Port F in the clock register, sets pins 3, 2, and 1 in Port F's direction register to output, sets pins 3, 2, 1 in Port F's digital register to digital.

ix) turn_blue_on, turn_blue_off, turn_red_on, turn_red_off, turn_green_on, turn_green_off

These subroutines either turn on or off the corresponding color. For blue, it makes pin 2 in Port F's data register to a 1 to light up blue, and 0 to turn off. For red, it's pin 1, and for green, it's 3. To make white, yellow, and purple, a specific combination of red, green, and blue are lit.

Subroutines from Lab 3:

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.

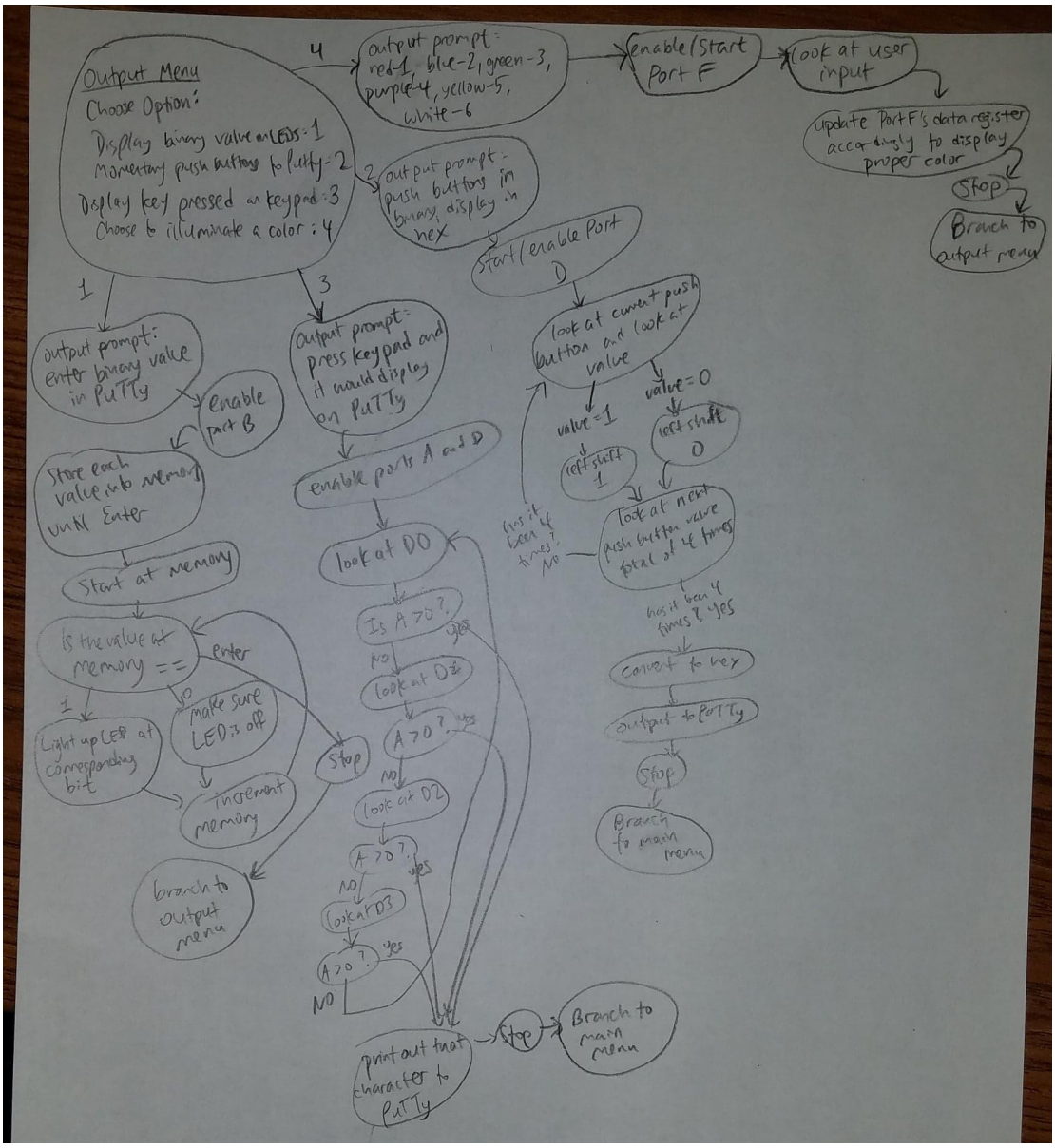
vi) output_newline

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.

