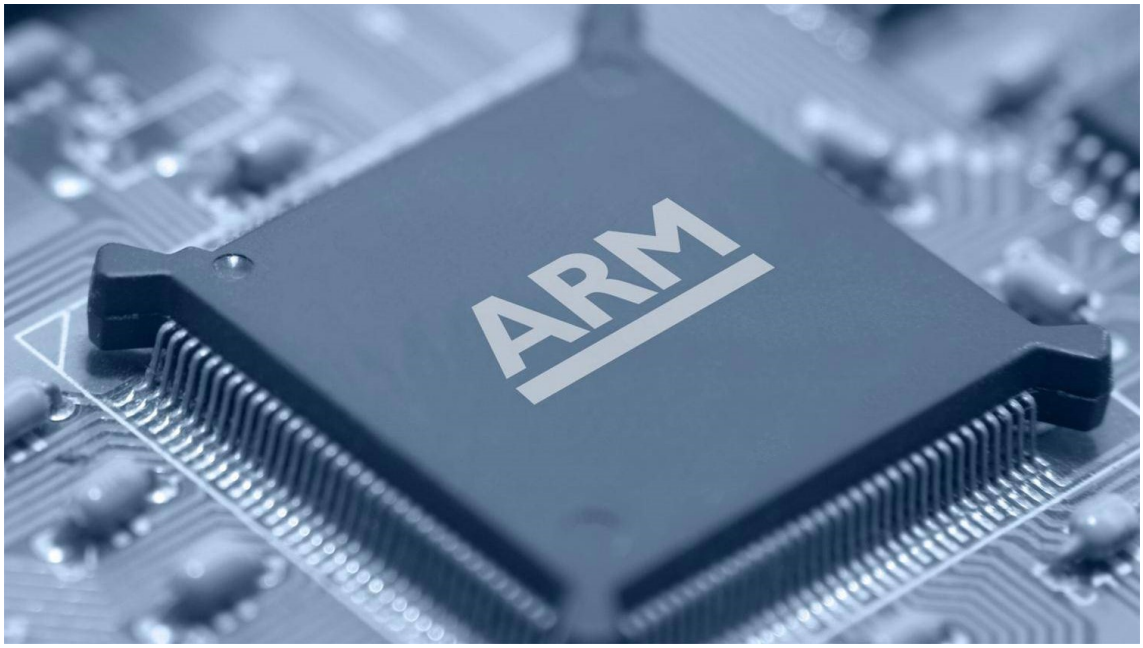
LAB 6 DOCUMENTATION

HENRY DACRES & ERIC LIU



[CSE 379]

Table of Contents

Division of Work

Subroutine	Creator
lab6:	Henry
lab6loop:	Both
interrupt_init:	Both
UART0_Handler:	Henry Dacres
Switches_Handler:	Henry Dacres
Timer0_Handler:	Both
j_pressed:	Eric Liu
i_pressed:	Eric Liu
m_pressed:	Eric Liu
k_pressed:	Eric Liu
timerexit:	Henry Dacres
lab6exit:	Henry Dacres

Purpose of Program

Objective

The objective of this lab is to create a timer-driven program. In this lab, students will learn how to use the timers on the ARM board. In addition, students will use the UART in combination with PuTTy, as has been done in previous laboratory experiments. Students will create a simple implementation of the Snake Game. When the program starts, a game board is 40 columns wide by 15 rows high. These specifications do not include the border. An asterisk will be present in the center of the board. If the asterisk hits a wall or crosses its own path, the game ends. As the game is played the score will be displayed which shows the number of asterisks that have been placed on the board. In other words, the score corresponds to the length of the path. When the game ends, the user should be notified that the game has ended.

Debugging Steps

The debugging process for this lab was rigorous as we did run into quite a few issues. During the testing process, we encountered issues when the snake was supposed to be moving left to right an asterisk would print in the opposite direction. After consulting with our previously created flowsheet and stepping through the registers we determined the register being used for the offset value was being changed for an undetermined reason. After reassessing we determined there was an issue in how we were preserving registers after our previous subroutines terminated. For debugging we followed the procedural steps of setting breakpoints on values that 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

!!! DO NOT PRESS ANY KEY OTHER THAN i, j, k or m!!

