CSE 379 Lab 7
Documentation Report

Ian Witmer: ianwitme
Brian Leavell: bsleavel

Lab Section: R2
Date: 5/6/2023

Table of Contents

Section 1

Partner Name: Brian Leavell

Project Contributions:
Responsible for the flowchart and logic designs for functionality-based subroutines ***toy_maker, wreck_it_ralph, update_OOB, life_count, Switch_Handler, lvls_counter, advance_check,*** and ***levelup.*** Helped debug final assembly code and compile documentation for the final report.

Partner Name: Ian Witmer

Project Contributions:
Responsible for the flowchart and logic designs for functionality-based subroutines ***startup, copy_string, reset_initial_values, Timer_Handler, UART0_Handler, update_paddle, update_ball, output_board, update_ball_color.*** Helped debug final assembly code and finalize the flow charts' designs.

Section 2

Program Overview

1.) Start running the program on PuTTY
2.) A prompt appears informing the user to hold down a number of pushbuttons on the daughterboard (SW2-5) and press S on the keyboard to start the game
3.) Based on how many buttons the user holds down, the game will begin with that many rows of bricks (1-row minimum by default if no buttons are pressed, 4 rows maximum)
4.) The ball will appear in the center of the screen moving down towards the paddle, and then the user can use A and D keys to move the paddle left and right
5.) When the ball hits certain parts of the paddle it will begin moving up at a certain angle (45, 60, or 90 degrees)
6.) When the ball hits a brick, the score will increase and the brick will be destroyed. The ball and RGB LED will change to the color of the most recently destroyed brick. Both the ball and LED colors reset on death (White ball, LED off)
7.) The user has 4 lives indicated by the LEDs on the daughter board. If the ball hits the bottom of the board past the paddle, a life is lost. At 0 lives the game will end
8.) To advance to the next level, the user must clear all the bricks on the board, at which point the board will reset and the ball will begin to move faster. After 4 levels the ball speed will no longer increase
9.) To pause the game, the user can press SW1 on the motherboard. The RGB will light up blue and the board will display the word PAUSED. To resume, the same button can be pressed
10.) When the game ends, the user can press Y to restart the game or Q to quit

Program Summary

This program integrates several important functions of the ARM processor. These include timers, interrupts, GPIO, lookup tables, escape sequences, and UART communication. The game begins

by telling the user the push down the switch buttons on the motherboard (SW2-SW5) and press S to start the game. The subroutines ***lvl_count*** and ***toy_maker*** are responsible for creating the board for the user. ***lvl_count*** is called once the user presses S, in which is checks the data register for the SW pushbuttons and checks to see how many buttons were pushed, and stores this value in memory. ***Toy_maker*** is then called which loads this value from memory to determine how many rows to print out. If no buttons were pressed and the user pressed S the default value of 1 is loaded and stored (Default row count is 1). Inside ***toy_maker***, an accumulator is set to make sure that 7 bricks are printed per row and the respective number of rows are made. The rows and brick colors are randomly generated by utilizing Timer A1, in which the current time value in that Timer's data register is pulled and has a "mod5" operation applied to the value. Based on the result, a certain colored brick is created by moving to a specific offset inside the board in memory, and writing 3 letters next to each other. Lower case letters *r,b,g,y, and p* all correspond to a color brick, which the lookup table prints out an ASCII Escape sequence to make the PuTTY terminal change colors. For example, if the user holds down 3 buttons and presses S the game will begin with 3 layers of bricks on the board. After this, ***life_count*** is called which loads the pointer to the user's lives and inserts a number into the LED data register based on the number of lives. At the start of the game the user gets 4 lives, so the value 15 is passed into the data register to turn all of the LEDs on. The game begins with the ball moving towards the paddle, with the user being able to press A and D to move the paddle left and right. The ball and paddle's simulated "movement" is handled inside the Timer Handler. In this handler, subroutines ***update_paddle*** and ***update_ball*** are constantly being called, which update the position of each inside the board (aka inside memory). Both of these subroutines use the lookup table to keep track of the positioning and color of both the ball and paddle. Based on what part of the paddle connects with the ball (seen in the lookup table), the ball will change its x and y vectors and begin to move at a certain angle. If the ball hits a wall it will bounce off; This is used in ***update_ball*** which updates the vectors and coordinates based on what character is in the space that it is **about** to connect with. If the ball collides with a brick, the subroutine ***wreck_it_ralph*** is called, which has the location of the brick that was just hit passed into it in two registered. This function determines what color the brick was (based on the lowercase letter it reads) and stores it at a pointer called ***ball_color*** which is used by ***update_ball*** to change the color of the ball, followed by passing a certain value into the RGB LED data register to light it up the same color. After, the subroutine uses the x and y coordinates passed to it to move to an offset in the board, after which is "deletes" the brick by writing 3 spaces in place of the lowercase letters. These spaces in the lookup table correspond to a blank space with a black background. Two pointers in memory are then loaded (level score and score), incremented, and stored back. The level score is based on the number of rows the user has in their board, which they need to clear in order to level up in the game. In the Timer Handler, the subroutine ***advance_check*** is called to determine if the user has cleared all the bricks by loading this score from memory and comparing it to either 7, 14, 21, or 28. If any of these conditions are met, a boolean in memory is set to 1, in which after ***advance_check*** returns we then jump to ***level_up***. This subroutine speeds up the ball

by subtracting a value from the timer frequency, and then resets the ball and paddle positioning along with the level score, and then creates a new board with newly generated rows of bricks. If the SW1 button is pushed, the Switch Handler is called which handles the interrupt created by the button press. In this handler,it checks a boolean value in memory to determine if this is the user pausing the game or resuming it. If it is a pause, the main timer for the game is disabled along with the RGB LED turning blue and the word “PAUSED” printed underneath the user’s board. If it is a resume, the screen is cleared, the RGB LED is turned back to the previous color (stored at a pointer in memory), and the timer is enabled again, which eventually will reprint the board and resume the game. If the ball hits the bottom of the board, aka out of bounds, ***update_ball*** will detect this based on the x and y coordinates of the ball. If it is true, ***update_OOB*** is called which passes 0 to the ***illuminate_RGB_LED*** subroutine which turns the light off. Next, the ball color is reset along with the positioning of the ball and paddle. After that, the pointer holding the user’s life count is loaded and decremented, followed by the calling of ***life_count*** to update the LEDs and in essence turn one of the LEDs off. If the user loses all 4 lives, all of the LEDs will turn off and the game will end, displaying a game over prompt. The user will also be able to see their final score, and have the option to restart the game or quit by pressing *y* or *q* respectively.. If the user chooses to restart the game, all initial values are restored (0 for level count, 0 for score, 4 for lives, ball and paddle coordinates reset), and the user has the option to select a new board with a different number of rows. If the user presses *q,* the program ends.

lab7


START
reset_initial_values
uart_init
gpio_btn_and_LEDS_init
uart_interrupt_init
gpio_interrupt_init
load 0xC into r0
output_character
load into r0 ptr_to_start_prompt
output_string
load into r0 ptr_to_prompt
output_string
load into r0 ptr_to_last_button_pressed
r0 == 115?
No
Yes
load into r0 ptr_to_template_bored
load into r1 ptr_to_bored
copy_string
Timer_init
lvls_counter
life_count
toy_maker
load into r0 ptr_to_bored
output_board
load into r0 ptr_to_lives
r0 == 0?
No
Yes
gameover

Section 3

Subroutine Descriptions:

lab7: Initializes GPIO, Timer, UART0, along with the interrupts for them. Resets all base values for pointers in memory, and outputs prompt followed by polling for a user to press S to begin the program. Calls all subroutines to build the board and check constantly if the user has run out of lives

get_length: Takes an integer in r0 and returns the length of the integer in r0.

get_nth: Takes one integer in r0 and another integer in r1, and returns an integer in r0. Returns the digit of the integer in r0 at the index described by the integer in r1. The indexing begins at the least significant digit, with an initial value of 0.

uart_interrupt_init: Configures the processor to allow the UART to interrupt it

gpio_interrupt_init: Turns on the clock for port F and sets the right pins to configure SW1 on the Tiva board to be an input. It also configures SW1 to be an interrupt (single falling edge trigger), clears the interrupt flag, and allows the GPIO port F to use interrupts.

gpio_btn_and_LEDs_init: Turns on the clocks, sets required pins to be inputs and outputs, and sets pull-up registers for required pins to operate the switches and corresponding LEDs on the daughter board.

Timer_init: Initializes the timers A0 and A1 and configures the processor to allow the timer A0 to interrupt it. The timer will interrupt the processor at a specific internal (5 interrupts per second).

read_character: Detects a character input from a keyboard and stores the ASCII value of the character in r0.

output_character: Takes an ASCII value of a character in r0 and stores the corresponding character to the memory-mapped I/O of a UART.

output_string: Takes a memory address as an input in r0. Reads a null-terminated string from the address in r0 and sequentially stores the values of the characters to memory-mapped I/O of a UART

uart_init: Modifies values in the memory-mapped I/O of a UART to prepare it to send and receive data in ways meaningful to the lab6 program. With this initialization, output_character and output_string will cause the value(s) they store in memory to be converted to a character(s) and printed to a PuTTY terminal.

int2string: Takes an integer in r0 and a memory address of a string buffer in r1. Converts the integer in r0 into a null-terminated string describing the number, and stores it at the buffer pointed to by r1.

illuminate_RGB_LED: Starts by loading the address offset for port F, setting pins F1-F3 to outputs as well as setting them to digital. Next, it loads the data register for port F, and inserts the 3 bits that were passed into the routine in r0 into the data register. This will turn on the corresponding pins and turn on the RGB LED with the specific color (according to the 3-bit combination)

illuminate_LEDs: Starts by loading the data register for port B, and inserting the 4 bits that were passed into the routine in r0 into the first 4 bits of r1. This new value is stored in the port B data register, hence turning on the corresponding LEDs.

UART0_Handler: Reads the key that the user last pressed using ***read_character*** and updates the direction that the paddle is moving using pointers in memory which are loaded in ***update_paddle.***

Switch_Handler: Pauses or resumes the game based on a boolean value that gets set each time SW1 is pressed. If set to a 0, the timer is disabled, RGB LED turned blue, and the word PAUSED is printed to the screen. If the boolean is a 1, RGB LED is turned to the previous color before being paused, and the timer is reenabled with PAUSED getting cleared from the screen

Timer_Handler: Checks to see if the user has completed the level first. Then it updates the positioning of the ball, paddle, and score count in memory, clears the screen and prints it all to the terminal.

Output_board: Similar to the output_string function, except when it comes across specific characters in the string to print, it instead accesses a lookup table full of strings and prints one that corresponds to the specific character. This allows the lookup table to be filled with escape sequences that when printed out in PuTTY, have formatting applied to characters.

update_paddle: Accesses an address in memory that contains either “a” or “d”; whichever was pressed most recently on the keyboard. Based on the value, loads the position of the paddle and moves it left or right. The value at the address can also be a space (which is only the case if “a” or “d” have not been pressed yet), and in this default case the paddle position does not change. The position will also not change if the paddle is trying to move past the boundaries of the board.

update_ball: Loads the x and y coordinates of the ball, and the movement vector of the ball. Adds the movement vector to the coordinates, and checks to see what character on the board is there. If it is not a space (there is a collision), the movement vector of the ball is updated according to the character that is being collided with. Also, if a block is being collided with, wreck_it_ralph is called to destroy the block. After updating the vector, the position of the ball is added to the new movement vector, and one more collision check is made. If there is a collision, the routine ends. If there is not, a character for the ball is placed in the new location, and a space is placed in the old location.

copy_string: Takes the address of a null-terminated string in r0 and an address to store a null-terminated string of the same size in r1. Copies all the characters in the string at r0 to the location in r1, with the null terminator. Does not destroy the string at r0. Used to reset the game board.

update_OOB: Resets the color of the ball and RGB LED based on a pointer in memory, repositions the ball and its vectors to its starting values, subtracts 1 from the user's life count value stored in memory followed by calling ***life_count***

gameover: Disables the timer, clears the screen and outputs the final score along with an option to restart the game or quit

wreck_it_ralph: Given the board position of the most recently hit brick (in r0 and r1), updates the color of the ball and RGB LED based on the color of the brick, increments the overall score and the level score, then writes spaces over the brick in memory to "destroy" the brick

toy_maker: Loads pointers from memory to check how many rows of bricks to print as well as how many bricks have been printed so far in a row (Default number of rows is 1). Loads current value of Timer A1, performs "mod5" operation on it to determine what color to make the brick and place it in the board in memory.

lvls_counter: Loads the data register from the GPIO SW2-SW5 buttons and determines how many buttons were pressed, then it updates the pointer in memory to the corresponding number.