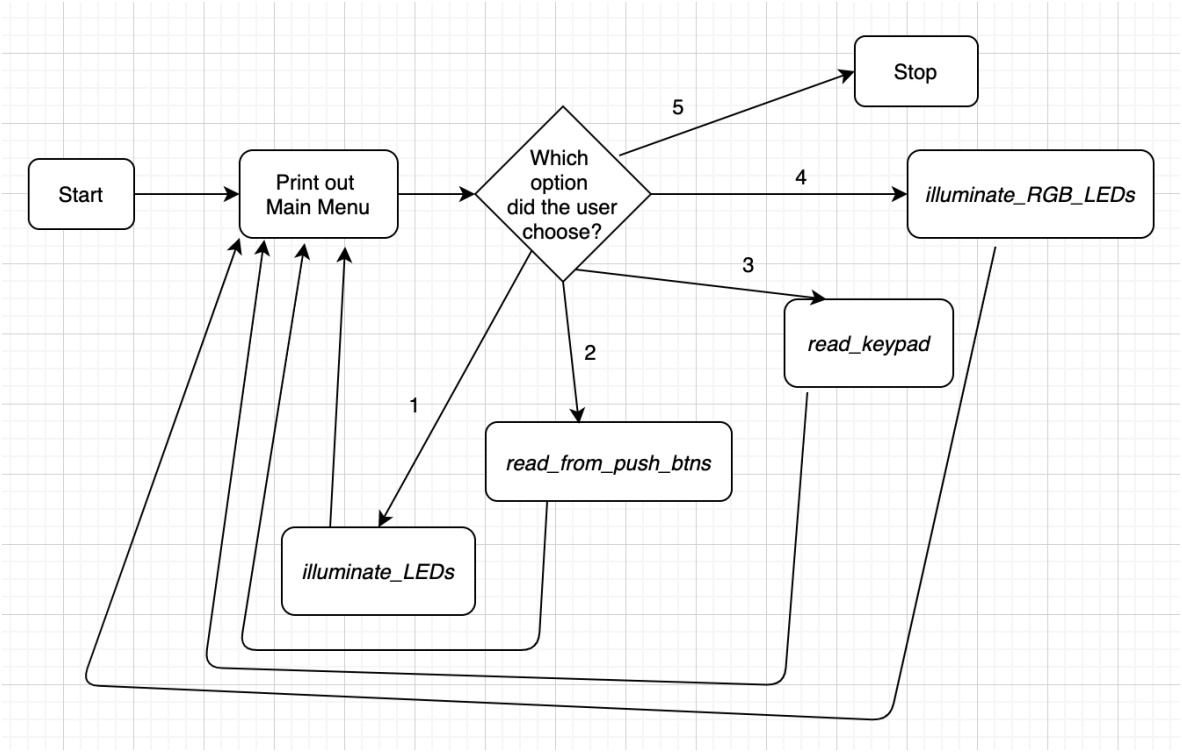
4) Flowcharts

All flowcharts are generated by draw.io.

