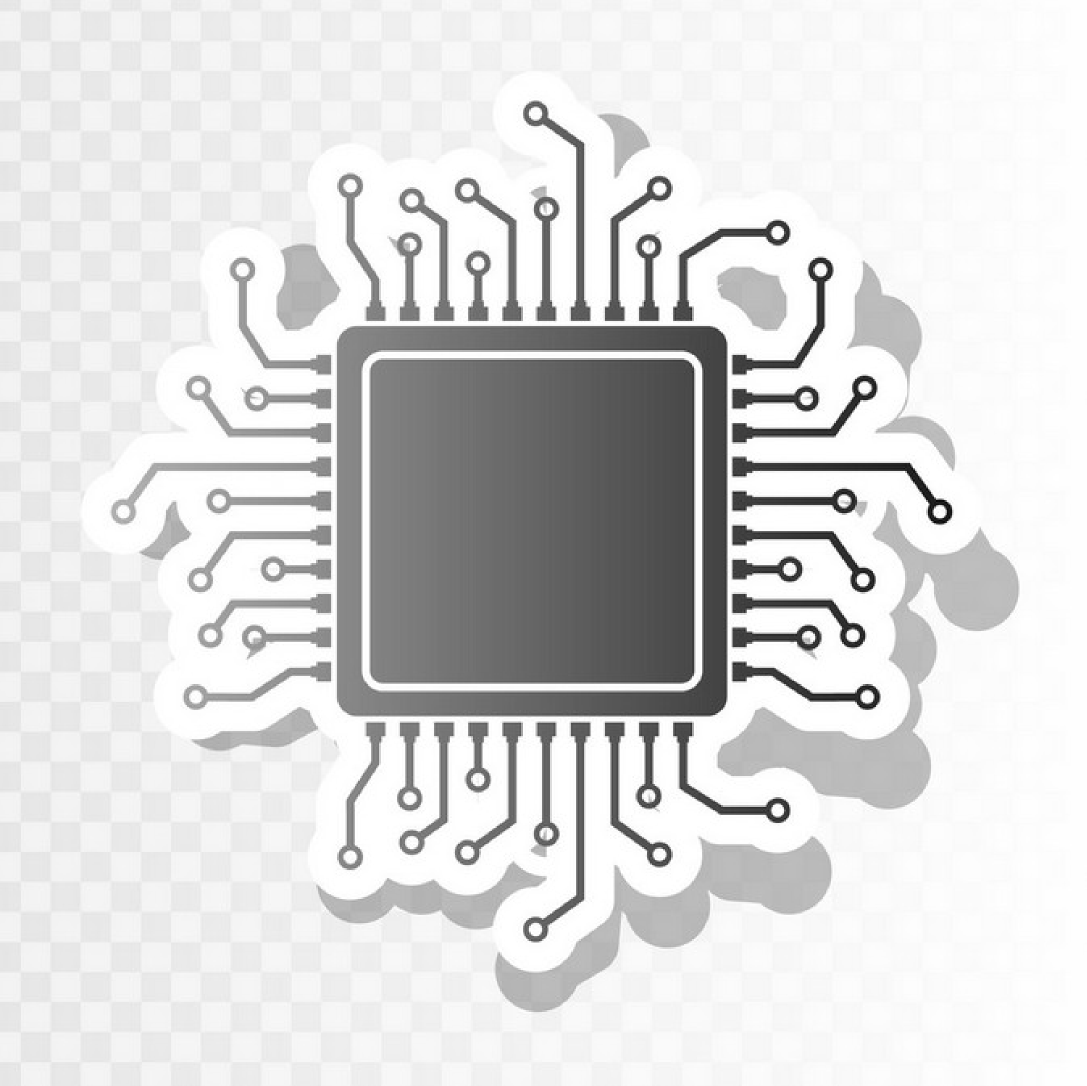
LAB 5 DOCUMENTATION

HENRY DACRES & ERIC LIU

Due March 10, 2020



CSE 379

Table of Contents

Division of Work

Subroutine	Creator
lab5:	Eric Liu
lab5loop:	Eric Liu
lab5exit:	Both
outputstrings	Henry Dacres
interrupt_init	Eric Liu
UART0_Handler	Henry Dacres
Uart_handler_exit	Henry Dacres
Switches_Handler	Both
debloop	Eric Liu

switch2_clicked	Both
switch3_clicked	Both
switch4_clicked	Both
switch5_clicked	Both
switch_exit	Henry Dacres

Purpose of Program

Objective

The objective of this Lab 5 is to create an interrupt-driven program that in real-time will display the number of times each of the four switches is pressed and the number of times a keyboard key is pressed in PuTTy. The lab focuses on getting students used to the concept of interrupts as opposed to polling—the continuous listening for an event. By the end of the lab 5, tallies should be displayed on the screen: one for switch #2, one for switch #3, one for switch #4, one for switch #5, and one of the number of keystrokes pressed. All counts will be displayed in decimal notation and will be updated on every press of a switch or a key. The program will end when the user hits “q” on the keyboard. “q” represents the action of quitting the program.