life_count: Loads a pointer from memory corresponding to the number of lives the user has available. Once this value is determined, a respective value is loaded into the GPIO SW2-SW5 data register to turn on the LEDs (3 -> 7 loaded into data register)

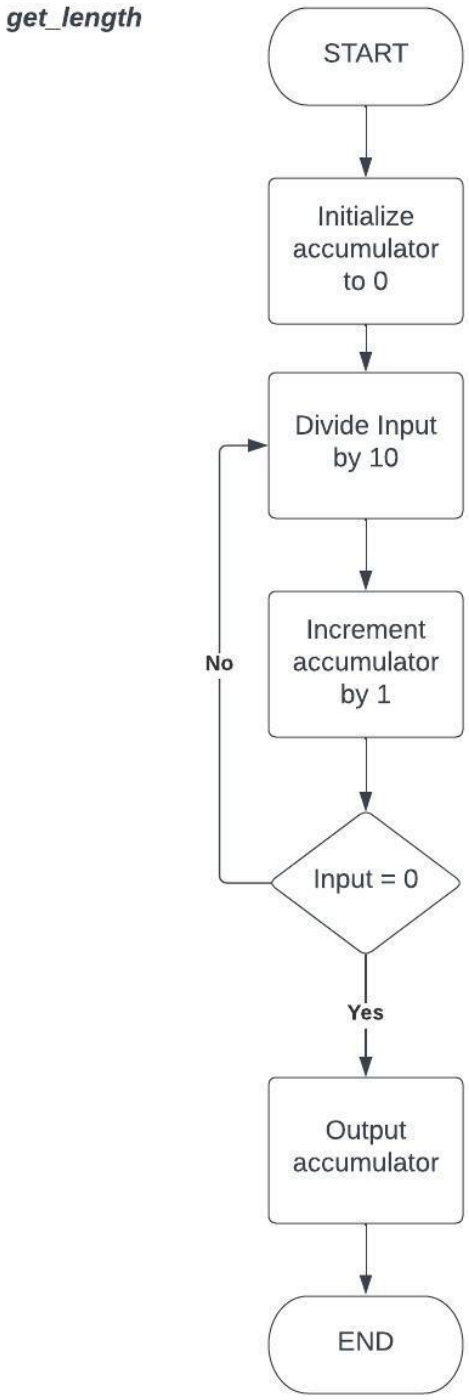
startup: Loads the starting prompts for the user and polls for the user to press S to start. If so, the subroutine exits

advance_check: Checks to see if the user has cleared all the bricks on the board based on the row count and level score. If conditions are met, a boolean value is set to 1 in memory

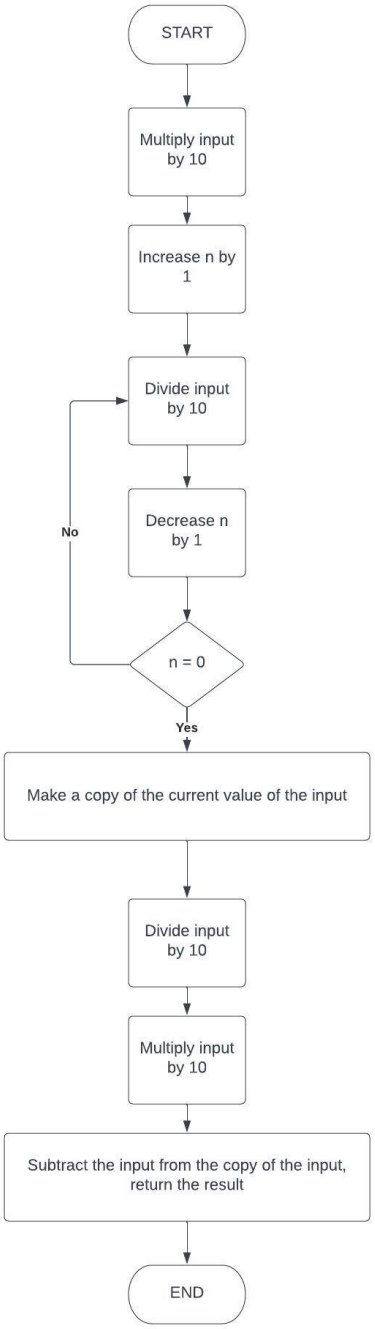
levelup: Speeds up the ball movement, resets the ball and paddle positioning to its initial states, and creates a new board for the user to clear

reset_initial_values: Resets all values in memory to their initial state, such as the life count, level count, boolean values, etc.

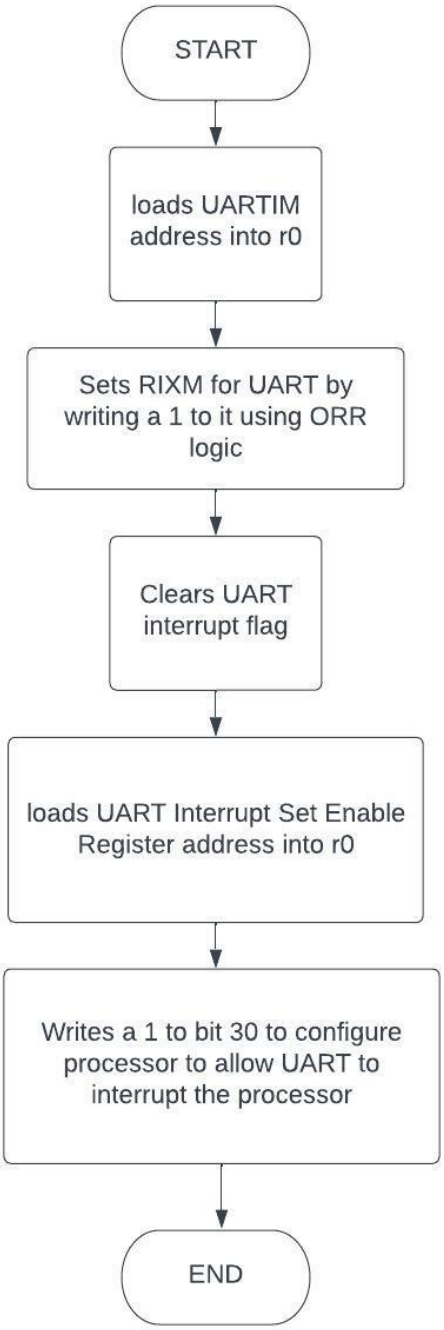
Section 4


get_length
START
Initialize accumulator to 0
Divide Input by 10
Increment accumulator by 1
No
Input = 0
Yes
Output accumulator
END

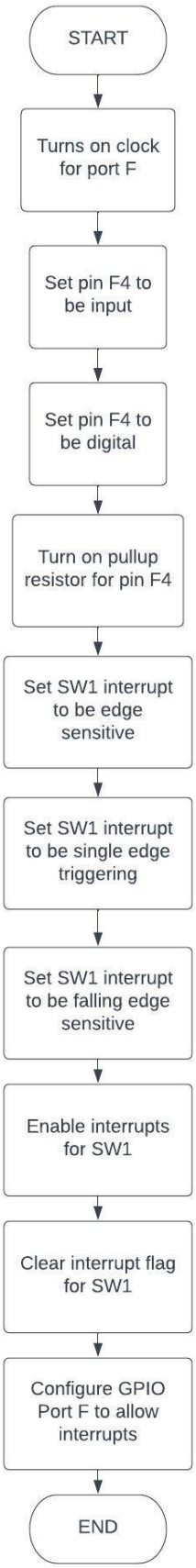
get_nth


START
Multiply input by 10
Increase n by 1
Divide input by 10
Decrease n by 1
No
n = 0
Yes
Make a copy of the current value of the input
Divide input by 10
Multiply input by 10
Subtract the input from the copy of the input, return the result
END