a) Rough Draft

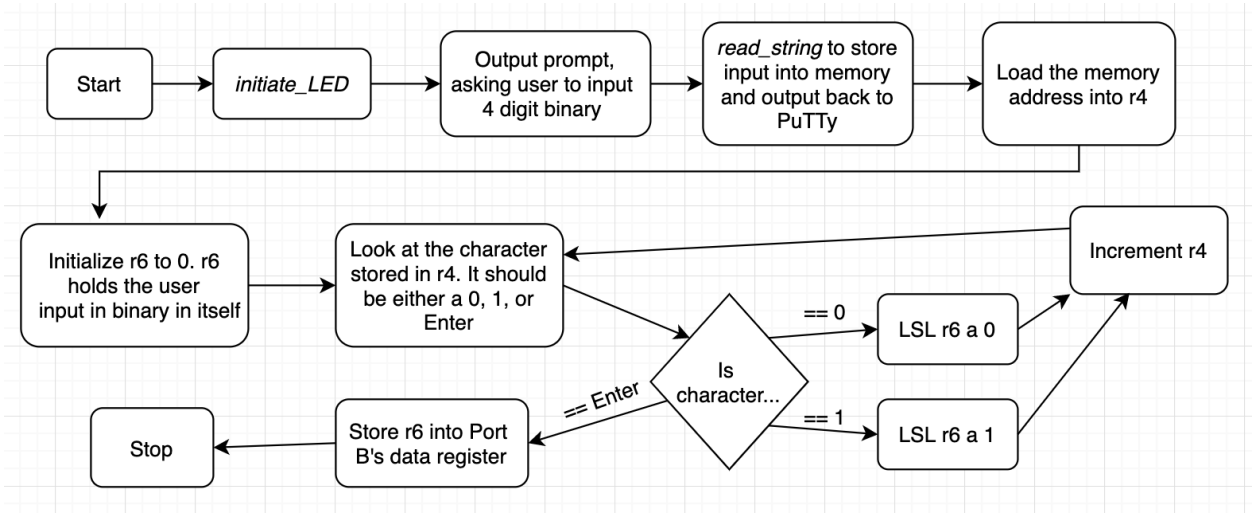


b) Main Program Flowchart

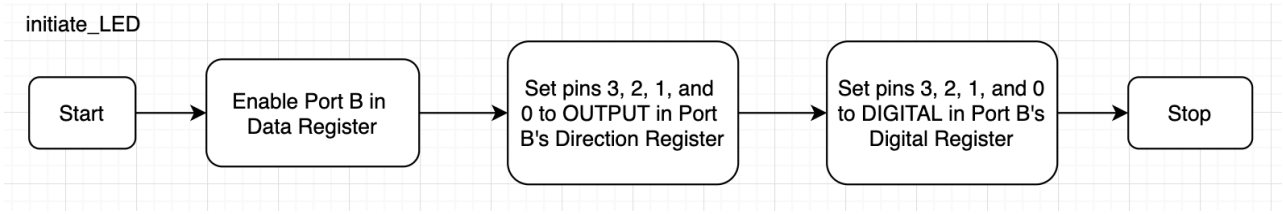


c) Subroutine Flowcharts

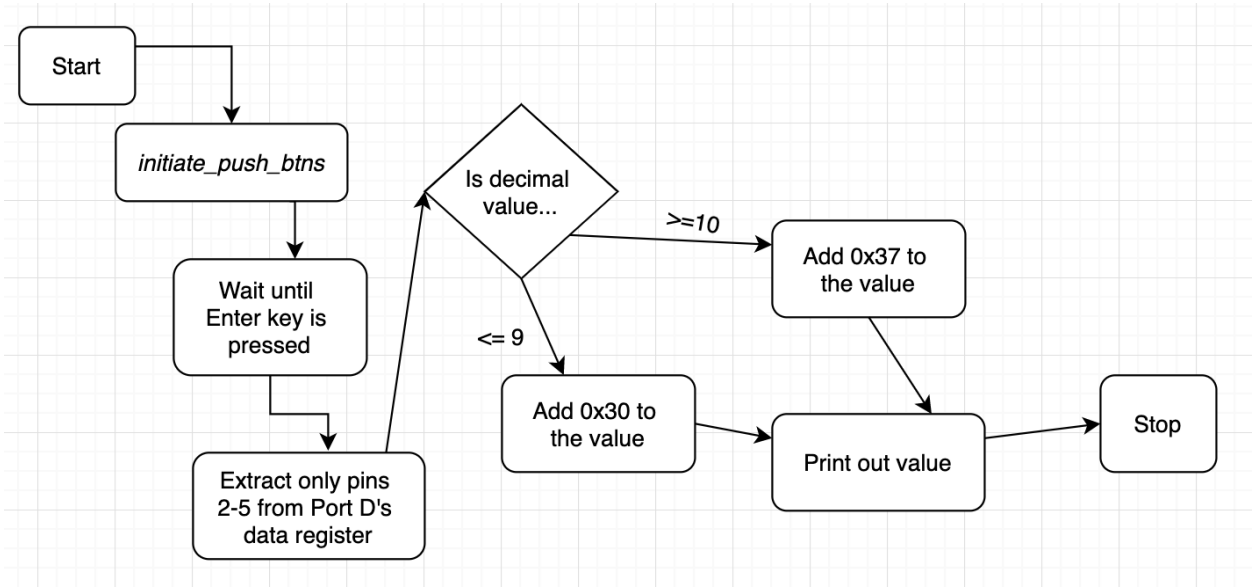