No.	Instruction
1	Open the Program within the terminal
2	Press ‘k’ to move the snake right
3	Press ‘m’ to move the snake downward
4	Press ‘j’ to move the snake left
5	Press ‘i’ to move the snake upward
7	Crossing a ‘-’ , ‘|’ or ‘*’ will end the game

Summary Of Flowchart

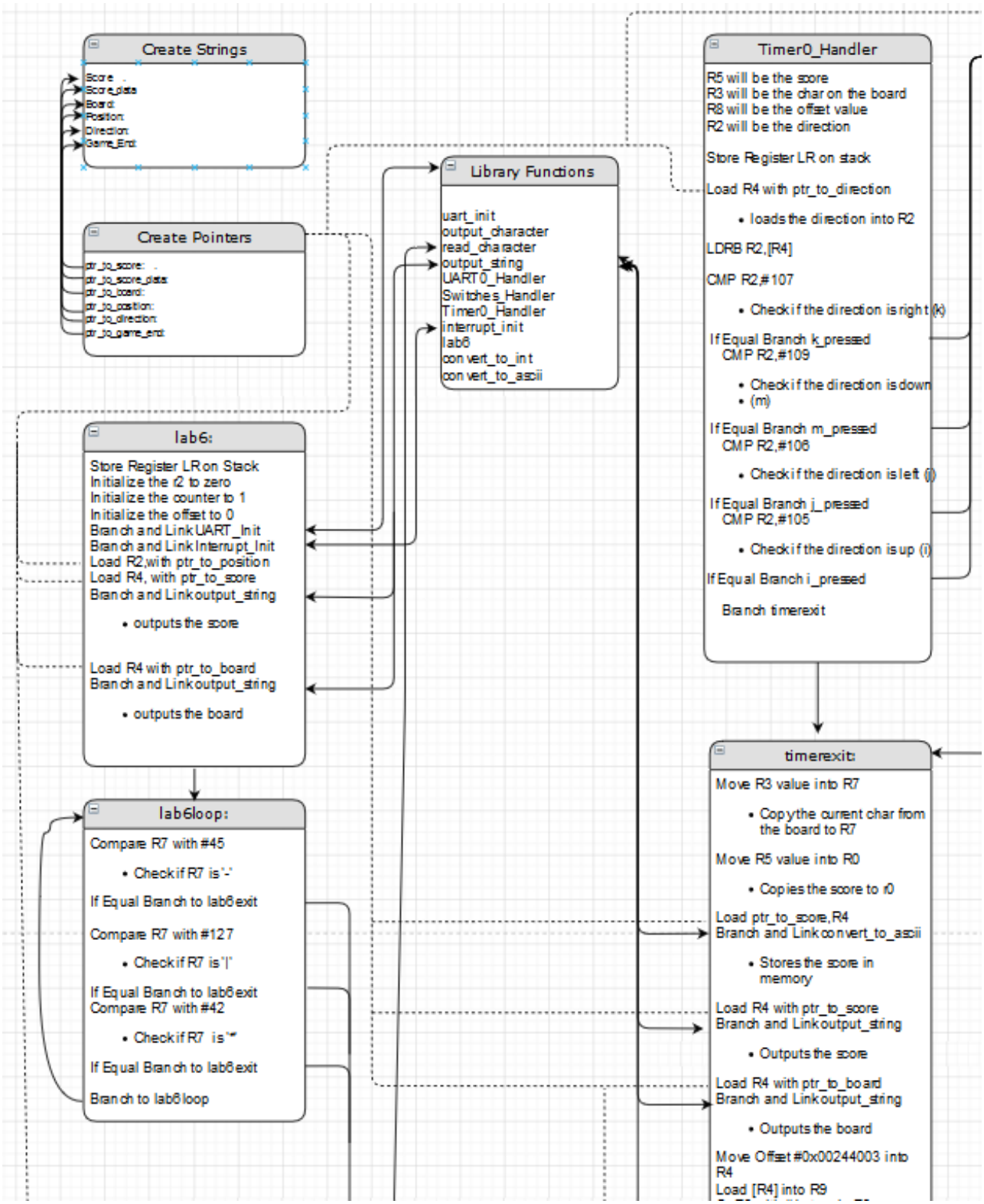
No.	Breakdown
1	Initialize Strings
2	Import Library Functions
3	Create Pointers to Strings
4	Output the new game board
5	Read user input, update board
6	Check if the “snake” has hit a Loss condition, if so output score and end game
7	If a loss condition has not been hit Read user input and update board