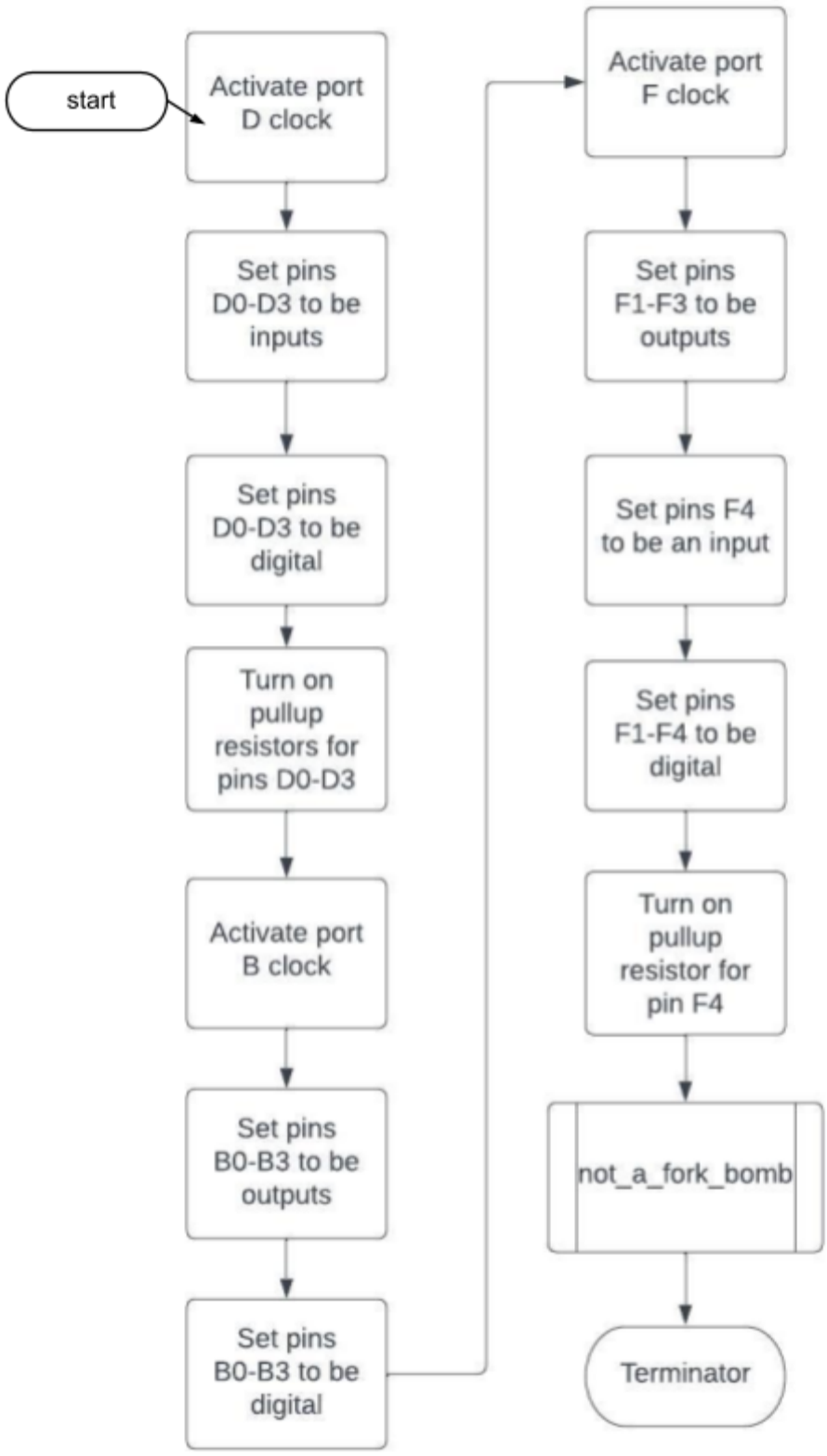
uart_interrupt_init



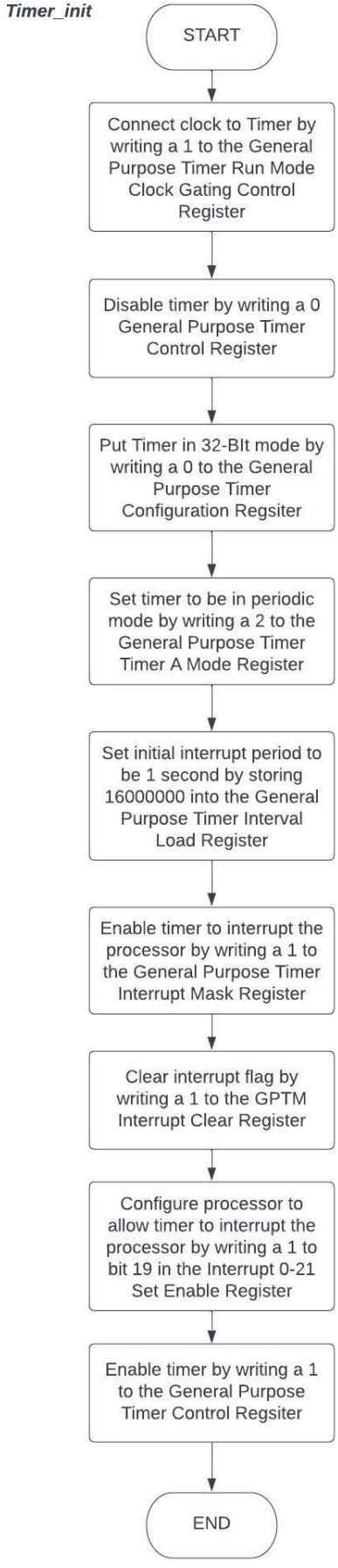
gpio_interrupt_init



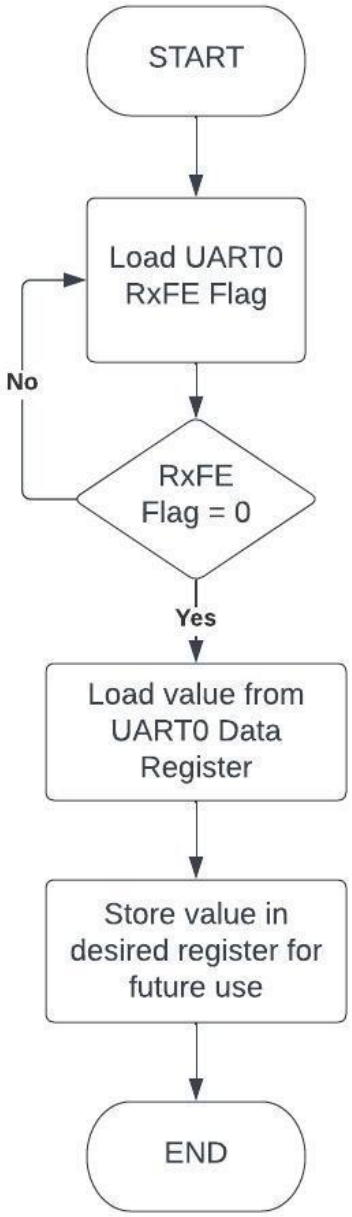
gpio_btn_and_LEDs_init



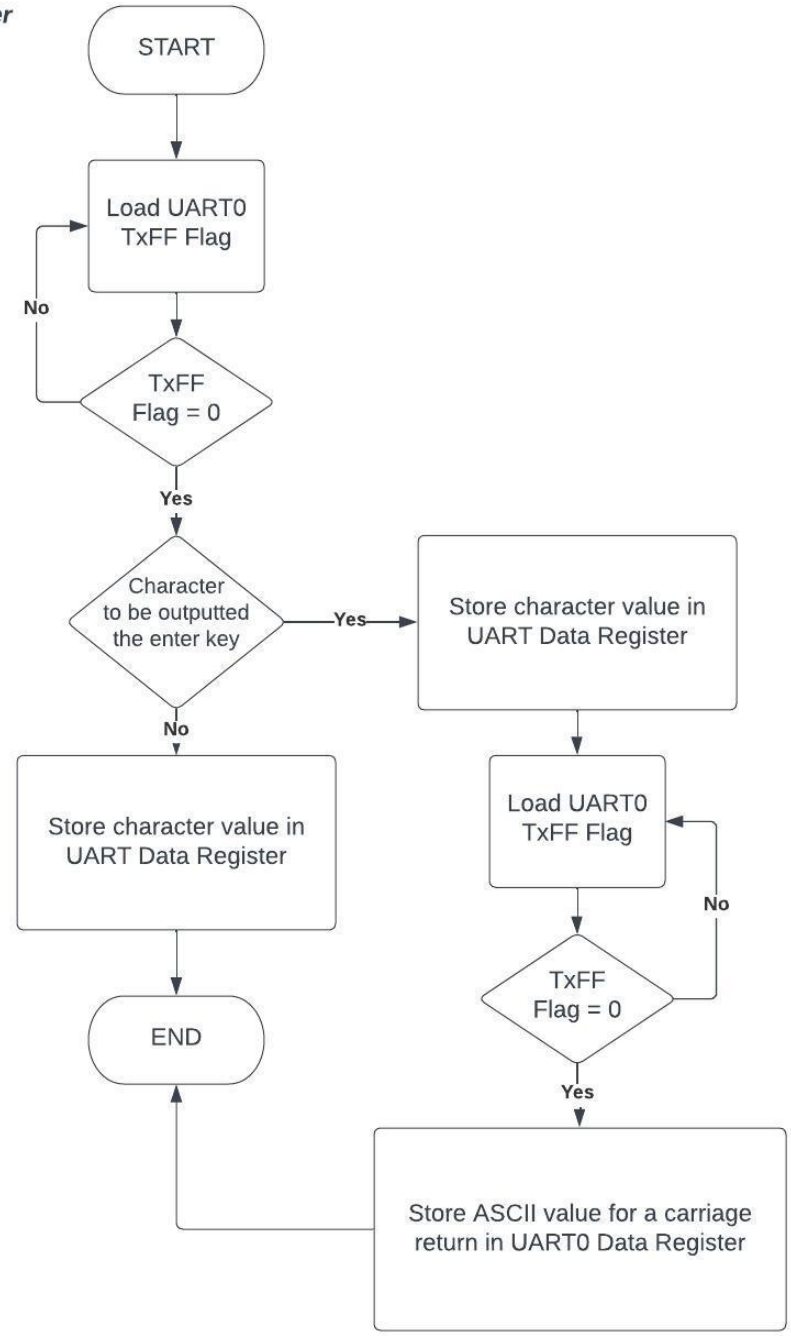
Timer_init


START
Connect clock to Timer by writing a 1 to the General Purpose Timer Run Mode Clock Gating Control Register
Disable timer by writing a 0 General Purpose Timer Control Register
Put Timer in 32-BIt mode by writing a 0 to the General Purpose Timer Configuration Regsiter
Set timer to be in periodic mode by writing a 2 to the General Purpose Timer Timer A Mode Register
Set initial interrupt period to be 1 second by storing 16000000 into the General Purpose Timer Interval Load Register
Enable timer to interrupt the processor by writing a 1 to the General Purpose Timer Interrupt Mask Register
Clear interrupt flag by writing a 1 to the GPTM Interrupt Clear Register
Configure processor to allow timer to interrupt the processor by writing a 1 to bit 19 in the Interrupt 0-21 Set Enable Register
Enable timer by writing a 1 to the General Purpose Timer Control Regsiter
END