i) illuminate_LEDs



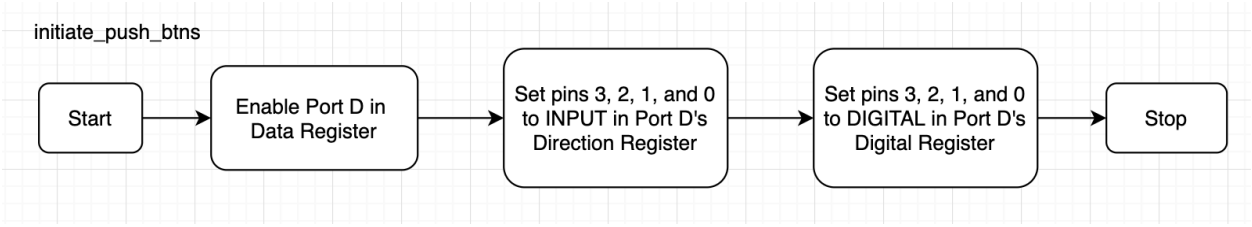
ii) initiate_LED



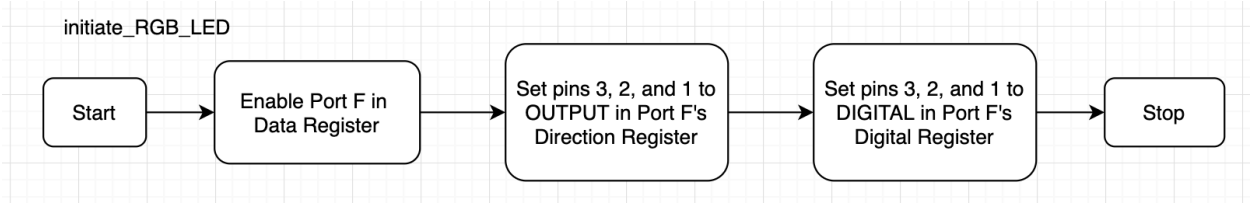
iii) read_from_push_btns



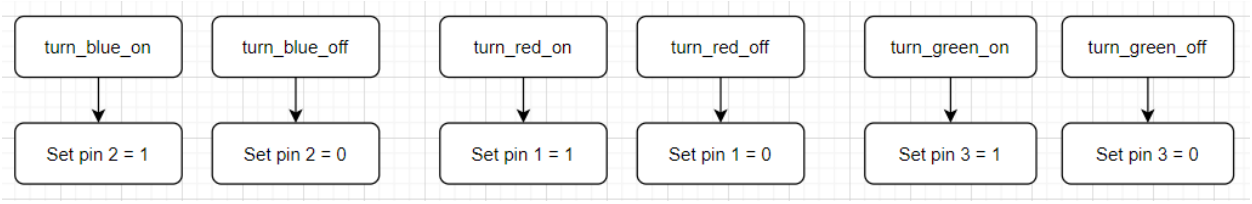
iv) initiate_push_btns



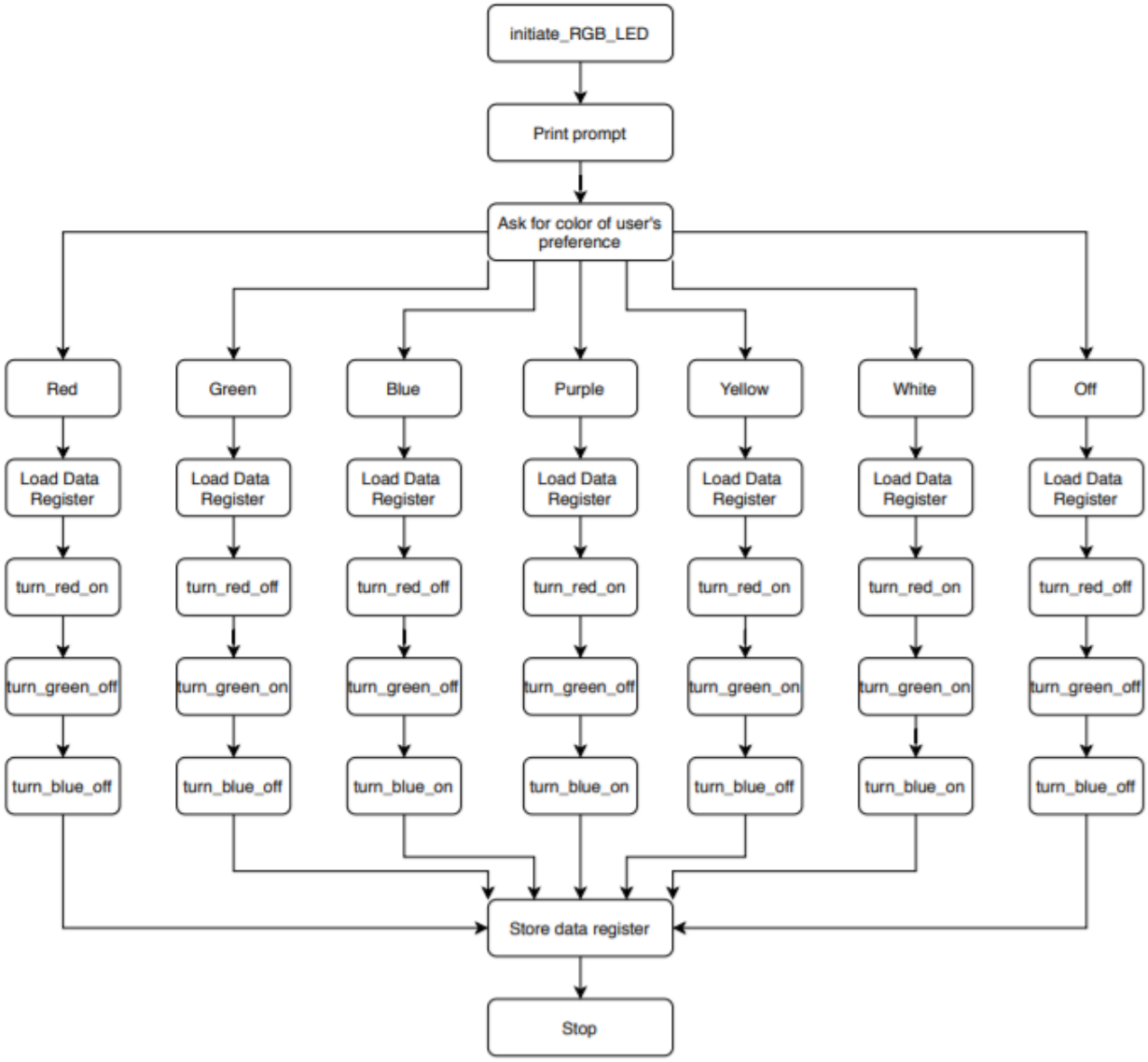
v) initiate_RGB_LED



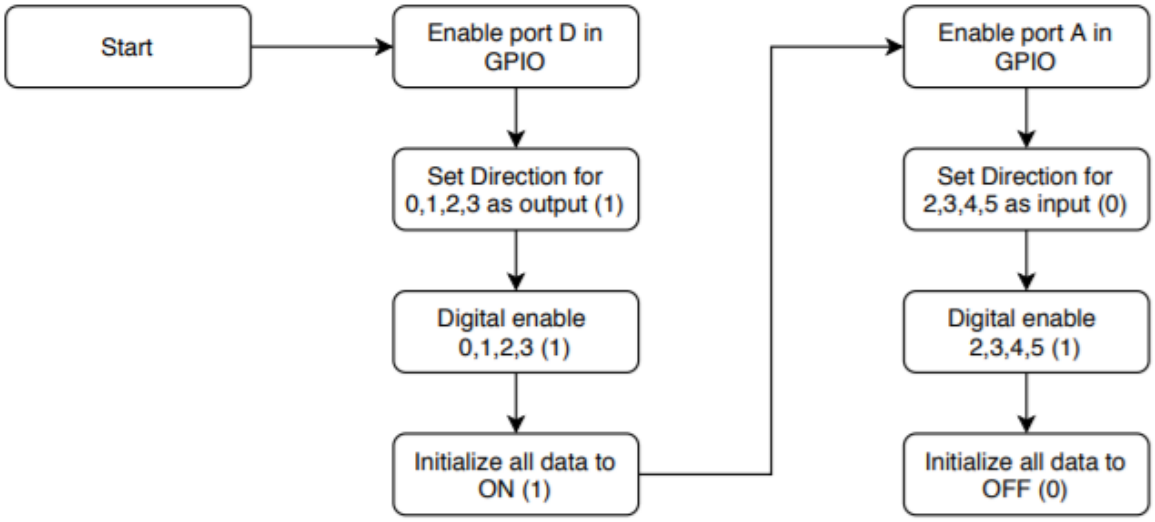
vi) turn_blue_on, turn_blue_off, turn_red_on, turn_red_off, turn_green_on, turn_green_off



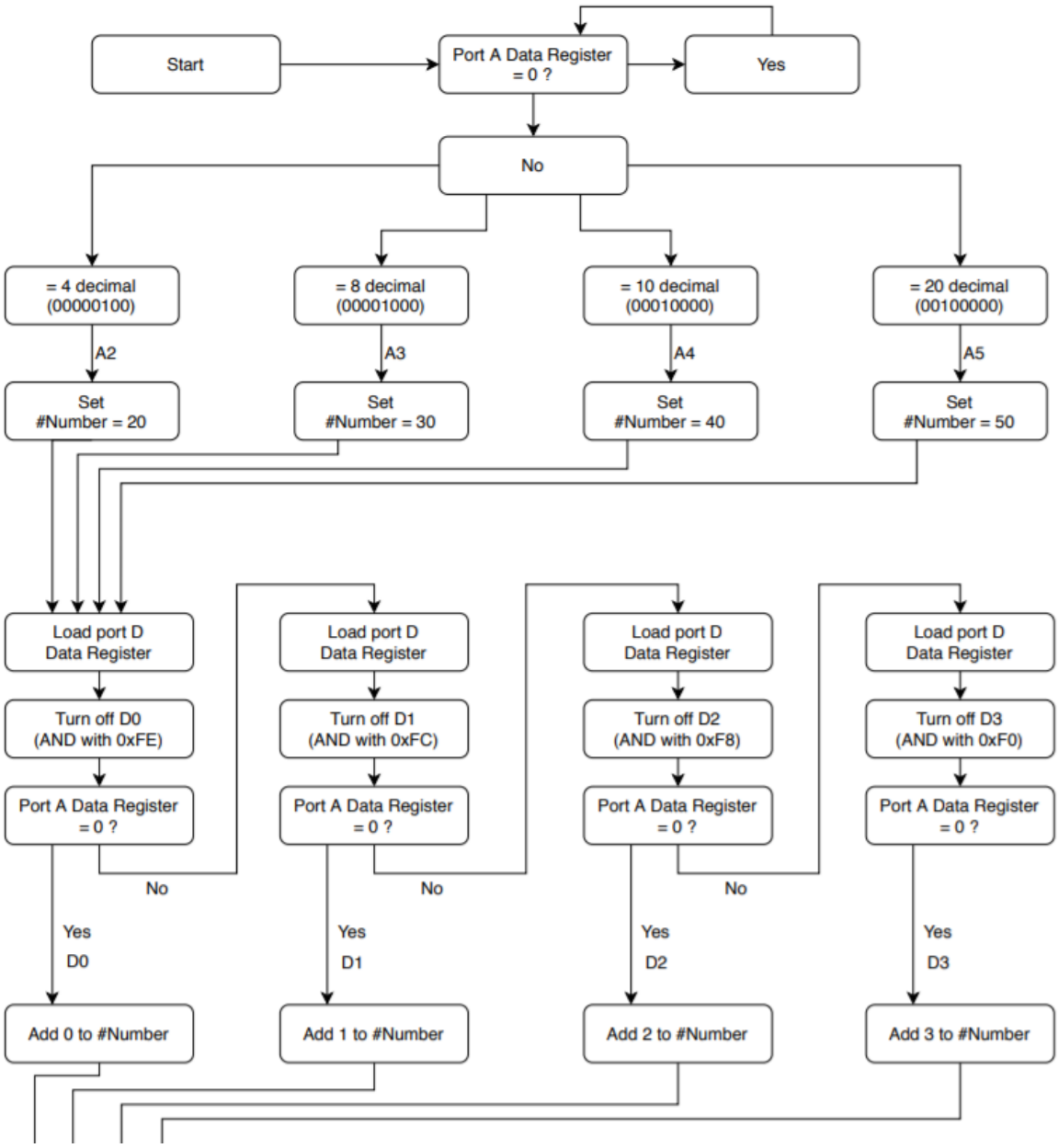
vii) illuminate_RGB_LED

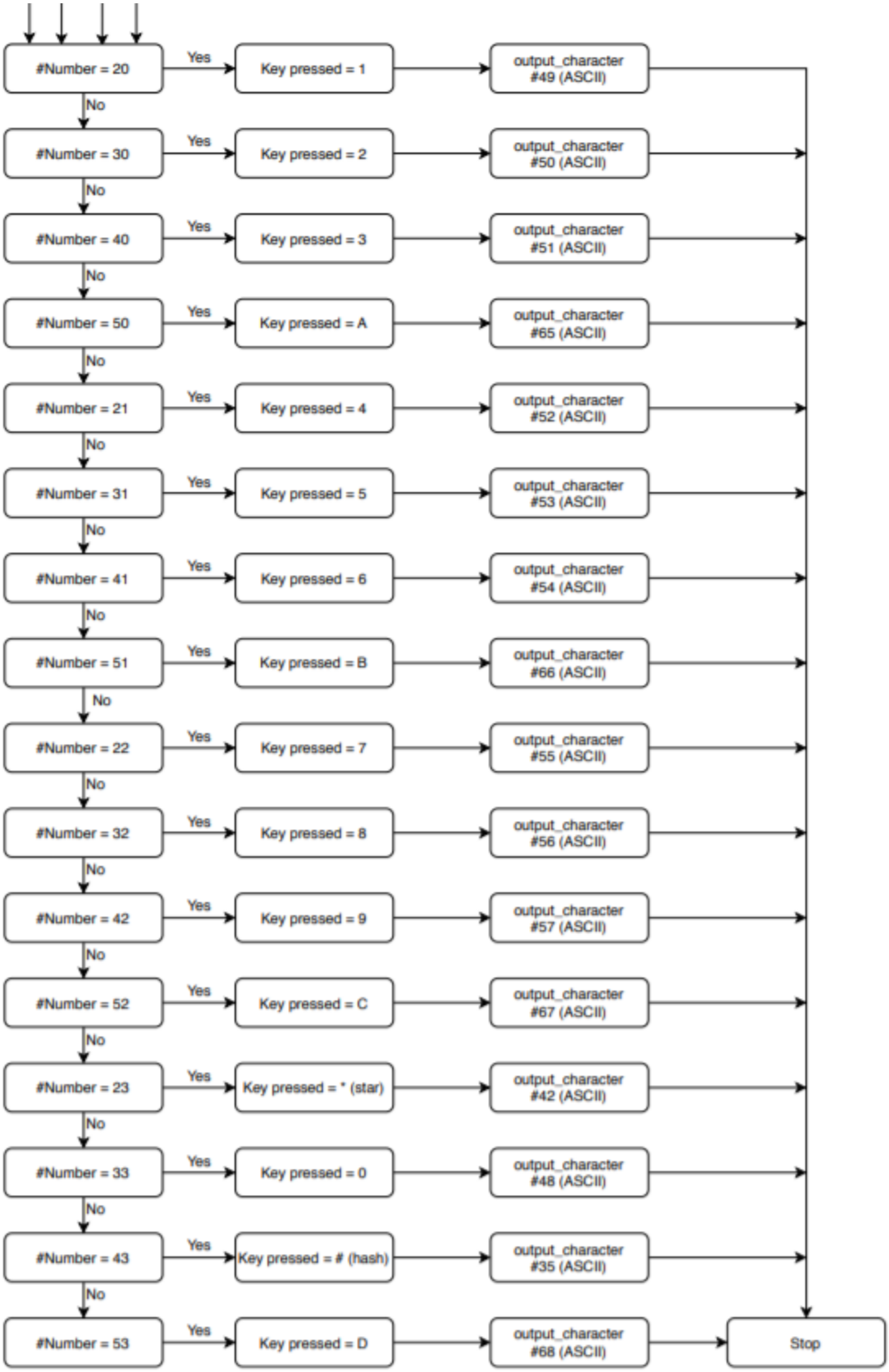


viii) initiate_read_keypad

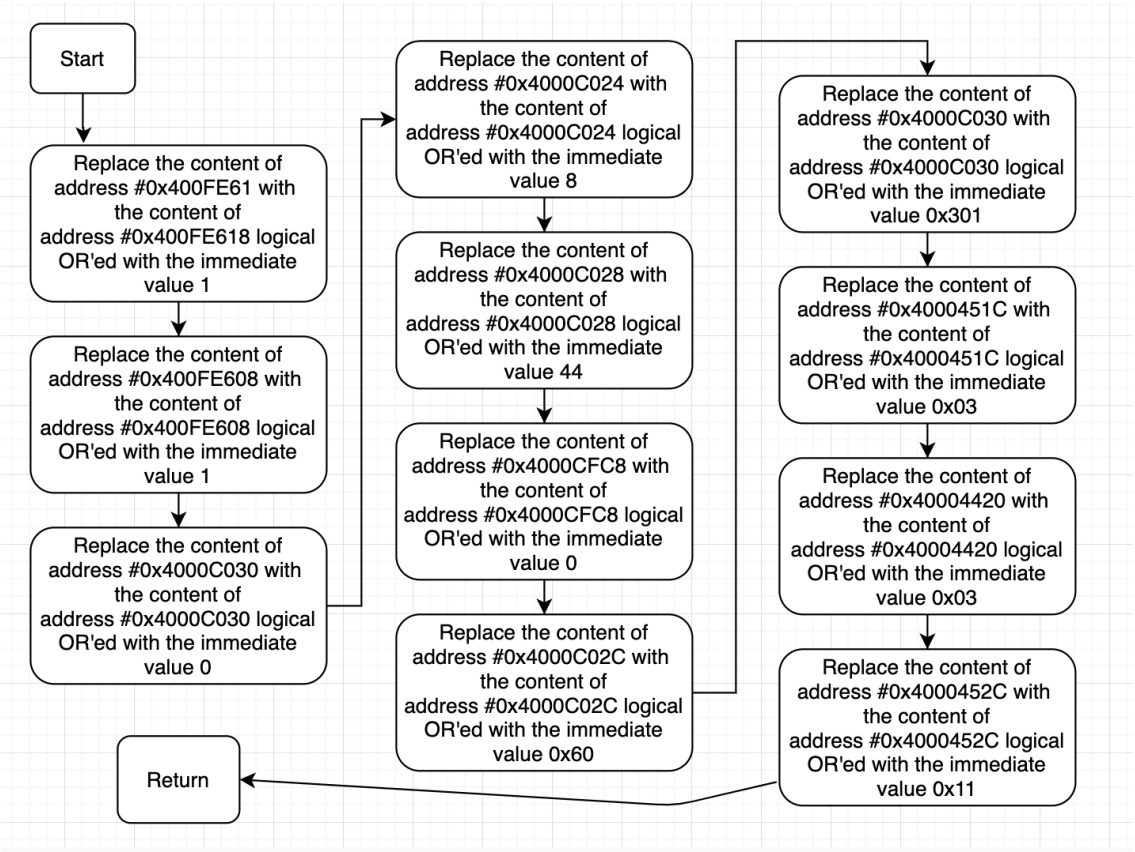


ix) read_keypad

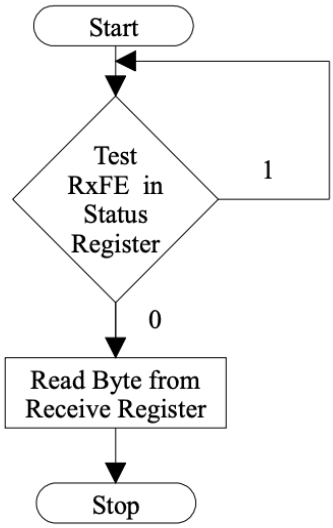



#Number = 20
Yes
Key pressed = 1
output_character #49 (ASCII)
No
#Number = 30
Yes
Key pressed = 2
output_character #50 (ASCII)
No
#Number = 40
Yes
Key pressed = 3
output_character #51 (ASCII)
No
#Number = 50
Yes
Key pressed = A
output_character #65 (ASCII)
No
#Number = 21
Yes
Key pressed = 4
output_character #52 (ASCII)
No
#Number = 31
Yes
Key pressed = 5
output_character #53 (ASCII)
No
#Number = 41
Yes
Key pressed = 6
output_character #54 (ASCII)
No
#Number = 51
Yes
Key pressed = B
output_character #66 (ASCII)
No
#Number = 22
Yes
Key pressed = 7
output_character #55 (ASCII)
No
#Number = 32
Yes
Key pressed = 8
output_character #56 (ASCII)
No
#Number = 42
Yes
Key pressed = 9
output_character #57 (ASCII)
No
#Number = 52
Yes
Key pressed = C
output_character #67 (ASCII)
No
#Number = 23
Yes
Key pressed = * (star)
output_character #42 (ASCII)
No
#Number = 33
Yes
Key pressed = 0
output_character #48 (ASCII)
No
#Number = 43
Yes
Key pressed = # (hash)
output_character #35 (ASCII)
No
#Number = 53
Yes
Key pressed = D
output_character #68 (ASCII)
Stop

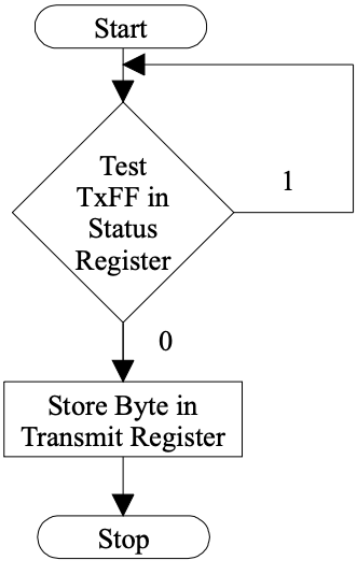
x) uart_init



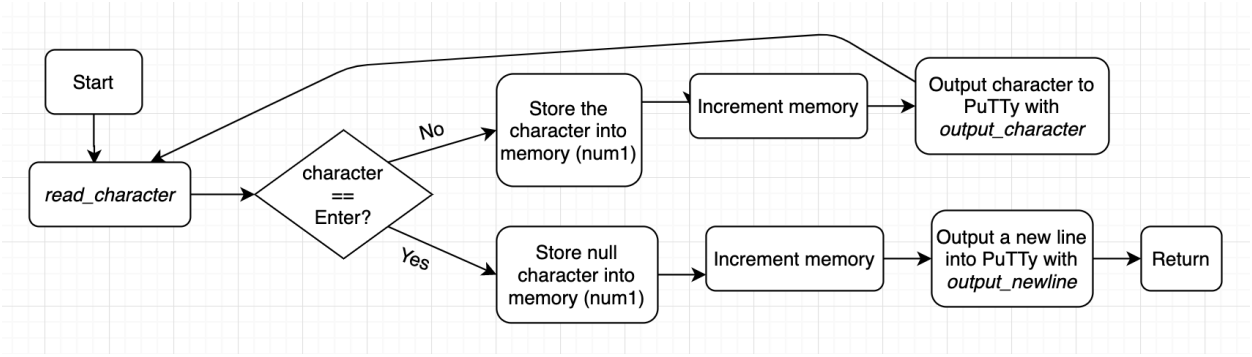
xi) read_character



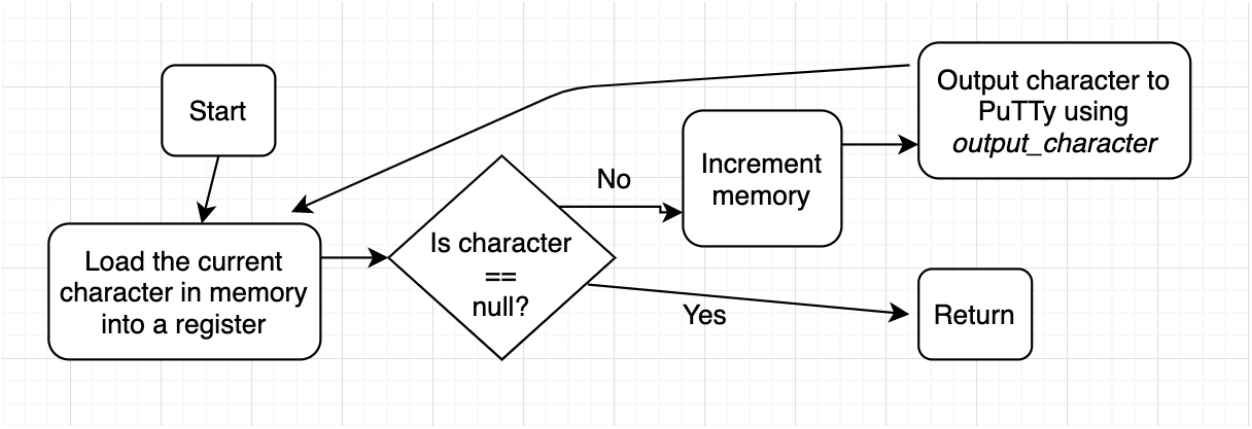
xii) output_character



xii) read_string



xiii) output_string



xiv) output_newline