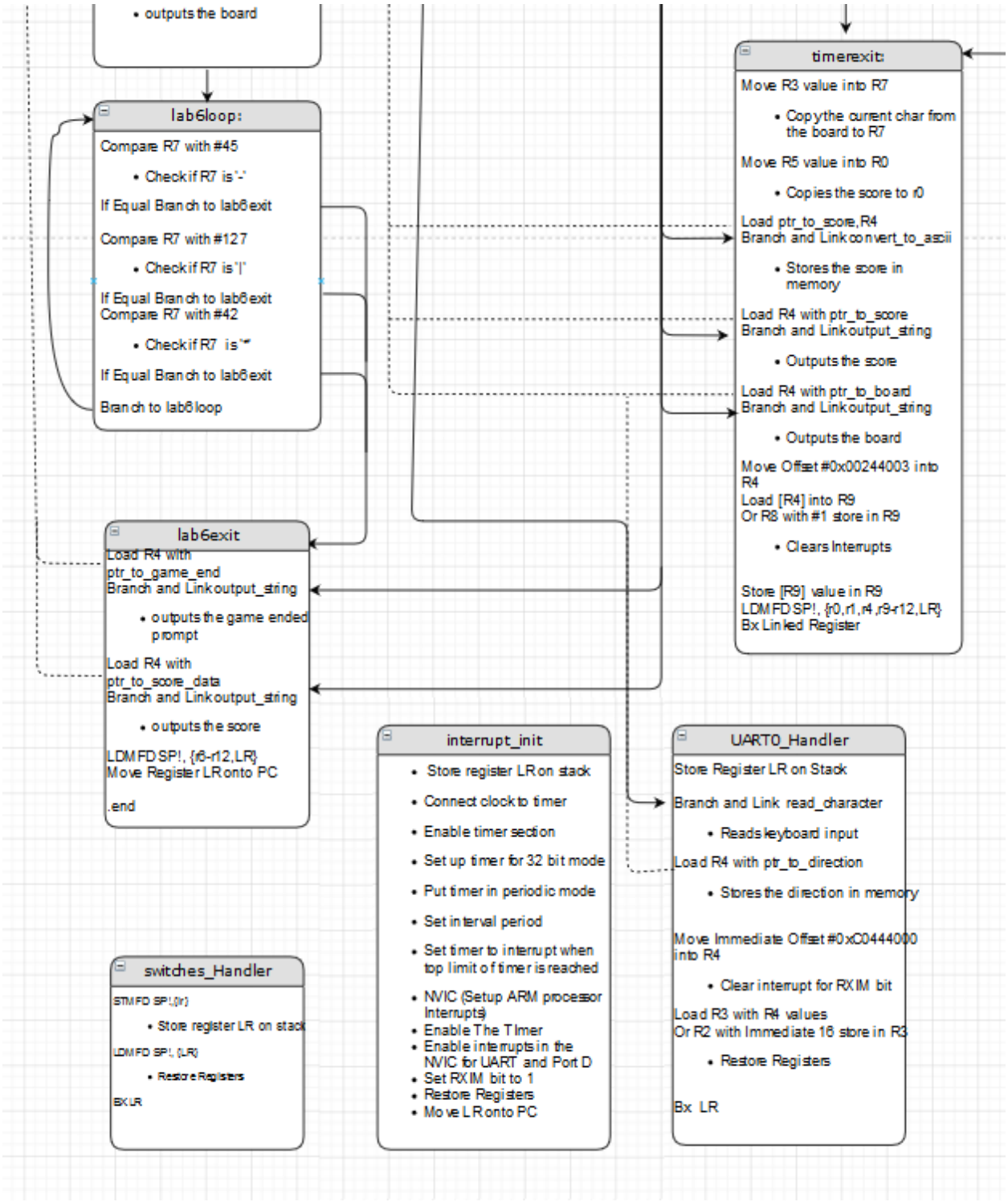
Subroutine	Summary
uart_init	Initializes UART for use with