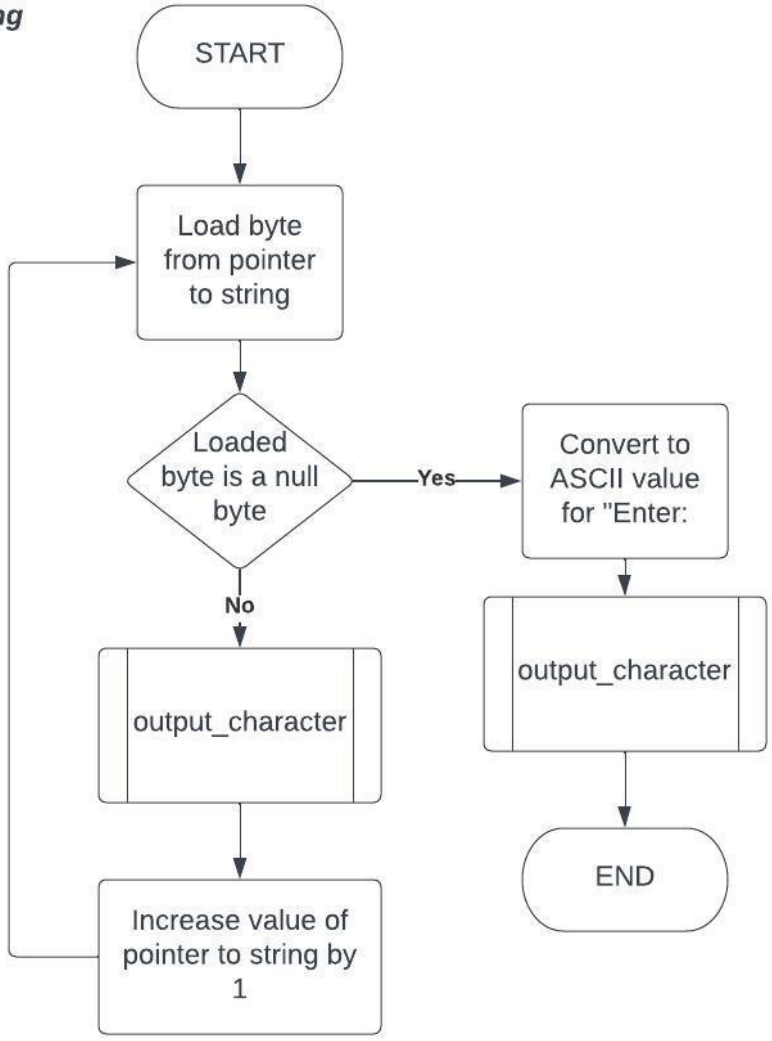
read_character

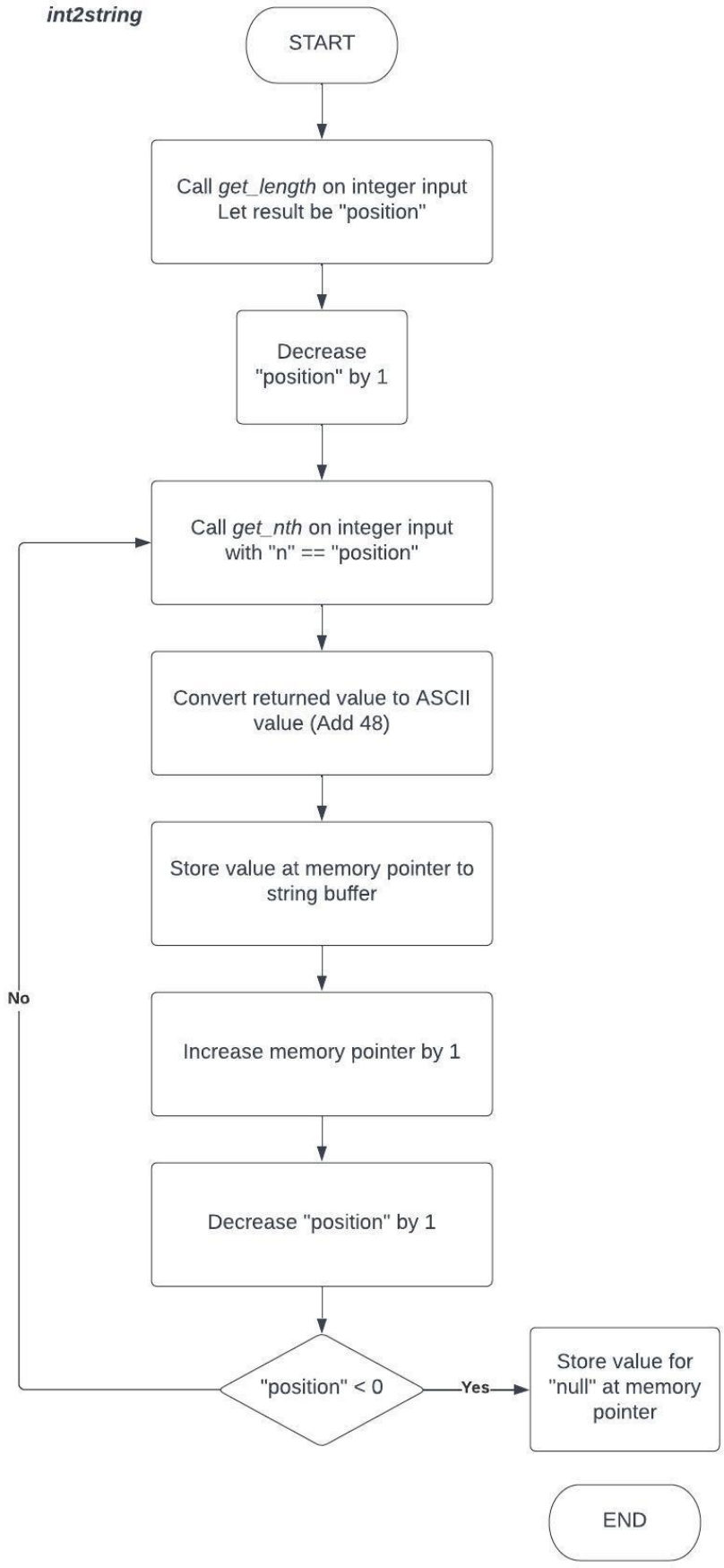


output_character

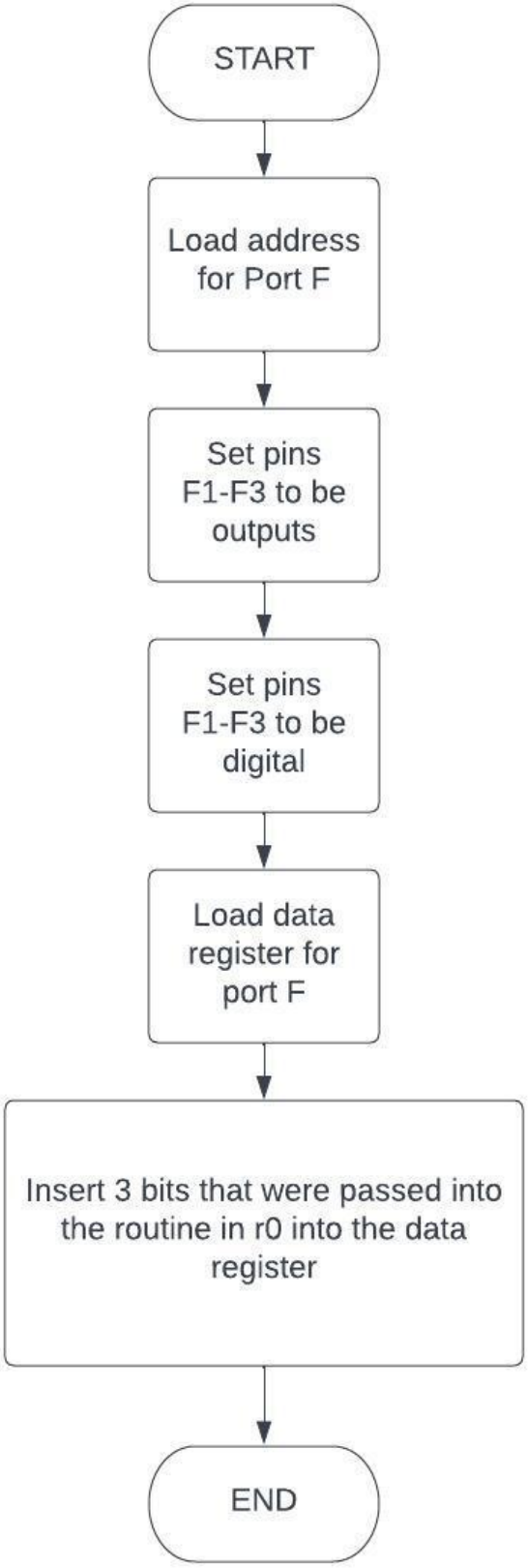


output_string

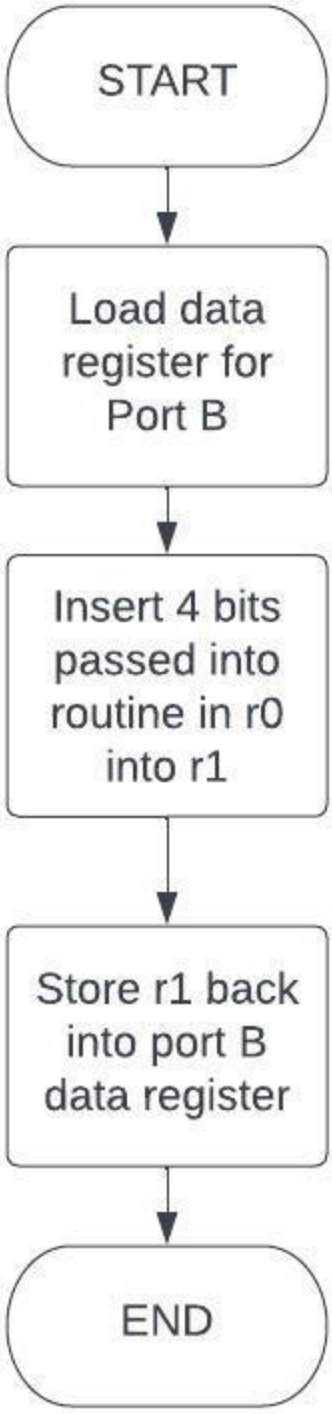



int2string
START
Call get_length on integer input
Let result be "position"
Decrease
"position" by 1
Call get_nth on integer input
with "n" == "position"
Convert returned value to ASCII
value (Add 48)
Store value at memory pointer to
string buffer
Increase memory pointer by 1
Decrease "position" by 1
"position" < 0
No
Yes
Store value for
"null" at memory
pointer
END