Debugging Steps

The debugging process for this lab was rigorous as we did run into quite a few issues getting the strokes from the keyboard to appear in the terminal. For debugging we followed the procedural steps of setting breakpoints on values the were incorrect and looked inside of those registers to correct suspect issues. In cases of logical and runtime errors, we referred to our flowsheets which were cross-referenced with our ARM Assembly code. We also referred to outside resources referenced below to try to potentially find routines that would shrink down our codebase.

Outside Resources

1. Arm Architecture Reference Manual
2. ARM Cortex-M4 Generic User Guide
3. ARM® and Thumb®-2 Instruction SetQuick Reference Card

Instructions

Important -

- **The highest the count will increment to is 9999**
- **After a “q” is pressed the count will no longer increment**
- **When Pushing Push Buttons wait at least 1 second before pushing another push button**

No.	Direction
1	Open Putty and load putty with the right configuration.
2	Run the program. The initial prompts will display.
3	Press the push buttons on the daughter board to increment the pushbutton count
4	Press the keys on the keyboard to update the keystroke count
5	After each press, check the Putty terminal. Following each press, the count will increment for the respective press.
6	Press “q” to quit

Summary Of Flowchart

No.	Breakdown
1	Initialize then declare global variables
2	Import library functions
3	Initialize Initial Prompt and output strings, Create Pointer to strings
4	Initialize port D(the push buttons). Initialize the interrupts
5	Unconditionally Loop
6	Output the starter prompt string along with the all the 5 counters
7	Increment each counter depending on if the pushbutton or key is pressed?
8	Exit if “q” key is pressed