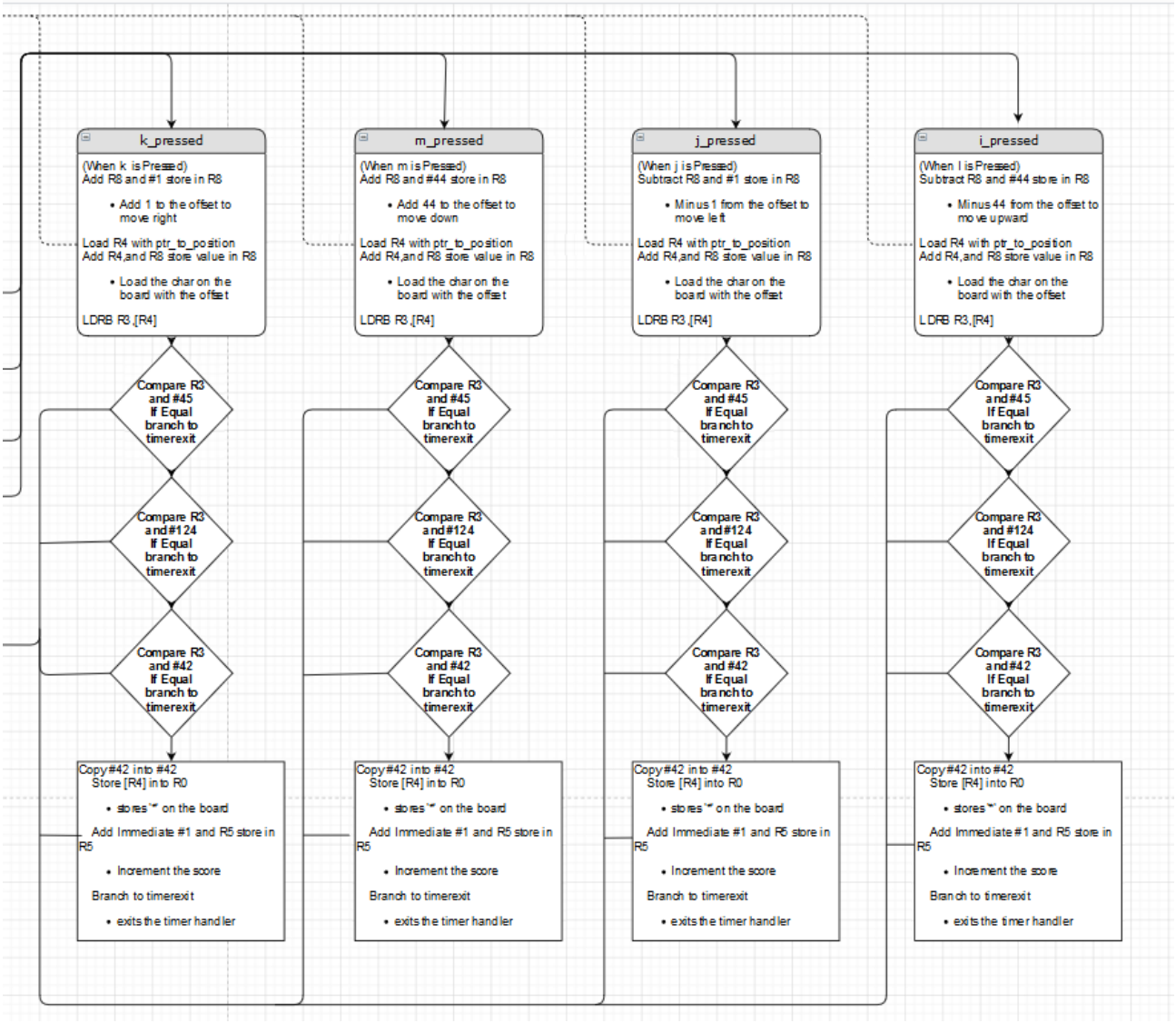
output_character	Outputs a single character to the terminal
read_character	Read a single character from user input
output_string	Outputs a string to the terminal
convert_to_int	Converts the value stored at a specific address to an integer.
convert_to_ascii	Converts the value stored at a specific address to it’s ASCII representation and stores the representation in memory.
lab6:	The Main
lab6loop	Unconditional loop waiting for interrupts to occur
lab6exit	Points to the end of the lab6 routine
interrupt_init	Initializes interrupts for push buttons on port D
UART0_Handler	This function allows the keyboard to interrupt the program
Switches_Handler	This function allows the push button to interrupt the program
timerexit	Points to the end of the Timer handler routine
Timer0Handler	This function allows the Timer to interrupt the program
k_pressed	Moveright if k is press