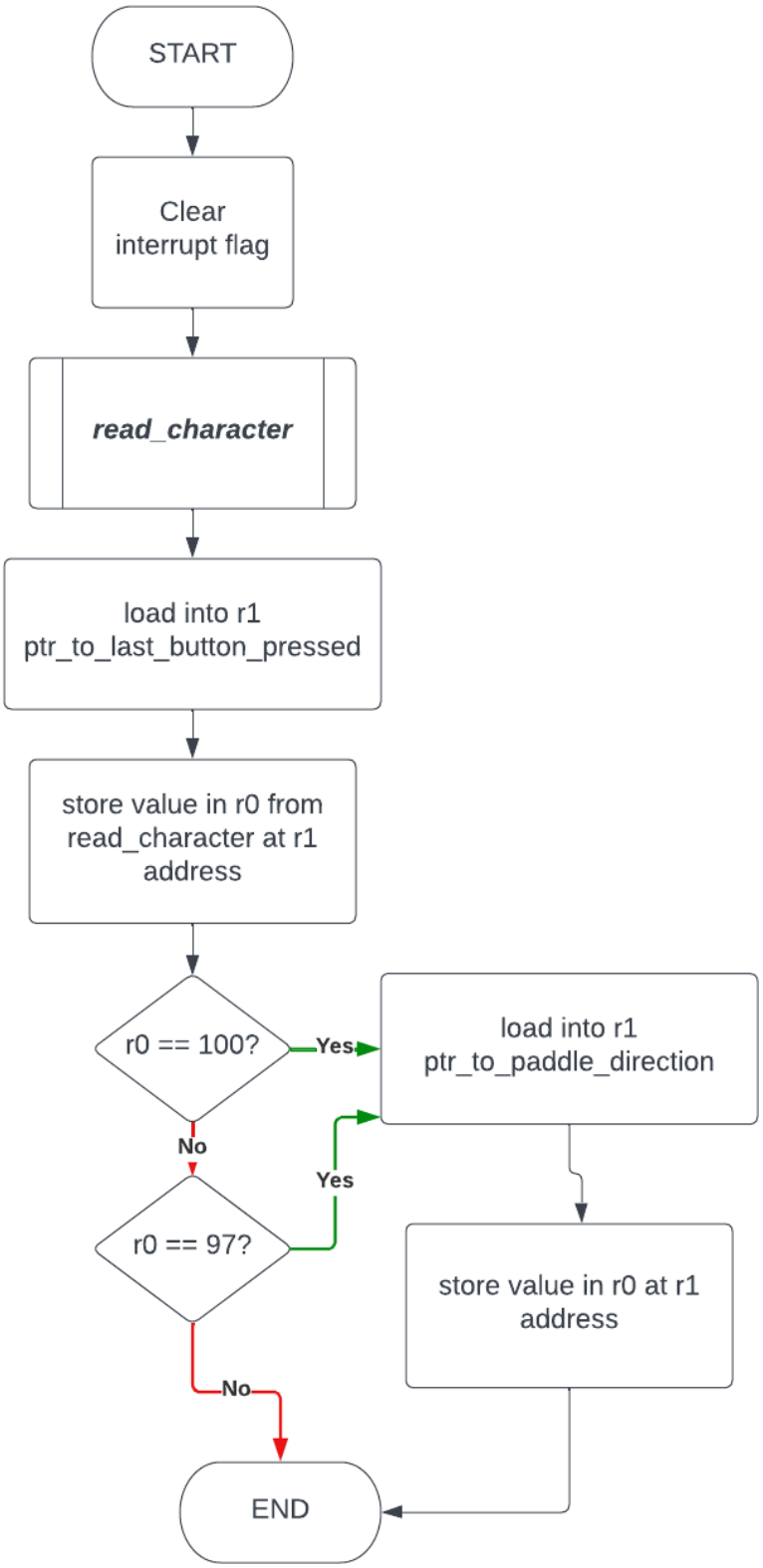
illuminate_RGB_LED



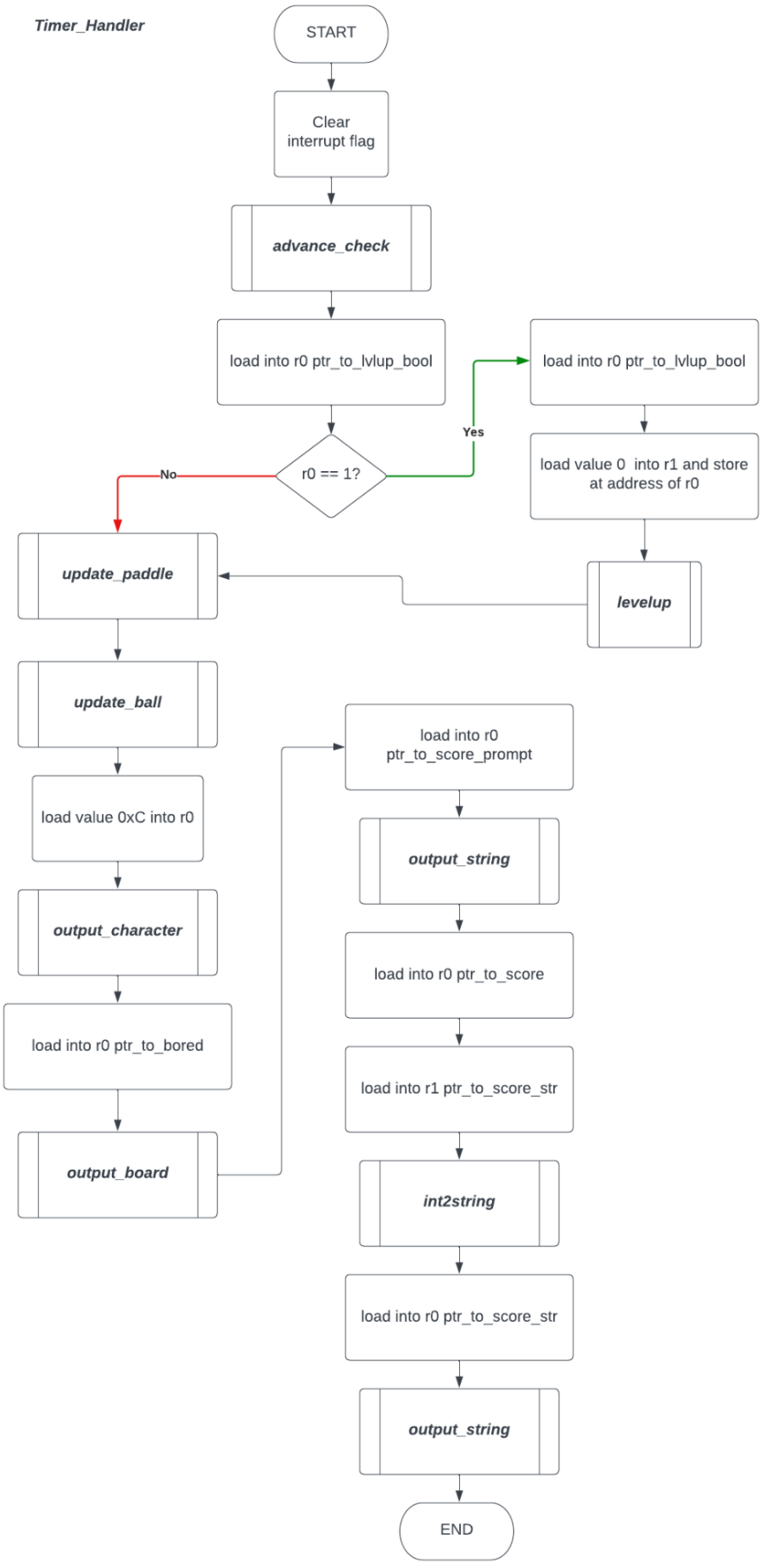
illuminate_LEDs



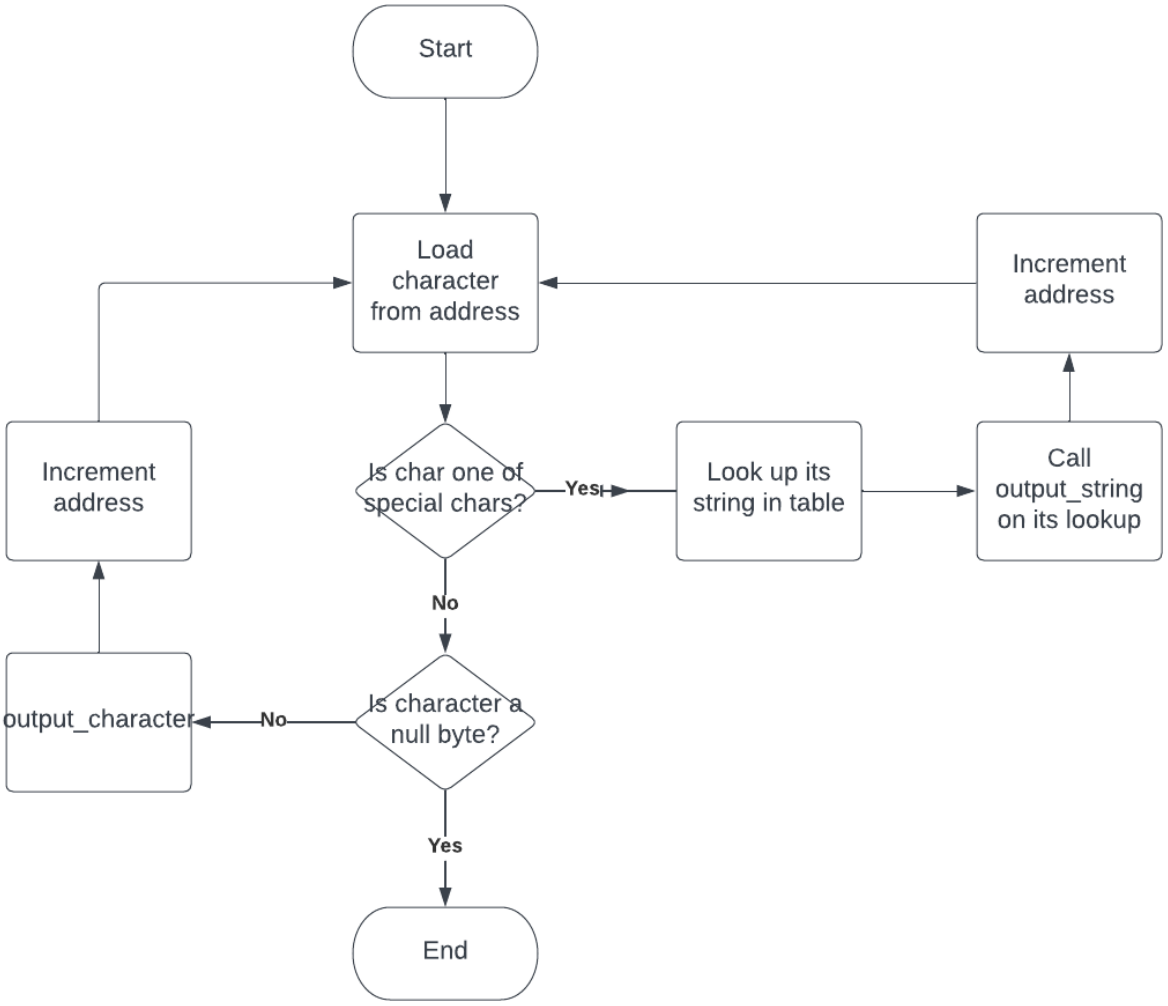
UART0_Handler



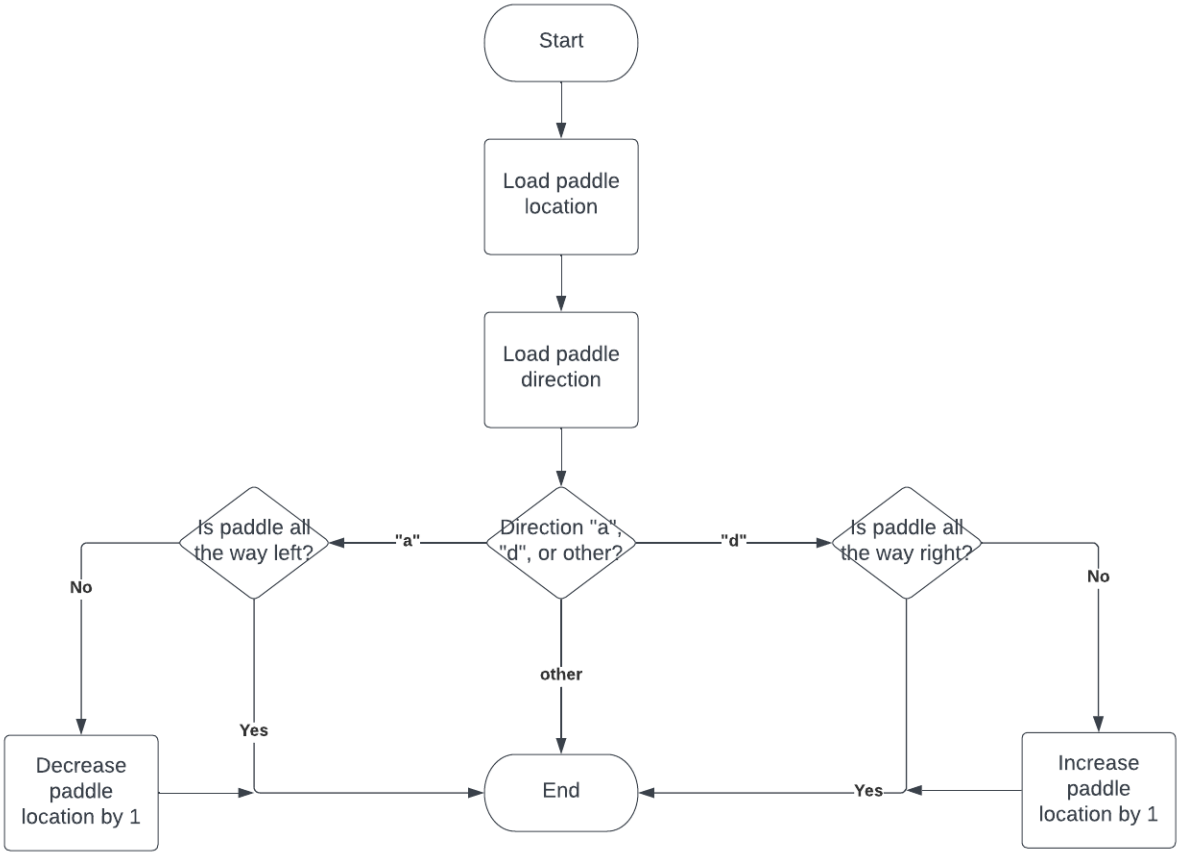

Switch_Handler
START
Clear interrupt flag
load into r0 ptr_to_pause_bool
r0 == 0?
Yes
No
Disable Timer A0 by storing 0 at bit 0
load value 2 into r0
illuminate_RGB_LED
load into r0 ptr_to_pause_bool
load value 1 into r1 and store at address of r0
load into r0 ptr_to_pause_prompt
output_string
END
load into r0 ptr_to_old_colorl
illuminate_RGB_LED
load into r0 ptr_to_pause_bool
load value 0 into r1 and store at address of r0
load value 0xC into r0
output_character
load into r0 ptr_to_bored
output_board
Enable Timer A0 by storing 1 at bit 0


Timer_Handler
START
Clear interrupt flag
advance_check
load into r0 ptr_to_lvlup_bool
r0 == 1?
Yes
No
load into r0 ptr_to_lvlup_bool
load value 0 into r1 and store at address of r0
levelup
update_paddle
update_ball
load value 0xC into r0
output_character
load into r0 ptr_to_bored
output_board
load into r0 ptr_to_score_prompt
output_string
load into r0 ptr_to_score
load into r1 ptr_to_score_str
int2string
load into r0 ptr_to_score_str
output_string
END

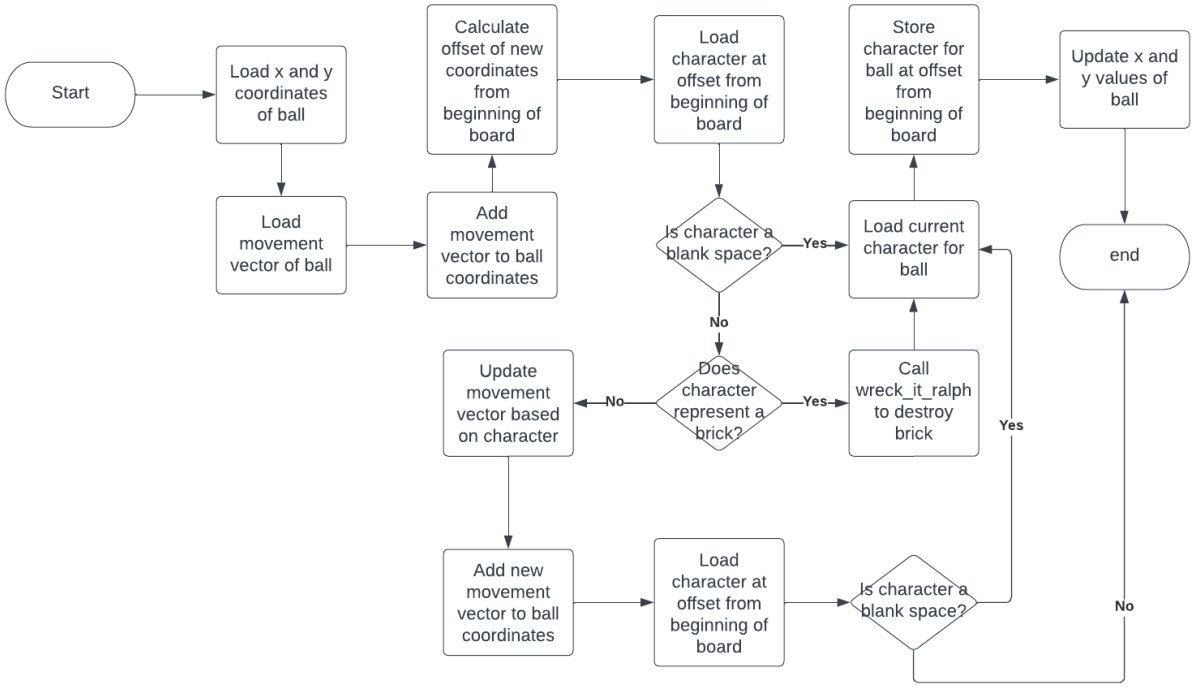
output_board:



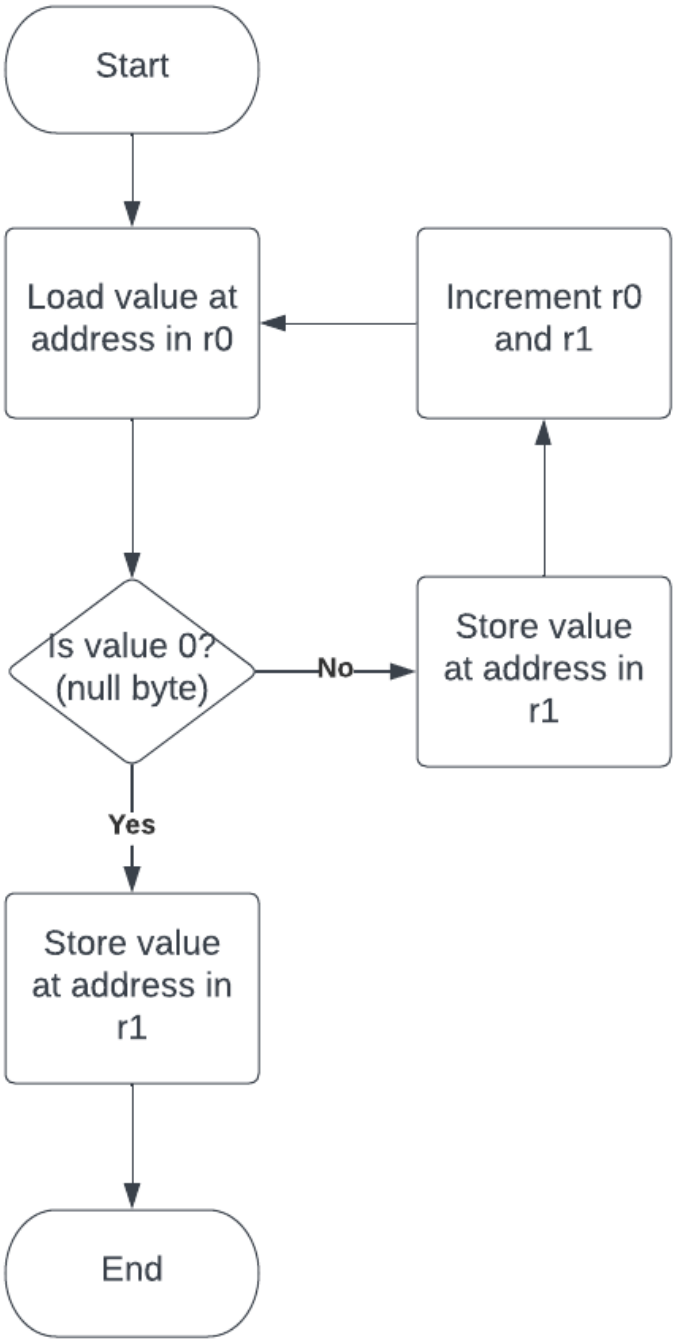
update_paddle:



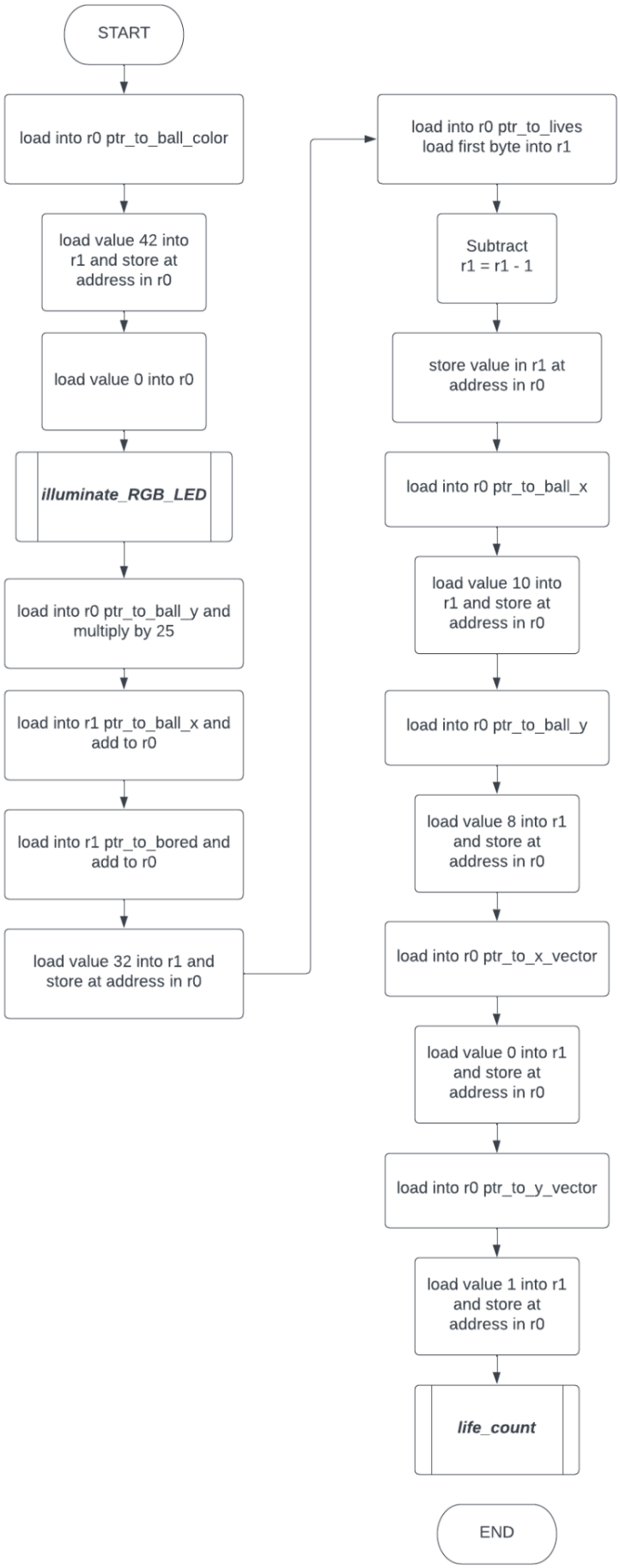
update_ball:

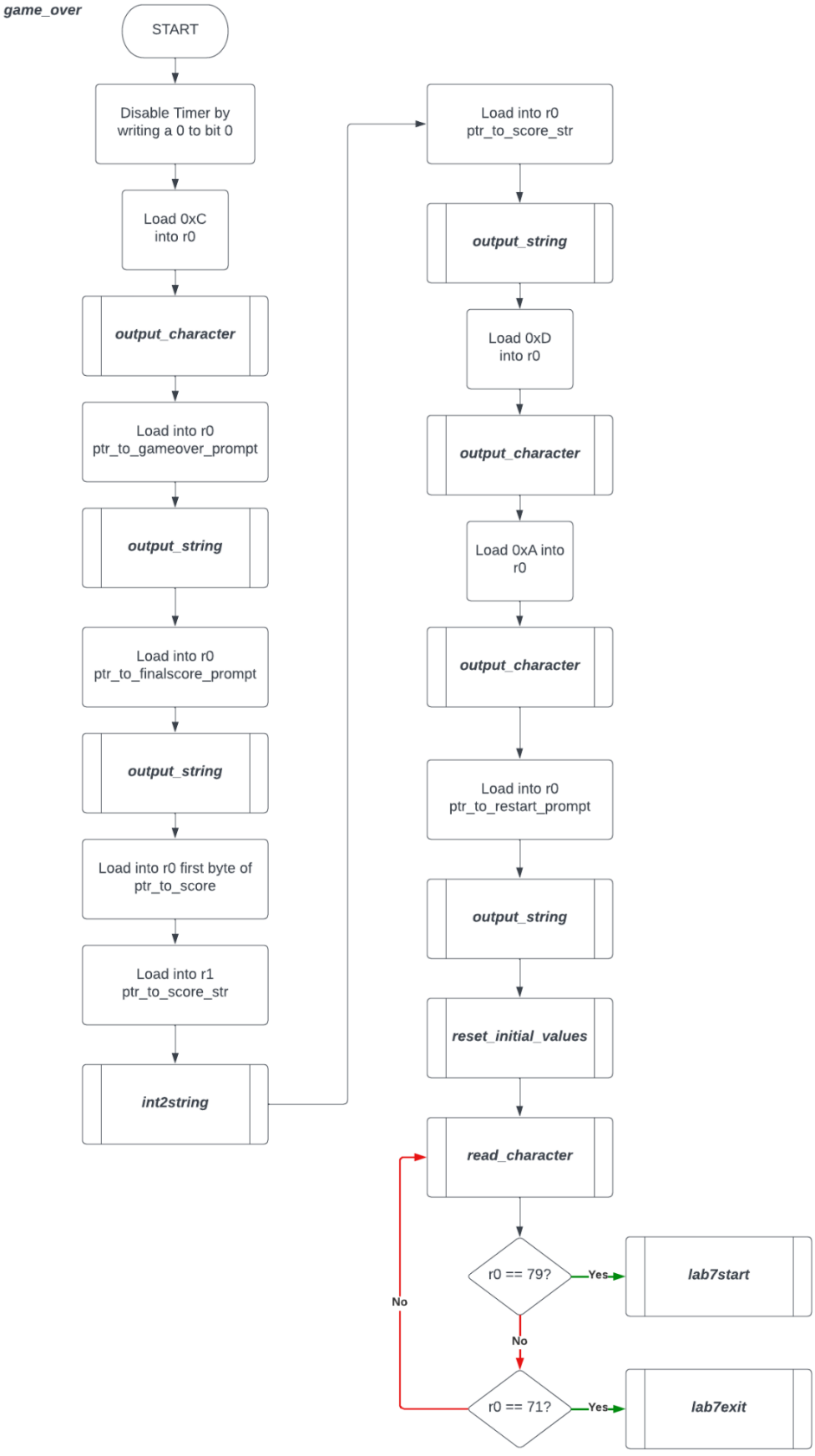

Start
Load x and y coordinates of ball
Load movement vector of ball
Add movement vector to ball coordinates
Calculate offset of new coordinates from beginning of board
Load character at offset from beginning of board
Is character a blank space?
Yes
No
Load current character for ball
Store character for ball at offset from beginning of board
Update x and y values of ball
end
Does character represent a brick?
No
Yes
Update movement vector based on character
Call wreck_it_ralph to destroy brick
Add new movement vector to ball coordinates
Load character at offset from beginning of board
Is character a blank space?
Yes
No

copy_string:

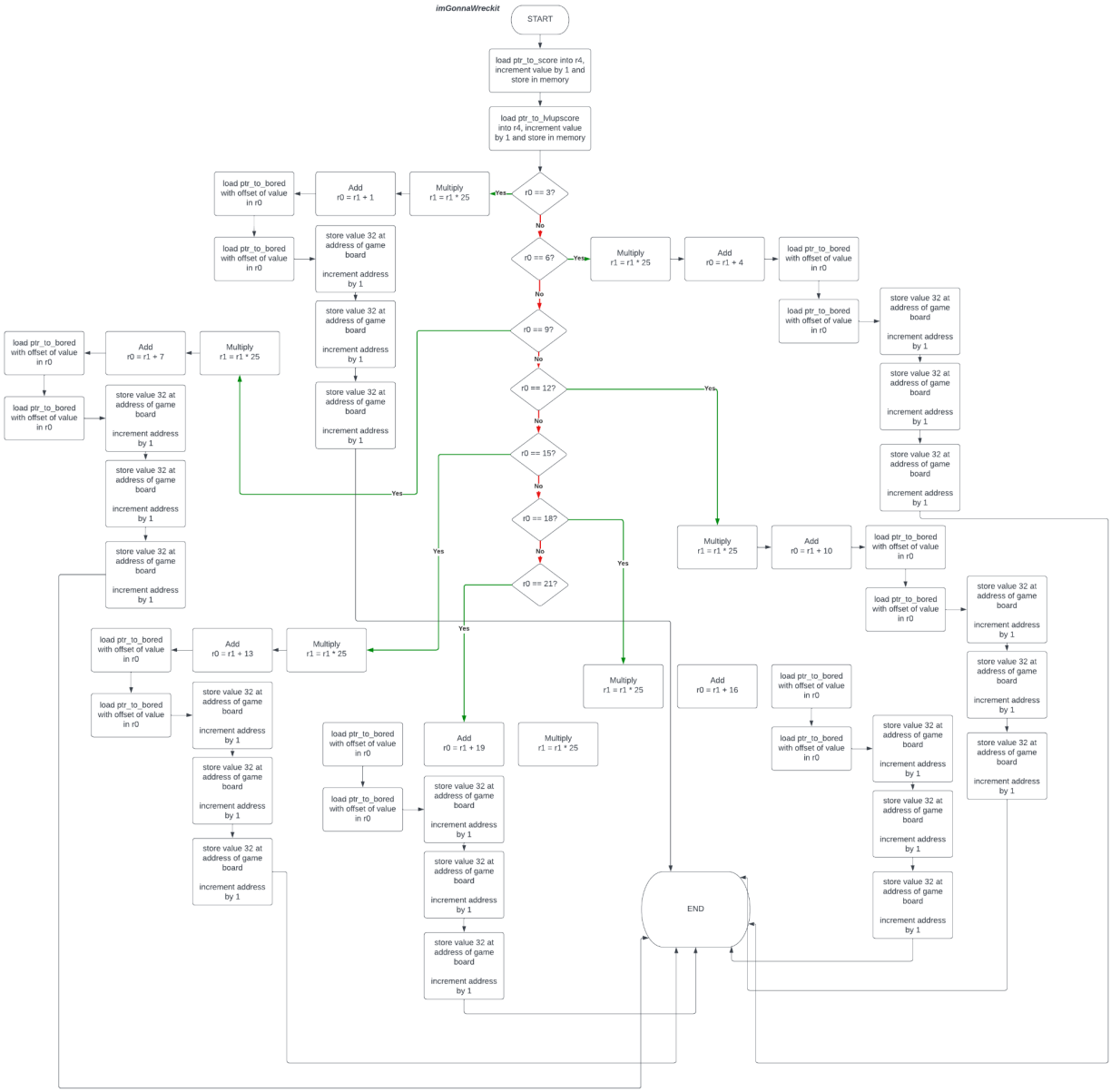

Start
Load value at address in r0
Increment r0 and r1
Is value 0? (null byte)
No
Store value at address in r1
Yes
Store value at address in r1
End