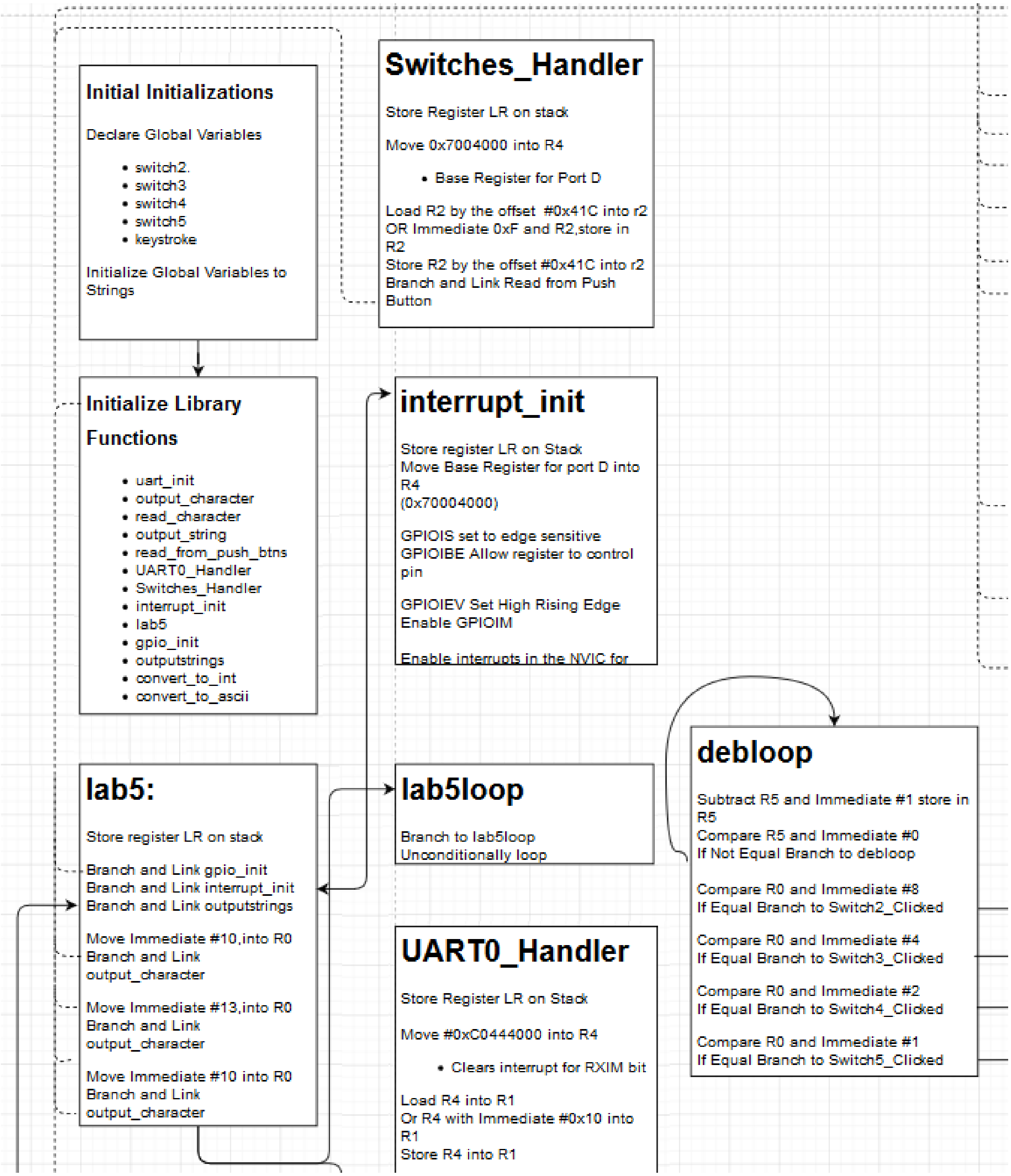
Summary of Subroutines

Subroutine	Summary
uart_init	Initializes UART for use with
output_character	Outputs a single character to the terminal
read_character	Read a single character from user input
output_string	Outputs a string to the terminal
read_from_push_btns	Gets input from push buttons.
gpio_init	Initializes port D(the push buttons)
outputstrings	Outputs the prompt again along with all the counts of the switches and keystrokes
convert_to_int	Converts the value stored at a specific address to an integer.
convert_to_ascii	Converts the value stored at a specific address to it's ASCII representation and stores the representation in memory.
lab5:	The Main
lab5loop	Unconditional loop waiting for interrupts to occur
lab5exit	Points to the end of the lab5 routine
interrupt_init	Initializes interrupts for push buttons on port D
UART0_Handler	This function allows the

	keyboard to interrupt the program
uart_handler_exit	Points to the end of the UART handler routine
Switches_Handler	This function allows the push button to interrupt the program
debloop	Dummy loop for to stall program for a couple of seconds to debounce
switch2_clicked	Increment Counter when switch 2 is pushed (Leftmost button)
switch3_clicked	Increment Counter when switch 3 is pushed (second leftmost button)
switch4_clicked	Increment Counter when switch 4 is pushed (Second rightmost button)
switch5_clicked	Increment Counter when switch 5 is pushed (Rightmost button)
switch_exit	Points to the end of the switch handler routine

Flowcharts

1. Lab 5



Initial Initializations

Declare Global Variables

- switch2.
- switch3
- switch4
- switch5
- keystroke

Initialize Global Variables to Strings

Switches_Handler

Store Register LR on stack

Move 0x7004000 into R4

- Base Register for Port D

Load R2 by the offset #0x41C into r2
OR Immediate 0xF and R2,store in R2
Store R2 by the offset #0x41C into r2
Branch and Link Read from Push Button

Initialize Library Functions

- uart_init
- output_character
- read_character
- output_string
- read_from_push_btns
- UART0_Handler
- Switches_Handler
- interrupt_init
- lab5
- gpio_init
- outputstrings
- convert_to_int
- convert_to_ascii

interrupt_init

Store register LR on Stack
Move Base Register for port D into R4
(0x70004000)

GPIOIS set to edge sensitive
GPIOIBE Allow register to control pin

GPIOIEV Set High Rising Edge
Enable GPIOIM

Enable interrupts in the NVIC for

debloop

Subtract R5 and Immediate #1 store in R5
Compare R5 and Immediate #0
If Not Equal Branch to debloop

Compare R0 and Immediate #8
If Equal Branch to Switch2_Clicked

Compare R0 and Immediate #4
If Equal Branch to Switch3_Clicked

Compare R0 and Immediate #2
If Equal Branch to Switch4_Clicked

Compare R0 and Immediate #1
If Equal Branch to Switch5_Clicked

lab5:

Store register LR on stack

Branch and Link gpio_init
Branch and Link interrupt_init
Branch and Link outputstrings