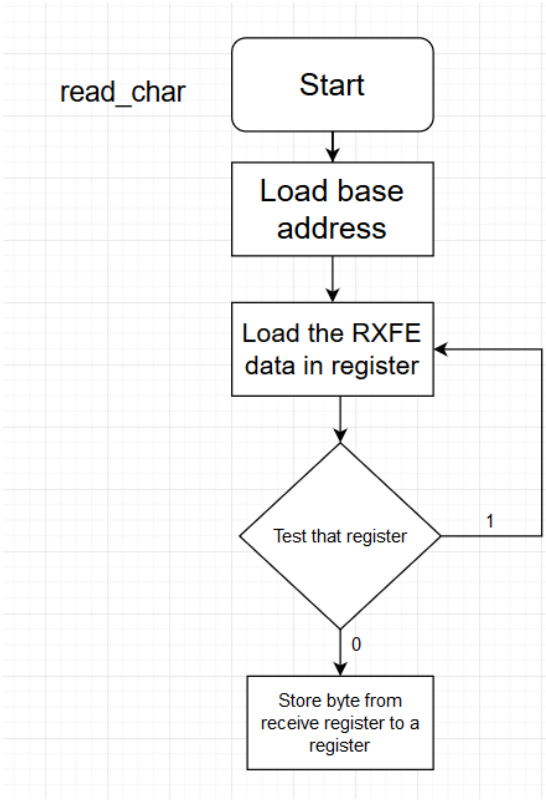
m_pressed	Move downward if m is press
j_pressed	Move left if j is press
i_pressed	Move upward if I is press

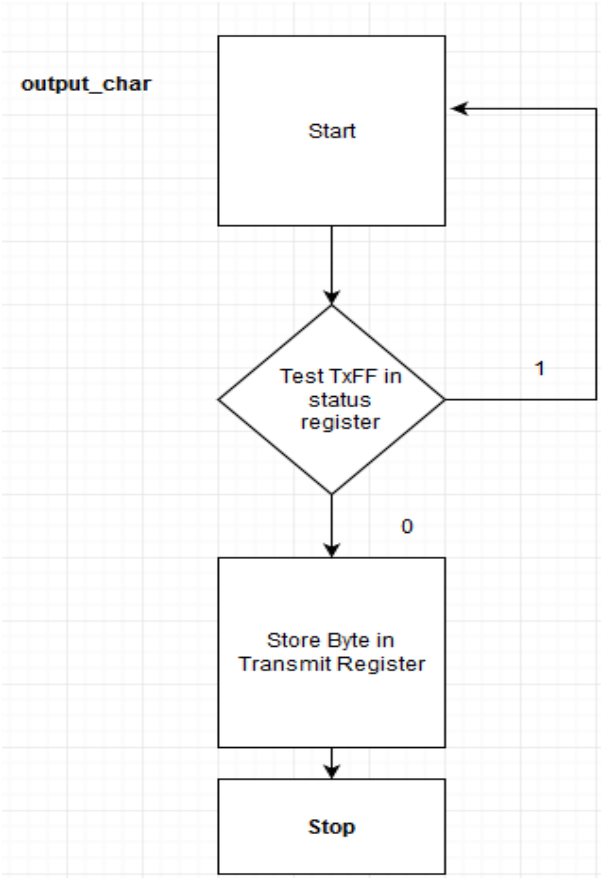
FLOWCHART

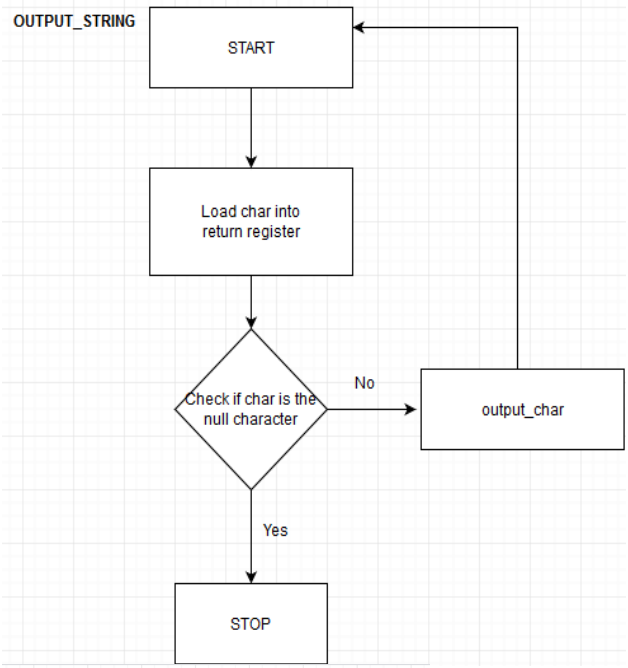

Create Strings
Score :
Score_plate
Board:
Position:
Direction:
Game_End:
Create Pointers
ptr_to_score:
ptr_to_score_plate:
ptr_to_board:
ptr_to_position:
ptr_to_direction:
ptr_to_game_end:
Library Functions
uart_init
output_character
read_character
output_string
UART0_Handler
Switches_Handler
Timer0_Handler
interrupt_init
lab6
convert_to_int
convert_to_ascii
Timer0_Handler
R5 will be the score
R3 will be the char on the board
R8 will be the offset value
R2 will be the direction
Store Register LR on stack
Load R4 with ptr_to_direction
• loads the direction into R2
LDRB R2,[R4]
CMP R2,#107
• Check if the direction is right (k)
If Equal Branch k_pressed
CMP R2,#109
• Check if the direction is down
• (m)
If Equal Branch m_pressed
CMP R2,#106
• Check if the direction is left (j)
If Equal Branch j_pressed
CMP R2,#105
• Check if the direction is up (i)
If Equal Branch i_pressed
Branch timerexit
lab6:
Store Register LR on Stack
Initialize the r2 to zero
Initialize the counter to 1
Initialize the offset to 0
Branch and Link UART_Init
Branch and Link Interrupt_Init
Load R2,with ptr_to_position
Load R4, with ptr_to_score
Branch and Link output_string
• outputs the score
Load R4 with ptr_to_board
Branch and Link output_string
• outputs the board
lab6loop:
Compare R7 with #45
• Check if R7 is '-'
If Equal Branch to lab6exit
Compare R7 with #127
• Check if R7 is '|'
If Equal Branch to lab6exit
Compare R7 with #42
• Check if R7 is '*'
If Equal Branch to lab6exit
Branch to lab6loop
timerexit:
Move R3 value into R7
• Copy the current char from the board to R7
Move R5 value into R0
• Copies the score to r0
Load ptr_to_score,R4
Branch and Link convert_to_ascii
• Stores the score in memory
Load R4 with ptr_to_score
Branch and Link output_string
• Outputs the score
Load R4 with ptr_to_board
Branch and Link output_string
• Outputs the board
Move Offset #0x00244003 into R4
Load [R4] into R9

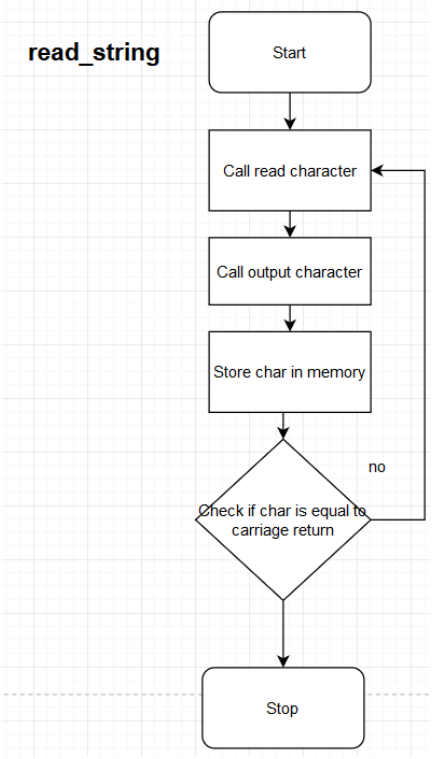

• outputs the board
lab6loop:
Compare R7 with #45
• Check if R7 is '-'
If Equal Branch to lab6exit
Compare R7 with #127
• Check if R7 is '|'
If Equal Branch to lab6exit
Compare R7 with #42
• Check if R7 is '*'
If Equal Branch to lab6exit
Branch to lab6loop
timerexit:
Move R3 value into R7
• Copy the current char from the board to R7
Move R5 value into R0
• Copies the score to r0
Load ptr_to_score,R4
Branch and Link convert_to_ascii
• Stores the score in memory
Load R4 with ptr_to_score
Branch and Link output_string
• Outputs the score
Load R4 with ptr_to_board
Branch and Link output_string
• Outputs the board
Move Offset #0x00244003 into R4
Load [R4] into R9
Or R8 with #1 store in R9
• Clears Interrupts
Store [R9] value in R9
LDMFD SP!, {r0,r1,r4,r9-r12,LR}
Bx Linked Register
lab6exit
Load R4 with ptr_to_game_end
Branch and Link output_string
• outputs the game ended prompt
Load R4 with ptr_to_score_data
Branch and Link output_string
• outputs the score
LDMFD SP!, {r6-r12,LR}
Move Register LR onto PC
.end
interrupt_init
• Store register LR on stack
• Connect clock to timer
• Enable timer section
• Set up timer for 32 bit mode
• Put timer in periodic mode
• Set interval period
• Set timer to interrupt when top limit of timer is reached
• NVIC (Setup ARM processor Interrupts)
• Enable The TImer
• Enable interrupts in the NVIC for UART and Port D
• Set RXIM bit to 1
• Restore Registers
• Move LR onto PC
UART0_Handler
Store Register LR on Stack
Branch and Link read_character
• Reads keyboard input
Load R4 with ptr_to_direction
• Stores the direction in memory
Move Immediate Offset #0xC0444000 into R4
• Clear interrupt for RXIM bit
Load R3 with R4 values
Or R2 with Immediate 16 store in R3
• Restore Registers
Bx LR
switches_Handler
STMFD SP!,{lr}
• Store register LR on stack
LDMFD SP!, {LR}
• Restore Registers
BX LR


k_pressed
(When k is Pressed)
Add R8 and #1 store in R8
• Add 1 to the offset to move right
Load R4 with ptr_to_position
Add R4,and R8 store value in R8
• Load the char on the board with the offset
LDRB R3,[R4]
m_pressed
(When m is Pressed)
Add R8 and #44 store in R8
• Add 44 to the offset to move down
Load R4 with ptr_to_position
Add R4,and R8 store value in R8
• Load the char on the board with the offset
LDRB R3,[R4]
j_pressed
(When j is Pressed)
Subtract R8 and #1 store in R8
• Minus 1 from the offset to move left
Load R4 with ptr_to_position
Add R4,and R8 store value in R8
• Load the char on the board with the offset
LDRB R3,[R4]
i_pressed
(When I is Pressed)
Subtract R8 and #44 store in R8
• Minus 44 from the offset to move upward
Load R4 with ptr_to_position
Add R4,and R8 store value in R8
• Load the char on the board with the offset
LDRB R3,[R4]
Compare R3 and #45 If Equal branch to timerexit
Compare R3 and #124 If Equal branch to timerexit
Compare R3 and #42 If Equal branch to timerexit
Copy#42 into #42
Store [R4] into R0
• stores '*' on the board
Add Immediate #1 and R5 store in R5
• Increment the score
Branch to timerexit
• exits the timer handler


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
Check if char is equal to carriage return
no
Stop