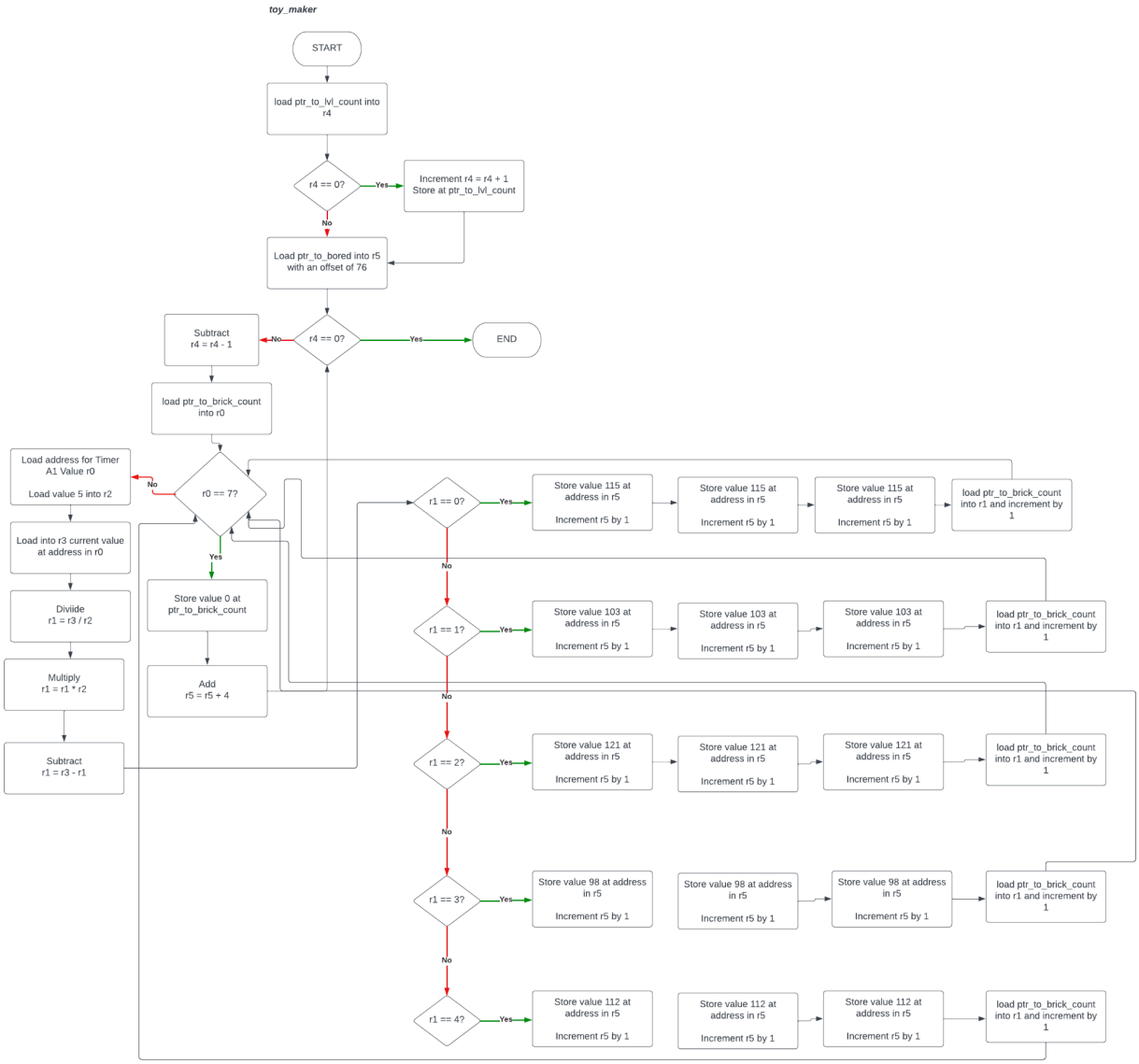
update_OOB


START
load into r0 ptr_to_ball_color
load value 42 into r1 and store at address in r0
load value 0 into r0
illuminate_RGB_LED
load into r0 ptr_to_ball_y and multiply by 25
load into r1 ptr_to_ball_x and add to r0
load into r1 ptr_to_bored and add to r0
load value 32 into r1 and store at address in r0
load into r0 ptr_to_lives load first byte into r1
Subtract r1 = r1 - 1
store value in r1 at address in r0
load into r0 ptr_to_ball_x
load value 10 into r1 and store at address in r0
load into r0 ptr_to_ball_y
load value 8 into r1 and store at address in r0
load into r0 ptr_to_x_vector
load value 0 into r1 and store at address in r0
load into r0 ptr_to_y_vector
load value 1 into r1 and store at address in r0
life_count
END


game_over
START
Disable Timer by writing a 0 to bit 0
Load 0xC into r0
output_character
Load into r0 ptr_to_gameover_prompt
output_string
Load into r0 ptr_to_finalscore_prompt
output_string
Load into r0 first byte of ptr_to_score
Load into r1 ptr_to_score_str
int2string
Load into r0 ptr_to_score_str
output_string
Load 0xD into r0
output_character
Load 0xA into r0
output_character
Load into r0 ptr_to_restart_prompt
output_string
reset_initial_values
read_character
r0 == 79?
Yes
lab7start
No
r0 == 71?
Yes
lab7exit
No

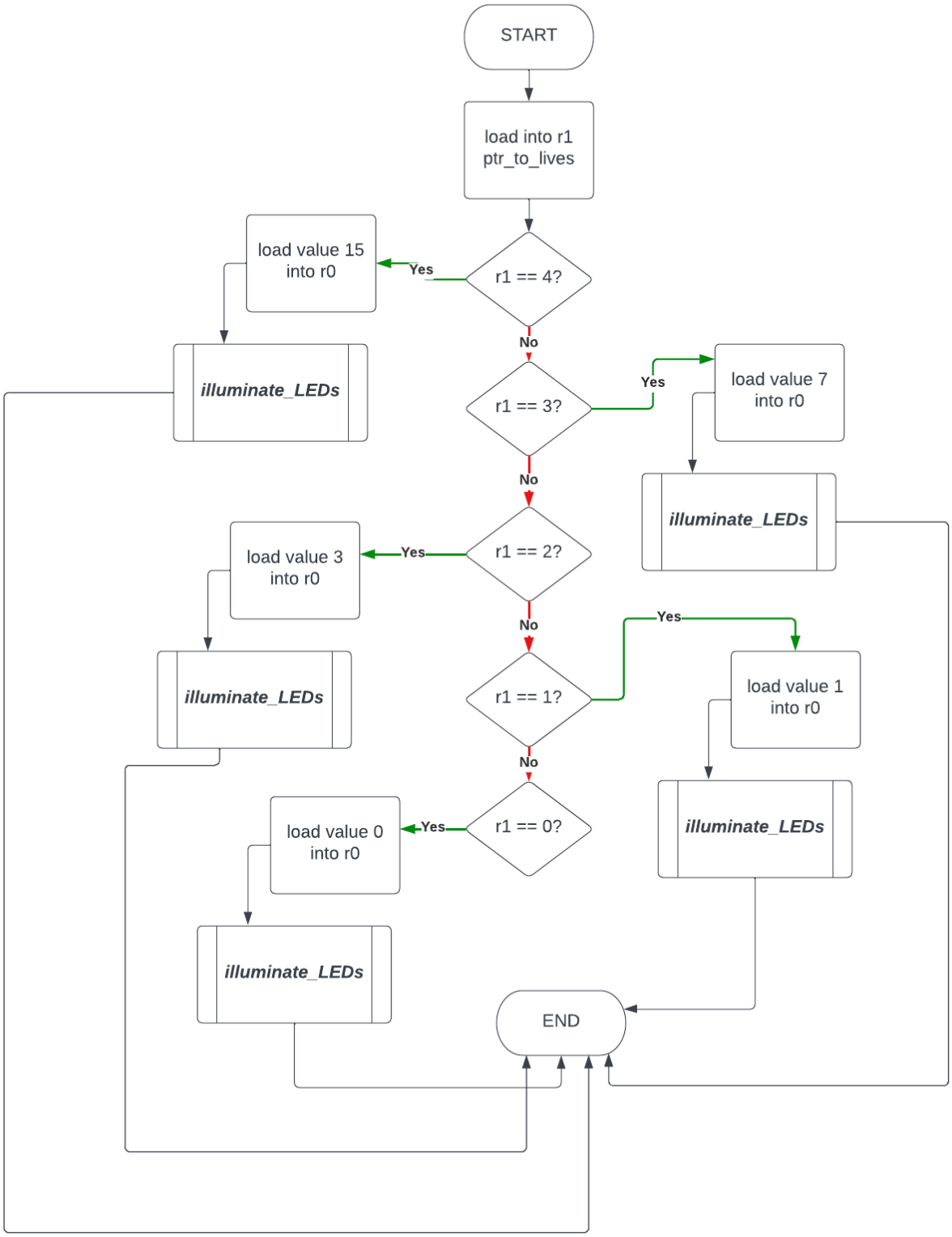

wreck_it_ralph
START
load value 25 into r2
Multiply r1 = r1 * r2
Add r0 = r0 + r1
load into r1 first byte of ptr_to_bored with an offset of value in r0
r2 == 114?
Yes
load value 82 into r1 and stiore at ptr_to_ball_color
load value 1 into r0 and stiore at ptr_to_old_color
illuminate_RGB_LED
No
r2 == 103?
Yes
load value 71 into r1 and stiore at ptr_to_ball_color
load value 4 into r0 and stiore at ptr_to_old_color
illuminate_RGB_LED
No
r2 == 98?
Yes
load value 66 into r1 and stiore at ptr_to_ball_color
load value 2 into r0 and stiore at ptr_to_old_color
illuminate_RGB_LED
No
r2 == 121?
Yes
load value 89 into r1 and stiore at ptr_to_ball_color
load value 5 into r0 and stiore at ptr_to_old_color
illuminate_RGB_LED
No
r2 == 112?
Yes
load value 86 into r1 and stiore at ptr_to_ball_color
load value 3 into r0 and stiore at ptr_to_old_color
illuminate_RGB_LED
imGonnaWreckIt
END


imGonnaWreckit
START
load ptr_to_score into r4, increment value by 1 and store in memory
load ptr_to_lvlupscore into r4, increment value by 1 and store in memory
r0 == 3?
Yes
Multiply r1 = r1 * 25
Add r0 = r1 + 1
load ptr_to_bored with offset of value in r0
load ptr_to_bored with offset of value in r0
store value 32 at address of game board increment address by 1
No
r0 == 6?
Yes
Multiply r1 = r1 * 25
Add r0 = r1 + 4
load ptr_to_bored with offset of value in r0
load ptr_to_bored with offset of value in r0
store value 32 at address of game board increment address by 1
store value 32 at address of game board increment address by 1
store value 32 at address of game board increment address by 1
No
r0 == 9?
Yes
Multiply r1 = r1 * 25
Add r0 = r1 + 7
load ptr_to_bored with offset of value in r0
load ptr_to_bored with offset of value in r0
store value 32 at address of game board increment address by 1
store value 32 at address of game board increment address by 1
store value 32 at address of game board increment address by 1
store value 32 at address of game board increment address by 1
No
r0 == 12?
Yes
Multiply r1 = r1 * 25
Add r0 = r1 + 10
load ptr_to_bored with offset of value in r0
load ptr_to_bored with offset of value in r0
store value 32 at address of game board increment address by 1
store value 32 at address of game board increment address by 1
store value 32 at address of game board increment address by 1
No
r0 == 15?
Yes
Multiply r1 = r1 * 25
Add r0 = r1 + 13
load ptr_to_bored with offset of value in r0
load ptr_to_bored with offset of value in r0
store value 32 at address of game board increment address by 1
store value 32 at address of game board increment address by 1
store value 32 at address of game board increment address by 1
No
r0 == 18?
Yes
Multiply r1 = r1 * 25
Add r0 = r1 + 16
load ptr_to_bored with offset of value in r0
load ptr_to_bored with offset of value in r0
store value 32 at address of game board increment address by 1
store value 32 at address of game board increment address by 1
store value 32 at address of game board increment address by 1
No
r0 == 21?
Yes
Multiply r1 = r1 * 25
Add r0 = r1 + 19
load ptr_to_bored with offset of value in r0
load ptr_to_bored with offset of value in r0
store value 32 at address of game board increment address by 1
store value 32 at address of game board increment address by 1
store value 32 at address of game board increment address by 1
END