Move Immediate #10,into R0
Branch and Link output_character

Move Immediate #13,into R0
Branch and Link output_character

Move Immediate #10 into R0
Branch and Link output_character

lab5loop

Branch to lab5loop
Unconditionally loop

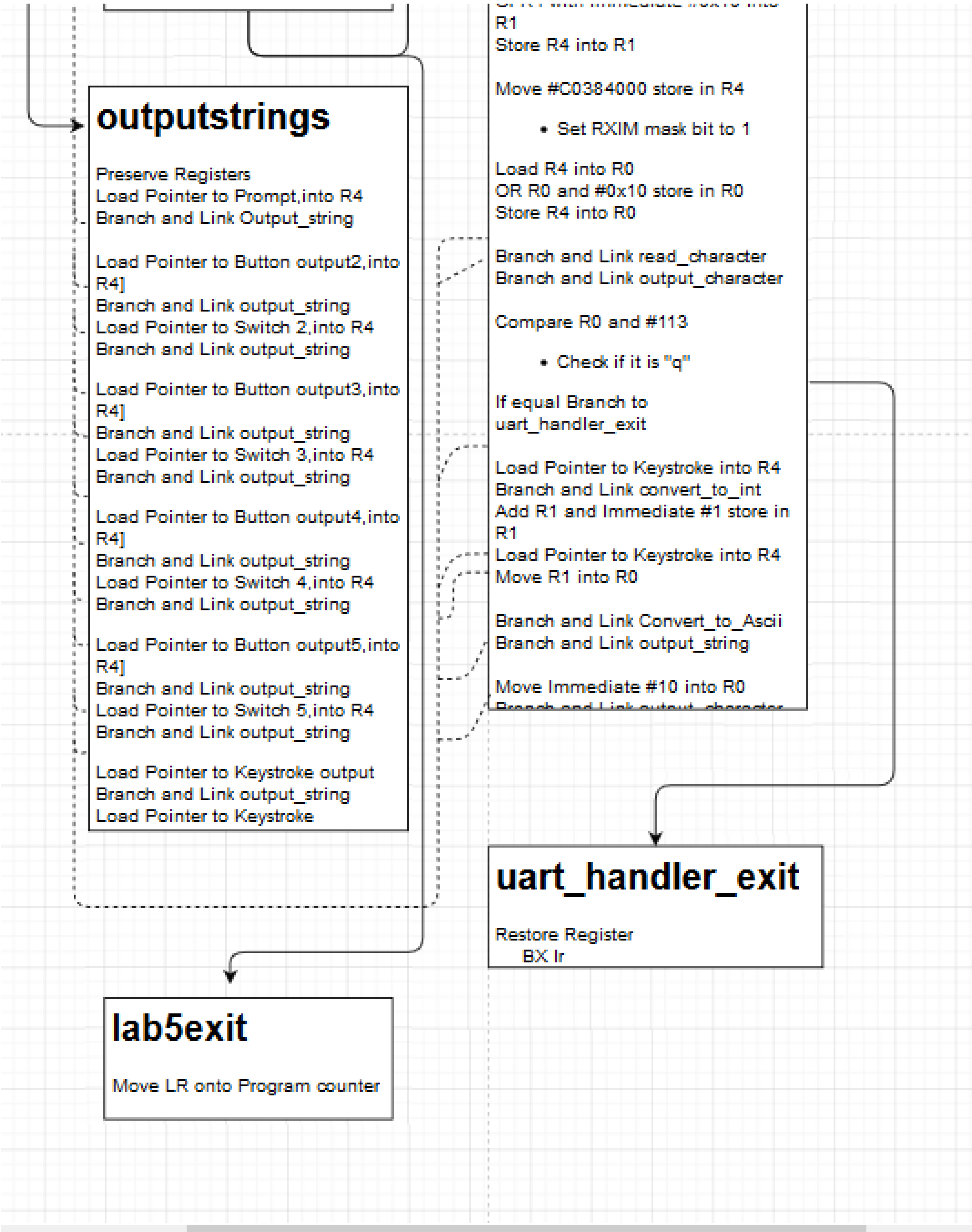
UART0_Handler

Store Register LR on Stack

Move #0xC0444000 into R4

- Clears interrupt for RXIM bit

Load R4 into R1
Or R4 with Immediate #0x10 into R1
Store R4 into R1



outputstrings

Preserve Registers
Load Pointer to Prompt,into R4
Branch and Link Output_string

Load Pointer to Button output2,into R4]
Branch and Link output_string
Load Pointer to Switch 2,into R4
Branch and Link output_string

Load Pointer to Button output3,into R4]
Branch and Link output_string
Load Pointer to Switch 3,into R4
Branch and Link output_string

Load Pointer to Button output4,into R4]
Branch and Link output_string
Load Pointer to Switch 4,into R4
Branch and Link output_string

Load Pointer to Button output5,into R4]
Branch and Link output_string
Load Pointer to Switch 5,into R4
Branch and Link output_string

Load Pointer to Keystroke output
Branch and Link output_string
Load Pointer to Keystroke

lab5exit

Move LR onto Program counter

R1
Store R4 into R1

Move #C0384000 store in R4

- Set RXIM mask bit to 1

Load R4 into R0
OR R0 and #0x10 store in R0
Store R4 into R0

Branch and Link read_character
Branch and Link output_character

Compare R0 and #113

- Check if it is "q"

If equal Branch to uart_handler_exit

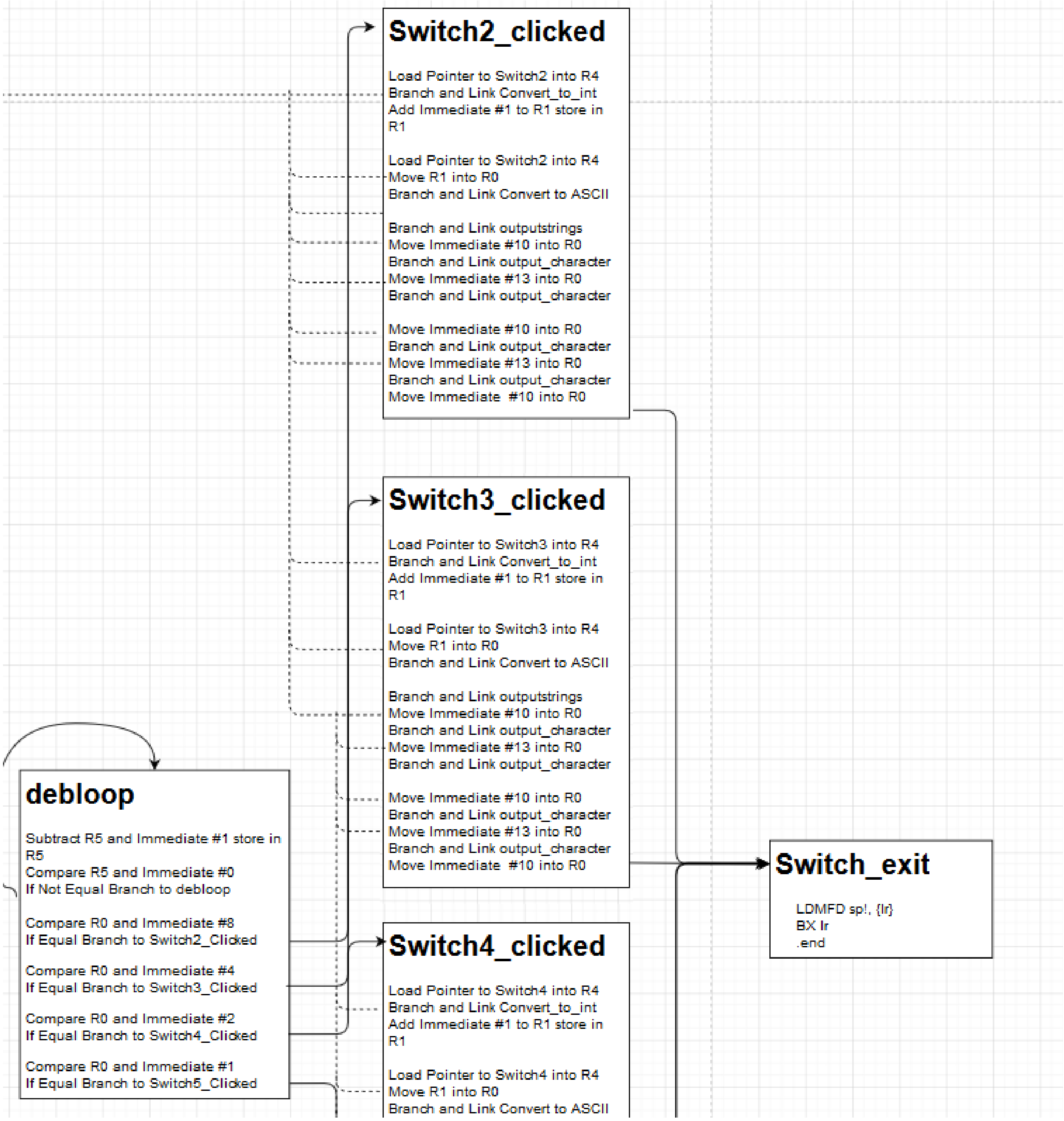
Load Pointer to Keystroke into R4
Branch and Link convert_to_int
Add R1 and Immediate #1 store in R1
Load Pointer to Keystroke into R4
Move R1 into R0

Branch and Link Convert_to_Ascii
Branch and Link output_string

Move Immediate #10 into R0
Branch and Link output_character

uart_handler_exit

Restore Register
BX lr



Switch2_clicked

```
Load Pointer to Switch2 into R4
Branch and Link Convert_to_int
Add Immediate #1 to R1 store in R1

Load Pointer to Switch2 into R4
Move R1 into R0
Branch and Link Convert to ASCII

Branch and Link outputstrings
Move Immediate #10 into R0
Branch and Link output_character
Move Immediate #13 into R0
Branch and Link output_character

Move Immediate #10 into R0
Branch and Link output_character
Move Immediate #13 into R0
Branch and Link output_character
Move Immediate  #10 into R0
```

Switch3_clicked

```
Load Pointer to Switch3 into R4
Branch and Link Convert_to_int
Add Immediate #1 to R1 store in R1

Load Pointer to Switch3 into R4
Move R1 into R0
Branch and Link Convert to ASCII

Branch and Link outputstrings
Move Immediate #10 into R0
Branch and Link output_character
Move Immediate #13 into R0
Branch and Link output_character

Move Immediate #10 into R0
Branch and Link output_character
Move Immediate #13 into R0
Branch and Link output_character
Move Immediate  #10 into R0
```

debloop

```
Subtract R5 and Immediate #1 store in R5
Compare R5 and Immediate #0
If Not Equal Branch to debloop

Compare R0 and Immediate #8
If Equal Branch to Switch2_Clicked

Compare R0 and Immediate #4
If Equal Branch to Switch3_Clicked

Compare R0 and Immediate #2
If Equal Branch to Switch4_Clicked

Compare R0 and Immediate #1
If Equal Branch to Switch5_Clicked
```

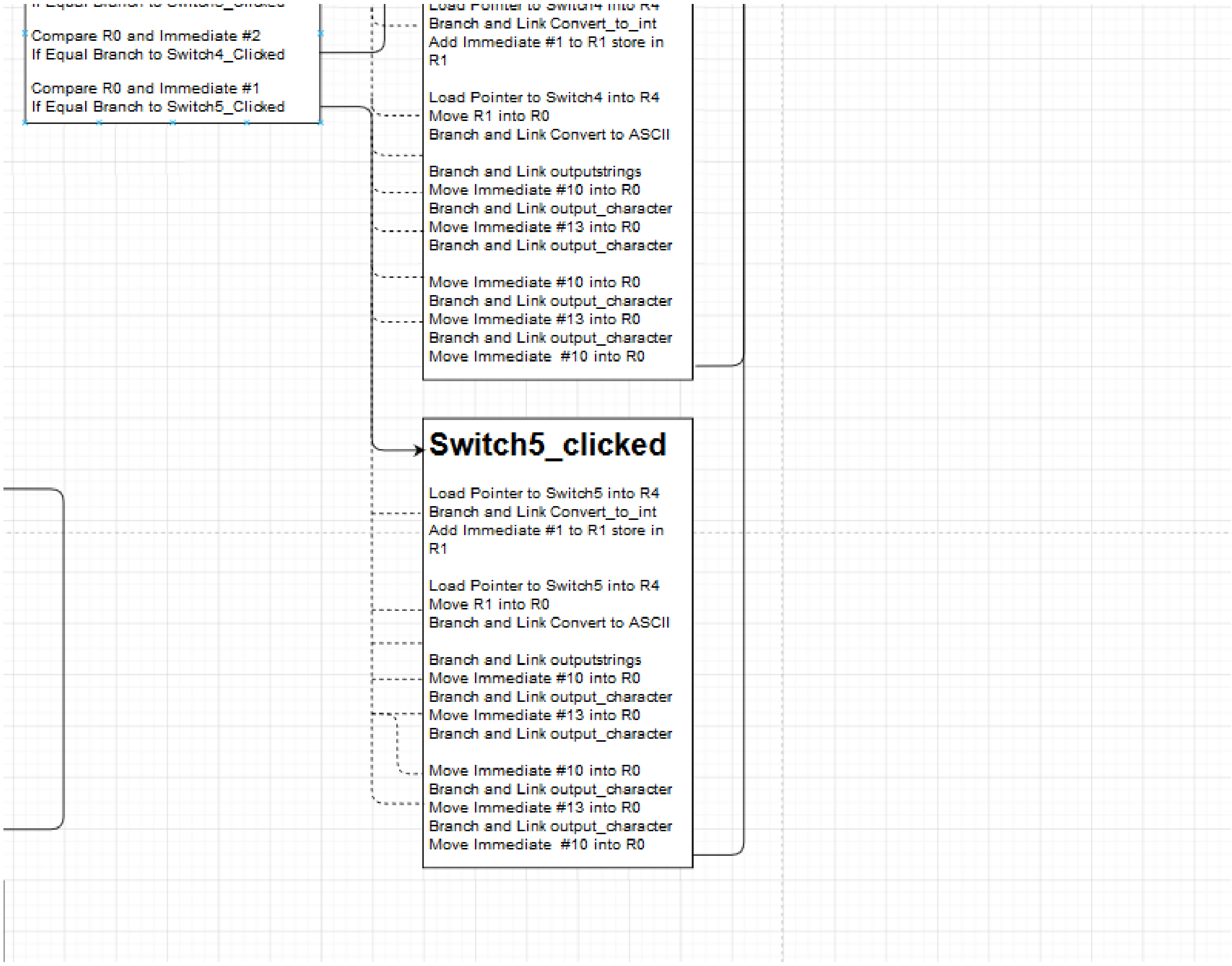
Switch_exit

```
LDMFD sp!, {lr}
BX lr
.end
```

Switch4_clicked

```
Load Pointer to Switch4 into R4
Branch and Link Convert_to_int
Add Immediate #1 to R1 store in R1

Load Pointer to Switch4 into R4
Move R1 into R0
Branch and Link Convert to ASCII
```



If Equal Branch to Switch3_Clicked

Compare R0 and Immediate #2
If Equal Branch to Switch4_Clicked

Compare R0 and Immediate #1
If Equal Branch to Switch5_Clicked

Load Pointer to Switch4 into R4
Branch and Link Convert_to_int
Add Immediate #1 to R1 store in R1

Load Pointer to Switch4 into R4
Move R1 into R0
Branch and Link Convert to ASCII

Branch and Link outputstrings
Move Immediate #10 into R0
Branch and Link output_character
Move Immediate #13 into R0
Branch and Link output_character

Move Immediate #10 into R0
Branch and Link output_character
Move Immediate #13 into R0
Branch and Link output_character
Move Immediate #10 into R0

Switch5_clicked

Load Pointer to Switch5 into R4
Branch and Link Convert_to_int
Add Immediate #1 to R1 store in R1

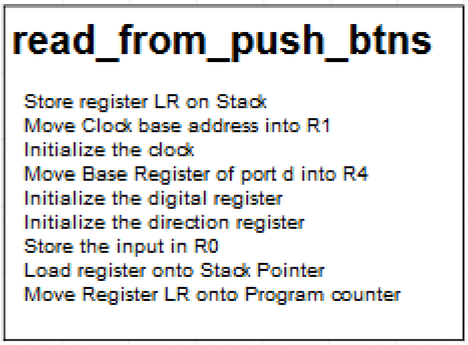
Load Pointer to Switch5 into R4
Move R1 into R0
Branch and Link Convert to ASCII

Branch and Link outputstrings
Move Immediate #10 into R0
Branch and Link output_character
Move Immediate #13 into R0
Branch and Link output_character

Move Immediate #10 into R0
Branch and Link output_character
Move Immediate #13 into R0
Branch and Link output_character
Move Immediate #10 into R0

2.

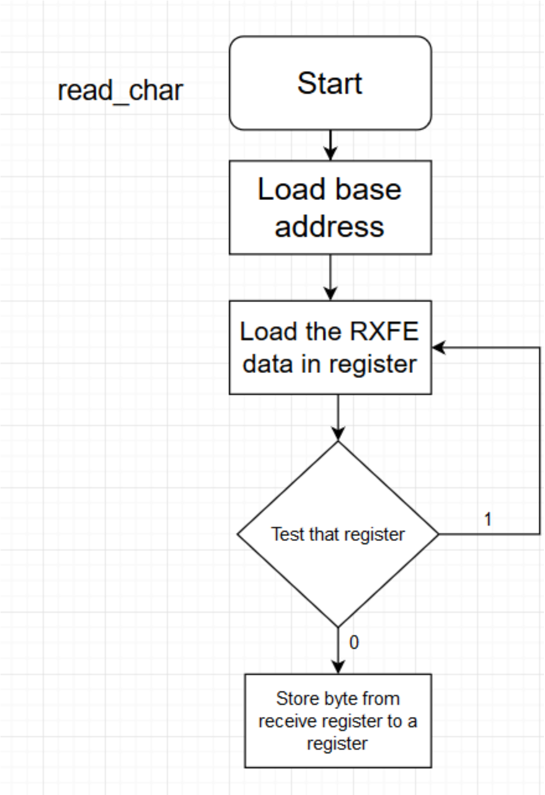
Read From Push Buttons

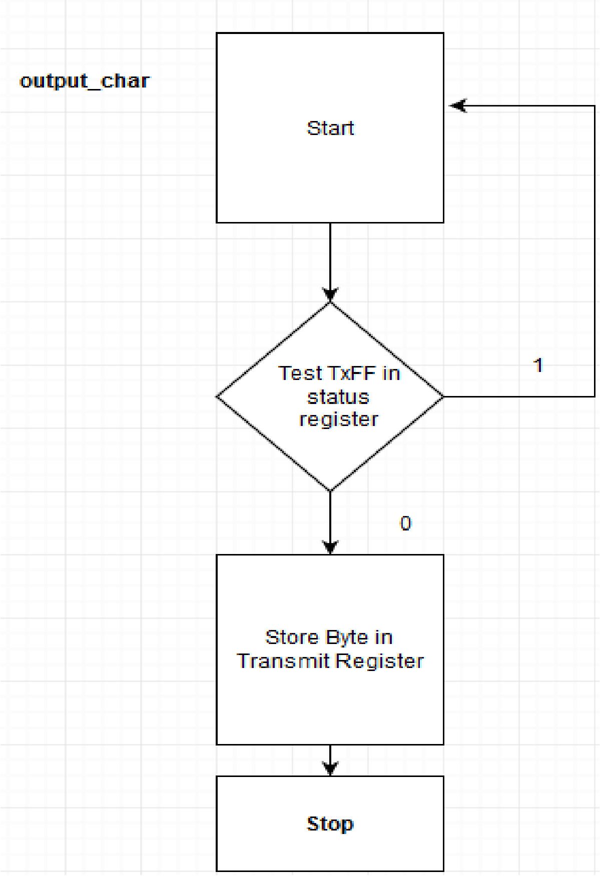


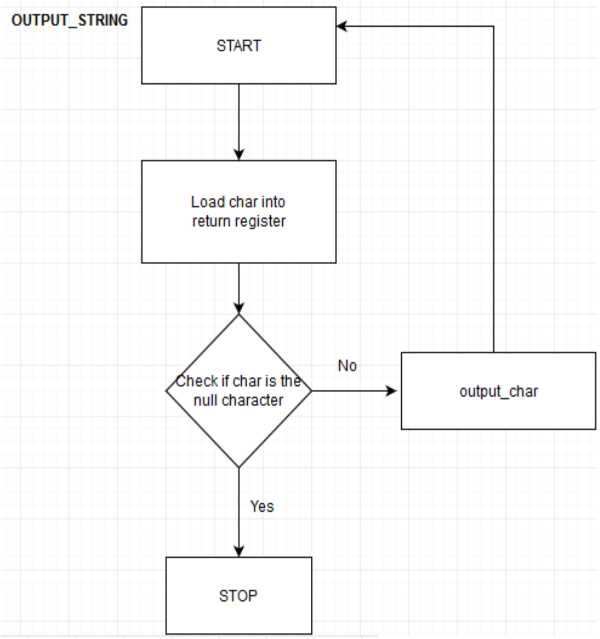
read_from_push_btns

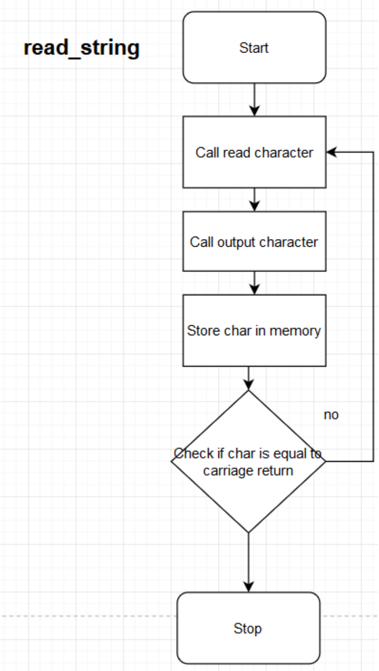
Store register LR on Stack
Move Clock base address into R1
Initialize the clock
Move Base Register of port d into R4
Initialize the digital register
Initialize the direction register
Store the input in R0
Load register onto Stack Pointer
Move Register LR onto Program counter

3.Read Char


read_char
Start
Load base address
Load the RXFE data in register
Test that register
1
0
Store byte from receive register to a register


output_char
Start
Test TxFF in status register
1
0
Store Byte in Transmit Register
Stop


OUTPUT_STRING
START
Load char into return register
Check if char is the null character
No
output_char
Yes
STOP


read_string
Start
Call read character
Call output character
Store char in memory
no
Check if char is equal to carriage return
Stop