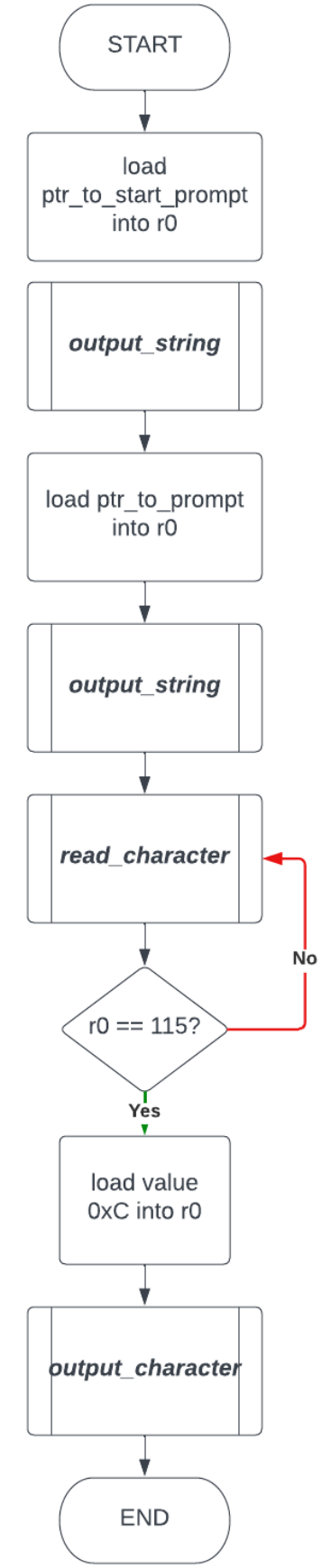

toy_maker
START
load ptr_to_lvl_count into r4
r4 == 0?
Yes
Increment r4 = r4 + 1
Store at ptr_to_lvl_count
No
Load ptr_to_bored into r5 with an offset of 76
Subtract r4 = r4 - 1
No
r4 == 0?
Yes
END
load ptr_to_brick_count into r0
Load address for Timer A1 Value r0
Load value 5 into r2
No
r0 == 7?
Yes
Load into r3 current value at address in r0
Diviide r1 = r3 / r2
Multiply r1 = r1 * r2
Subtract r1 = r3 - r1
Store value 0 at ptr_to_brick_count
Add r5 = r5 + 4
r1 == 0?
Yes
Store value 115 at address in r5
Increment r5 by 1
Store value 115 at address in r5
Increment r5 by 1
Store value 115 at address in r5
Increment r5 by 1
load ptr_to_brick_count into r1 and increment by 1
No
r1 == 1?
Yes
Store value 103 at address in r5
Increment r5 by 1
Store value 103 at address in r5
Increment r5 by 1
Store value 103 at address in r5
Increment r5 by 1
load ptr_to_brick_count into r1 and increment by 1
No
r1 == 2?
Yes
Store value 121 at address in r5
Increment r5 by 1
Store value 121 at address in r5
Increment r5 by 1
Store value 121 at address in r5
Increment r5 by 1
load ptr_to_brick_count into r1 and increment by 1
No
r1 == 3?
Yes
Store value 98 at address in r5
Increment r5 by 1
Store value 98 at address in r5
Increment r5 by 1
Store value 98 at address in r5
Increment r5 by 1
load ptr_to_brick_count into r1 and increment by 1
No
r1 == 4?
Yes
Store value 112 at address in r5
Increment r5 by 1
Store value 112 at address in r5
Increment r5 by 1
Store value 112 at address in r5
Increment r5 by 1
load ptr_to_brick_count into r1 and increment by 1

\

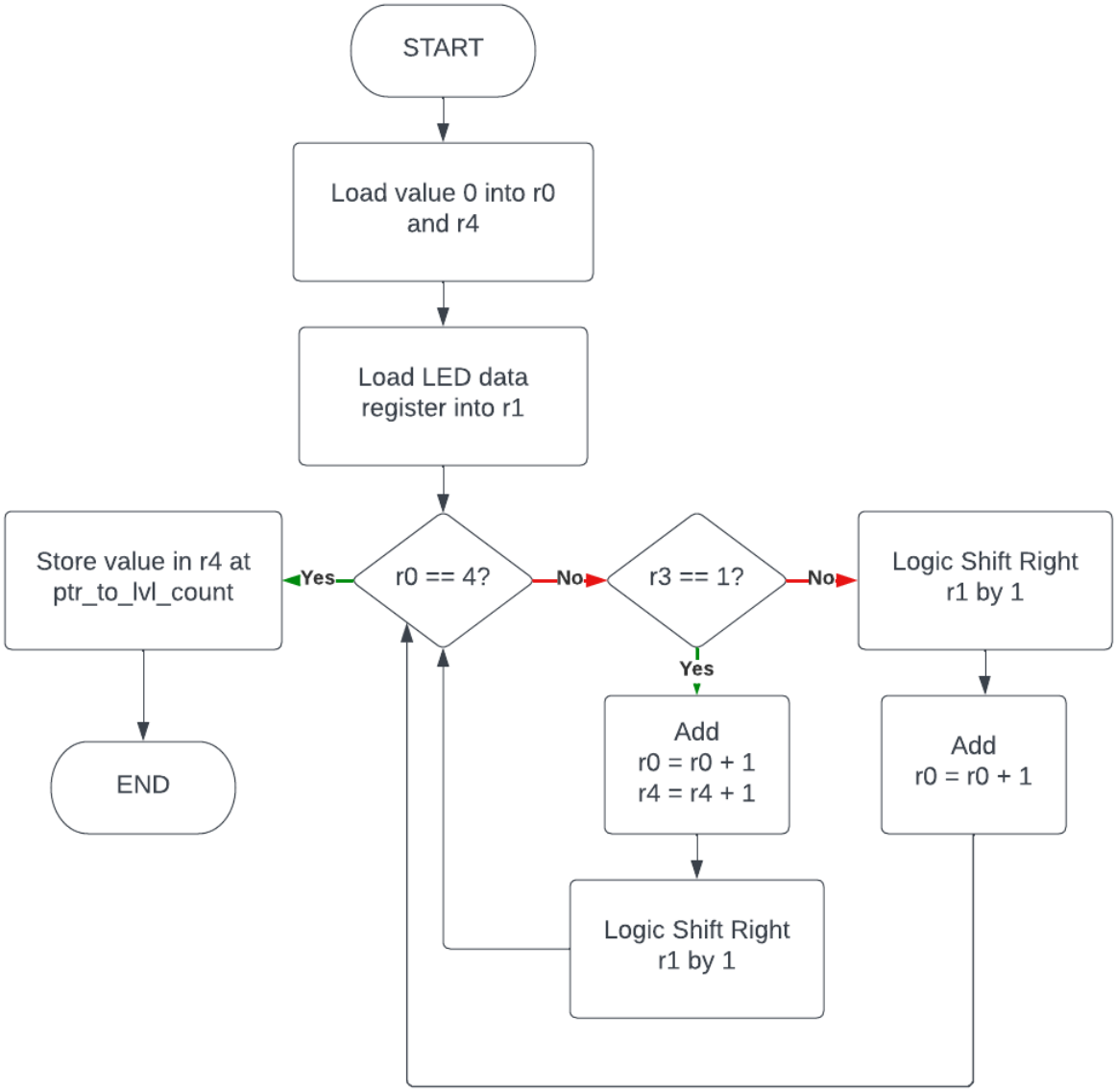
life_count


START
load into r1
ptr_to_lives
r1 == 4?
Yes
load value 15
into r0
No
illuminate_LEDs
r1 == 3?
Yes
load value 7
into r0
No
illuminate_LEDs
r1 == 2?
Yes
load value 3
into r0
No
illuminate_LEDs
r1 == 1?
Yes
load value 1
into r0
No
illuminate_LEDs
r1 == 0?
Yes
load value 0
into r0
illuminate_LEDs
END

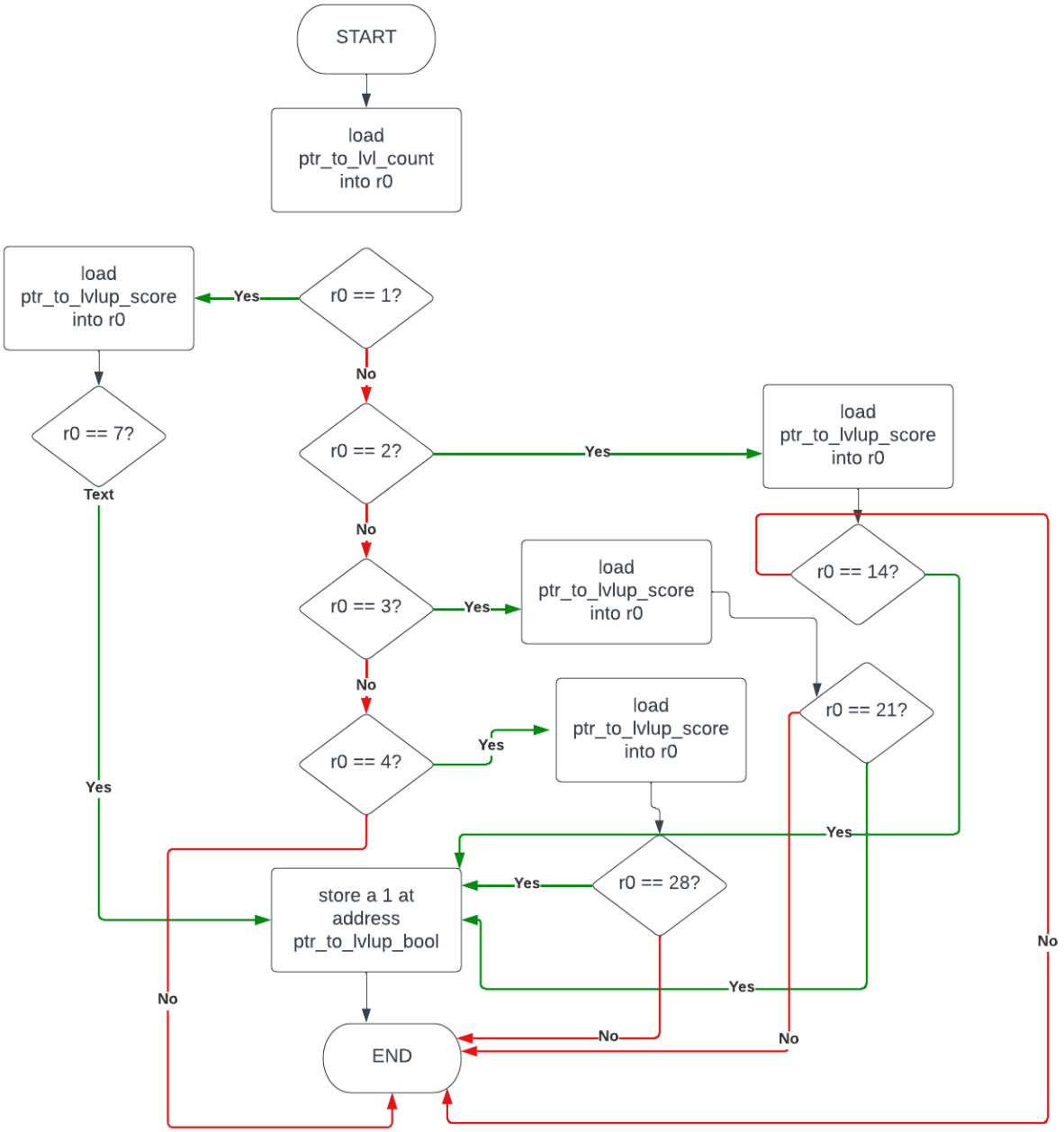
life_count

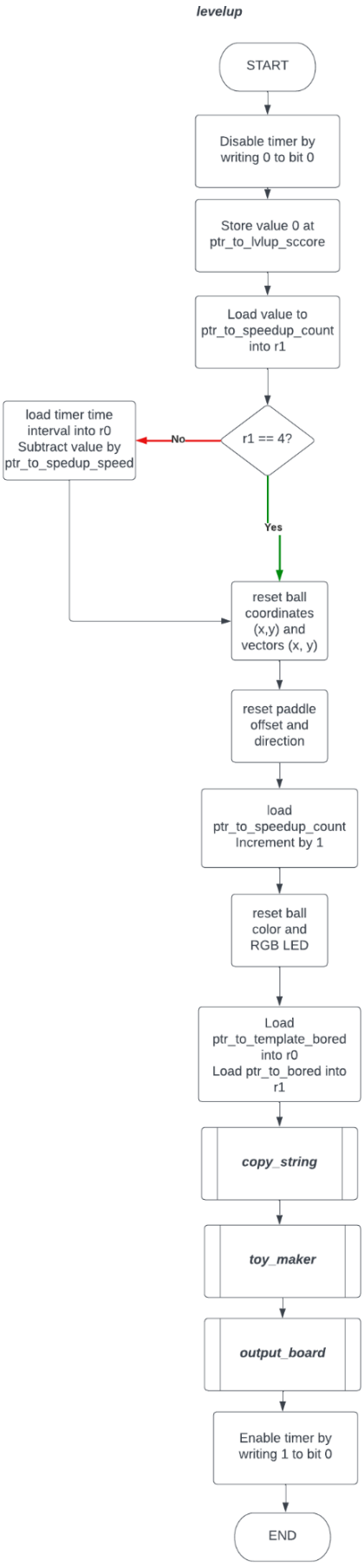

START
load ptr_to_start_prompt into r0
output_string
load ptr_to_prompt into r0
output_string
read_character
r0 == 115?
No
Yes
load value 0xC into r0
output_character
END

lvls_counter



advanceCheck




levelup
START
Disable timer by writing 0 to bit 0
Store value 0 at ptr_to_lvlup_sccore
Load value to ptr_to_speedup_count into r1
r1 == 4?
No
load timer time interval into r0
Subtract value by ptr_to_spedup_speed
Yes
reset ball coordinates (x,y) and vectors (x, y)
reset paddle offset and direction
load ptr_to_speedup_count Increment by 1
reset ball color and RGB LED
Load ptr_to_template_bored into r0
Load ptr_to_bored into r1
copy_string
toy_maker
output_board
Enable timer by writing 1 to bit 0
END

reset_initial_values: