Lab #7
CSE 379
Xudong Liu, Alex Tisdale
xliu243, atisdale
Lab Section: R4
04-26-2024

Table of Contents

Division of Work

Both partners of ours resigned the course at the same time. When we partnered up we discussed how far we both were in the lab and it is very evident that Lui is much more ahead of me in this lab. We have decided to use Lui's code to move forward and build upon the project.

Partner 1: Xudong Liu

- All routines

Partner 2: Alex Tisdale

- Documentation
- Some code sprinkled in that was salvaged from my wreck

The Program

Program Overview

- Start the program.
- Press the spacebar to start the game.
- Use the w,a,s,d keys to move the character up, left, down, and right respectively.
  - Every time a key is pressed, the move counter will increase
  - Keys are case sensitive
- Everytime you move a square your move count will increase by one.
- You will not be able to move to a square with the same color as the character.
- If you try to move to a square that is the same color and doesn't let you, the move counter will still increase.
- Press the spacebar to swap the color of the character with the color of the square the character is currently on.
- Move the character off of the board to move onto a new side of the cube.
- Press SW1 on the Tiva Board to pause the game.
- When the game is restarted, a new game board will show with a different color configuration. Your previous progress will be lost.
- When resumed, the board and color configuration will remain as it did before you pressed pause.
- When quit is pressed, you will be prompted that the game is over.
- You can press one of the 4 push buttons on the alice board before you start the game to select a game-mode(SW1: Timed 100, SW2: Timed 200, SW3: Timed 300, SW4: Unlimited Time).
- There is a timer next to the move counter that counts up and stops at its respective number depending on the game-mode 1,2,or 3, or infinitely counts up until the game is won for game-mode 4.
- In the event of the first three game-modes, when time runs out, the player will be shown a "Game Over" screen
- In the event where all sides of the cube each hold the same color, the player will be shown a "Game Over" screen.

Program Summary

Creating the retro video game, Video Cube, which when the game is started shows a 3-by-3 square grid of randomized colors(Blue,Red,Yellow,White,Magenta,Green) with a smaller colored square, which denotes the character, in the center square. Notice the character is a different color than the square it is on. There are 5 other grids –we'll call faces– that create a full cube. Each face has a different color configuration. You can use the w,a,s,d keys to move up, left, down, right, respectively. Moving "off of" the face will move the character to the next face in respect to which side the character moves off of. For example moving off of the right side of the board will move you to the face that is to the right of the face you were currently on. The goal of the game is to move your character, "pick-up" and "place" the colors of each square on the faces to make it so that each face has all 9 of its squares the same color. Pressing SW1 on the Tiva board will show you a pause screen if you want to quit, restart, or just stop playing for a bit to resume later. If you want to set a timer for your game you can press one of the four push buttons on the Alice Board. If you want to play the game for only 100 seconds: press button 1, 200 seconds: button 2, 300 seconds: button 3, or play indefinitely until the board is complete: button 4.

Subroutine Descriptions

Lab7

Lab7 does not take in any arguments and does not return anything. This routine starts by setting the default values of the ameStateFlag, scoreValue, timeValue, and the cursor position. It then branch and links to make_random_cube that randomly places the colors on the game cube. Itthen enters a loop which checks the game state. If the game state is equal to 5 then that means the game is over and will exit the loop and end the program. If the gamestate is any other value then the program will continue looping.

---

timer_handler

Timer_handler does not take in any arguments and does not return any value. This routine starts by disabling all interrupts and then clearing the timer interrupt. Then two Branch and links are made, one to get the input value, and another to update the input state. A compare is made on the input state and if it is equal to 0x30 the program branch and links to set_updateOn. If it is not 0x30, the program will continue on to load firstFlag and if firstFlag is 0x30, a branch and link is made to set_updateOn. If it is not equal to 0x30 then the program proceeds and sets first flag to 0x30 followed by enabling all interrupts and ending the routine.

---

set_updateOn

set_updateOn does not take in any arguments and does not return any value. Set_updateOn loads the gameStateFlag and checks if it is 0, 1, 2, 3, or 4. If the game state is 0, a Branch and link is made to static_0. If the game state is 1, a Branch and link is made to character_move_1. If the game state is 2, a Branch and link is made to cube_rota_2. If the game state is 3, a Branch and link is made to pause_3. If the game state is 4, a Branch and link is made to game_over_4. Each branch and link returns and then the routine is exited.

---

static_0

set_updateOn does not take in any arguments and does not return any value. The routine loads a pointer for location into registers r10(x) and r11(y) and also loads the current input value into r12. The main function of this routine is to compare the value of r12. If r12 equals 0x77(w) a branch is made to static_0_up which initially checks if r11 is equal to 0x30 and if it is then if branches to handle the rotation of the cube's faces that arent visibly seen and in this case will exit the routine when finished. If r11 is not equal to 0x30 the routine is continued to handle the upwards movement of the player by loading the location of the cursor, subtracting both registers by 0x30, multiplying the second location by the first location, adding the two locations, and subtracting that sum by 3. The character color is then loaded into r9. A branch and link is made to take_face. R8 is then compared to r9 and if they are equal, the routine exits, if it isn't, r11 is subtracted by 1 and stored into the pointer location and then exited.if r12 is equal to 0x73(s), the same concept is done as previous but now working to move the player down. If r12 is equal to 0x61(a) the same process is made but now moving left. And if r12 is equal to 0x64(d), you guessed it, the same thing is don't but now to move the character right. If r12 is equal to 0x20(space) then it deals with swapping the player and squares color. This is done by getting the cursors location, subtracting both by 0x30, multiplying them together, adding them together, subtracting that sum by 3, and loading the character color into r9 before branching and linking to take_face.Finallythe value in r7 is then stored as the character color.

---

static_face_draw

static_face_draw takes in r0 as an argument and does not return a value. The point of this routine is to "draw" the face of the cube that is being shown in putty including the colors and the character position on the face. The routine does this by grabbing each color by branching and linking to take_face in order to grab the colors of the face and store them into r5. From here, it loads from 9 offsets of r5 and branch and links to draw_3x3_block, which draws the colors onto the screen, each time. Once it has printed the colors of the face, it then checks ptr_to_location which holds the x and y value of the character. Once it has the location of the character it starts by checking the x value and whether it is 0x30, 0x31, or 0x32, based on these values it determines where the character should be placed on the face. After it finds the x value, it repeats the same function

but now with the y value. Once it has found both x and y coordinates of the character, the routine branch and links to draw_single_squire which will print the character and its color.

---

step_counter_draw

All step_counter_daw does is print the move counter below the board. It loads the score value and turns it into a string to be able to print to putty next to text that says “SCORE: ”.

---

draw_3x3_cube

Draw_3x3_cube takes in r0 as an argument and returns r0. This routine checks r0 for 7 values, 8,2,4,10,12,14, and 0. Based on the value of r0, it loads the ansi escape sequence of the color back into r0. Red - 8, Blue - 2, Green - 4, Purple - 10, Yellow - 12, White - 14, Black - 0.

---

draw_single_squire

draw_single_squire has a similar function to draw_3x3_square in that it takes in r0 as an argument and checks its value of 8,2,4,10,12,14, and 0. Red - 8, Blue - 2, Green - 4, Purple - 10, Yellow - 12, White - 14, Black - 0. When that number is hit, the pointer to that associated color is stored into r0

---

make_random_cube

Make_random_cube takes in r0 and r1 as an argument and returns r1 and r3. This routine begins by storing r0 into memory address 0xE000E010. Then sets r0 as 0x000002FA and stores that into the memory address 0xE000E014. ptr_to_favce is loaded into r5, r6-r11 are set to 9, r1 is set to 6, and r12 is set to 0. The routine then enters a loop and begins by branching a linking to div_and_mod with r0 set to the value stored in memory address 0xE000E018. Div_and_mod returns r1 which is then checked for a value 0,1,2,3,4, or 5. 0 - Red, 1 - Blue, 2 - Green, 3 - Purple, 4 - yellow, 5 - white. When r1 = 0, a branch is made to assign_red which begins with comparing r6 to 0 and if they are equal a branch is made to assign_blue. If they are not equal r6 is decremented by 1 and r1 is set to 8 and the routine branches to assign_end. If r1 = 1, a branch is made to assign_blue

which begins with comparing r7 with 0. If they are equal a branch is made to assign_green. If they are not equal r7 is decremented by 1 and r1 is set to 2 and the routine branches to assign_end. If r1 = 2, a branch is made to assign_green which begins with comparing r8 with 0. If they are equal a branch is made to assign_purple. If they are not equal r8 is decremented by 1 and r1 is set to 4 and the routine branches to assign_end. If r1 = 3, a branch is made to assign_purple which begins with comparing r9 with 0. If they are equal a branch is made to assign_yellow. If they are not equal r9 is decremented by 1 and r1 is set to 10 and the routine branches to assign_end. If r1 = 4, a branch is made to assign_yellow which begins with comparing r10 with 0 and the routine branches to assign_end. If they are equal a branch is made to assign_white. If they are not equal r10 is decremented by 1 and r1 is set to 12 and the routine branches to assign_end. If r1 = 5, a branch is made to assign_white which begins with comparing r11 with 0. If they are equal a branch is made to assign_red. If they are not equal r11 is decremented by 1 and r1 is set to 14 and the routine branches to assign_end. As mentioned, each assign-routine ends with branching to assign_end. This sub-routine starts with storing r1 into the memory address at r5 followed by incrementing r5 by 1. Then, increments r12 by 1 and compares it to 54. If r12 is not equal to 54, it loops back to do the entire above process over again. If r12 is equal to 54, the value at ptr_to_face0 is compared to 8. If they are equal a branch is made to set_char_blue, and if they are not equal a branch is made to set_char_red. Set_char_blue stores the value 2 into the memory address stored in ptr_to_charcolor which holds the color of the character. Set_char_red stores the value 8 into the memory address stored in ptr_to_charcolor. Both routines are followed by storing 0 into the memory address 0xE000E010 and setting r1 and r3 to 0. That being said, Based on what r1 is, that color will be stored to be printed to the screen. Using div_and_mod creates this pseudo-random function.

---

get_current_input_value

get_current_input_value does not take in any arguments and does not return any values. The routine begins by loading the value in ptr_to_keySpaceFlag and comparing that value to 0x30. If they are equal a branch is made to get_current_input_value_w. If they are not equal the value of ptr_to_keySpaceState is compared to 0x31. If they are equal a branch is made to get_current_input_value_w. If they are not equal, r11 is set to 0x20 and the routine is exited. Get_current_input_value_w - subroutine starts by loading the value in

ptr_to_keyWFlag and comparing that value to 0x30. If they are equal a branch is made to get_current_input_value_a. If they are not equal the value of ptr_to_keyWState is compared to 0x31. If they are equal a branch is made to get_current_input_value_a. If they are not equal, r11 is set to 0x77 and the routine is exited.
Get_current_input_value_a - subroutine starts by loading the value in ptr_to_keyAFlag and comparing that value to 0x30. If they are equal a branch is made to get_current_input_value_s. If they are not equal the value of ptr_to_keyAState is compared to 0x31. If they are equal a branch is made to get_current_input_value_s. If they are not equal, r11 is set to 0x61 and the routine is exited.
Get_current_input_value_s - subroutine starts by loading the value in ptr_to_keySFlag and comparing that value to 0x30. If they are equal a branch is made to get_current_input_value_d. If they are not equal the value of ptr_to_keySState is compared to 0x31. If they are equal a branch is made to get_current_input_value_d. If they are not equal, r11 is set to 0x73 and the routine is exited.
Get_current_input_value_d - subroutine starts by loading the value in ptr_to_keyDFlag and comparing that value to 0x30. If they are equal a branch is made to get_current_input_value_set_0. If they are not equal the value of ptr_to_keyDState is compared to 0x31. If they are equal a branch is made to get_current_input_value_set_0. If they are not equal, r11 is set to 0x64 and the routine is exited.
Get_current_input_value_set_0 - stores 0x30 into the memory address are ptr_to_currentInputValue and exits the routine

---

update_input_state

update_input_state does not take in any arguments and does not return any values. This routine is pretty straight forward(thank god, I'm so tired of typing). The value in ptr_to_keySpaceFlag is stored into the address at ptr_to_keySpaceState and 0x30 is stored into memory address at ptr_to_keySpaceFlag. The same functionality is repeated for ptr_to_keyWFlag/ptr_to_keyWState, ptr_to_keyW=AFlag/ptr_to_keyAState, ptr_to_keySFlag/ptr_to_keySState, and ptr_to_keyDFlag/ptr_to_keyDState.

---

get_input_flag

Get_input_flag does not take in any arguments and does not return any values. A branch and link is made to read_character which is returned into r0. r0 is compared to a whole lot of numbers. It is first compared to 0x20, 0x57/0x77, 0x41/0x61, 0x53/0x73, and

0x44/0x64. When r0 is equal to 0x20, a branch is made to space_flag_on which stores 0x31 into the address at ptr_to_keySpaceFlag. If r0 is equal to 0x57 or 0x77 a branch is made to w_flag_on which stores 0x31 into the address at ptr_to_keyWFlag. If r0 is equal to 0x41 or 0x61 a branch is made to a_flag_on which stores 0x31 into the address at ptr_to_keyAFlag. If r0 is equal to 0x0x53 or 0x73 a branch is made to s_flag_on which stores 0x31 into the address at ptr_to_keySFlag. If r0 is equal to 0x44 or 0x64 a branch is made to d_flag_on which stores 0x31 into the address at ptr_to_keyDFlag. Each subroutine end by exiting the routine.

---

take_face

Take_face takes in r0 as an argument and returns r1. R0 in this routine is used as the offset when loading in ptr_to_front_up_down_left_right_back_id. The value at that offset is compared to 0x30, 0x31, 0x32, 0x33, 0x34, and 0x35. When equal to 0x30 ptr_to_face0 is loaded into r1. ptr_to_face1 is loaded into r1 when equal to 0x31. ptr_to_face2 when equal to 0x32. ptr_to_face3 when equal to 0x33. ptr_to_face4 when equal to 0x34, and ptr_to_face5 is loaded into r1 when the value is equal to 0x35.

---

check_all_face_completed

Check_all_face_completed takes in no arguments and returns r0. This routine loops 5 times. Before the loop, both r4 and r0 are set to 0. The loop starts by branching and linking to take_face and check_face_completed and after and then increments r4 by r0 and increments r0 by 1. Once the loop is finished, r4 is compared to 0,1,2,3,4,5, and 6. When r4 is equal to 0, 0 is stored into r0. When r4 is equal to 1, 1 is stored into r0. When r4 is equal to 2, 3 is stored into r0. When r4 is equal to 3, 7 is stored into r0. When r4 is equal to 4, 0xF is stored into r0. When r4 is equal to 5, 5 is stored into r0. When r4 is equal to 6, 6 is stored into r0. After each subroutine there is then a branch and link to illuminate_LEDS which will take in r0 as an argument and illuminate the LED accordingly.

---

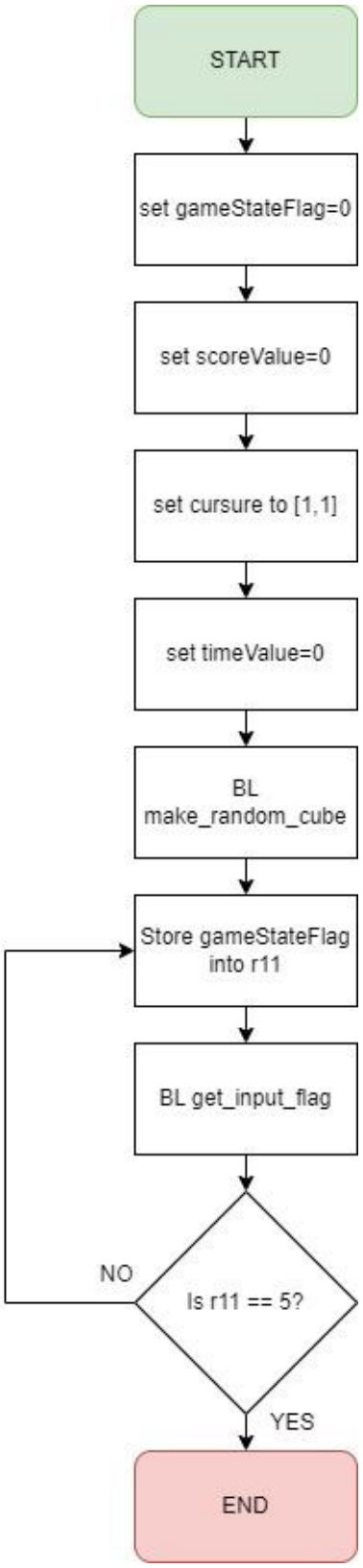
check_face_completed

Check_face_completed takes in r1 as an argument and returns r0. Check_face_completed starts by setting r4 to 1 and r5 to 8. Then the

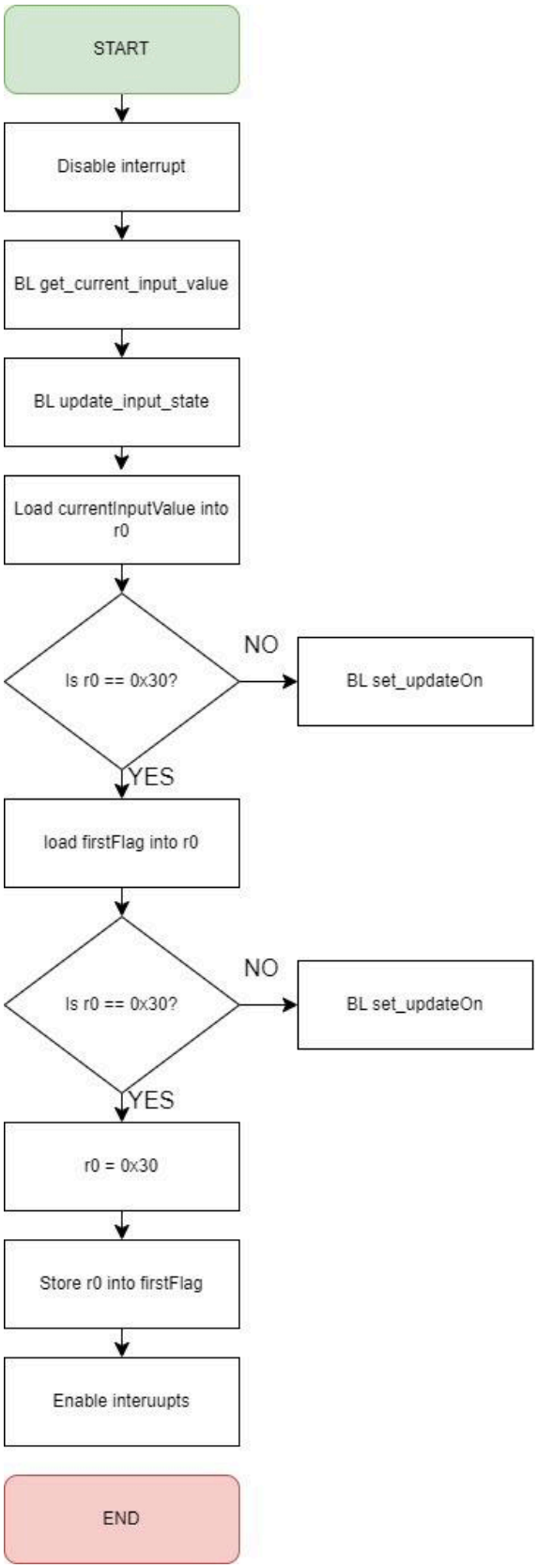
value at the memory address stored in r1 is loaded into r6. The routine then enters a loop that begins by loading r1 with an offset of r4 into r7 and compares that value to the value in r6. If they are not equal, r0 is set to 1 and the routine is exited. If they are equal, r4 is compared to r5 and r4 is incremented by 1. If r4 and r5 are not equal, it returns to the start of the loop.

Subroutine Flowcharts

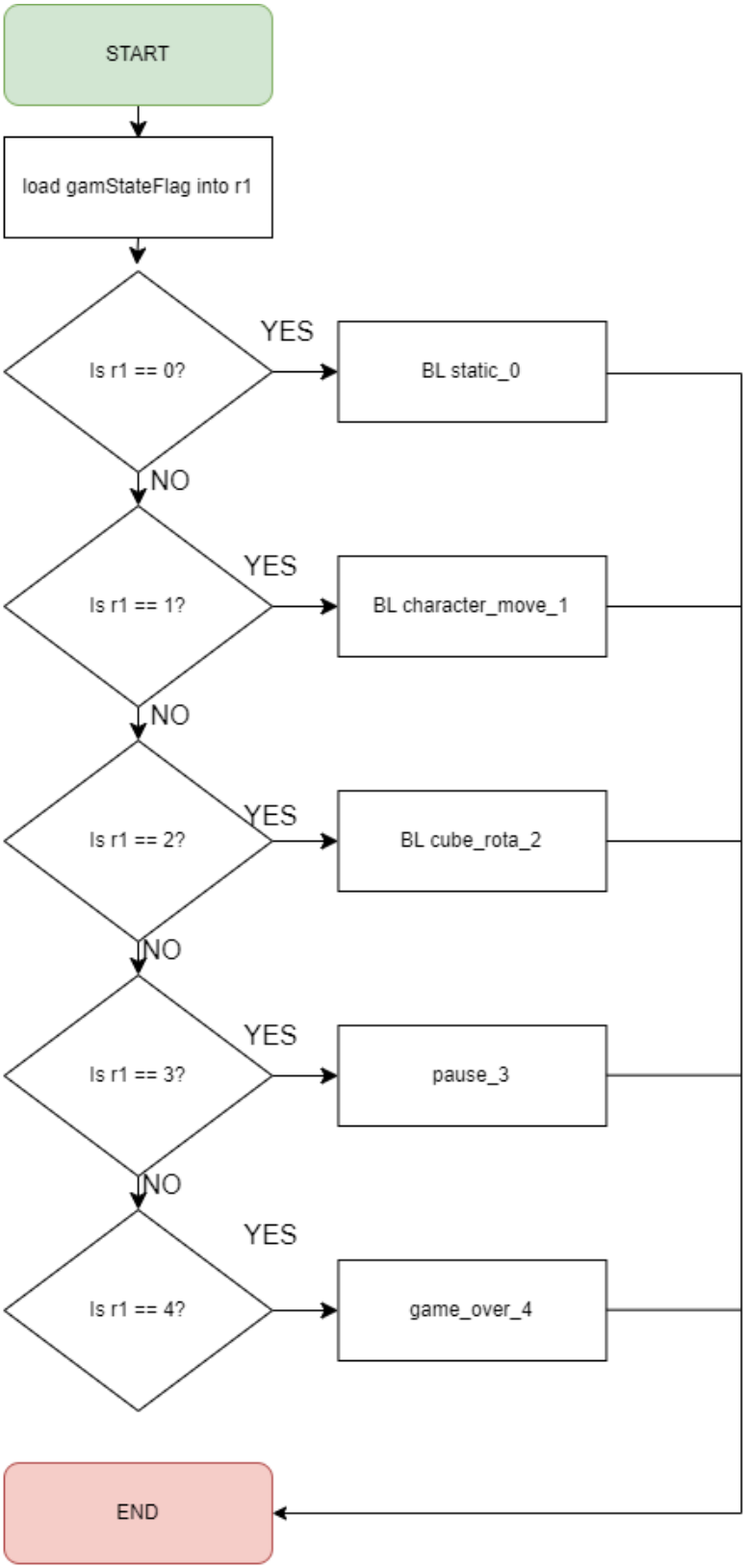
lab7



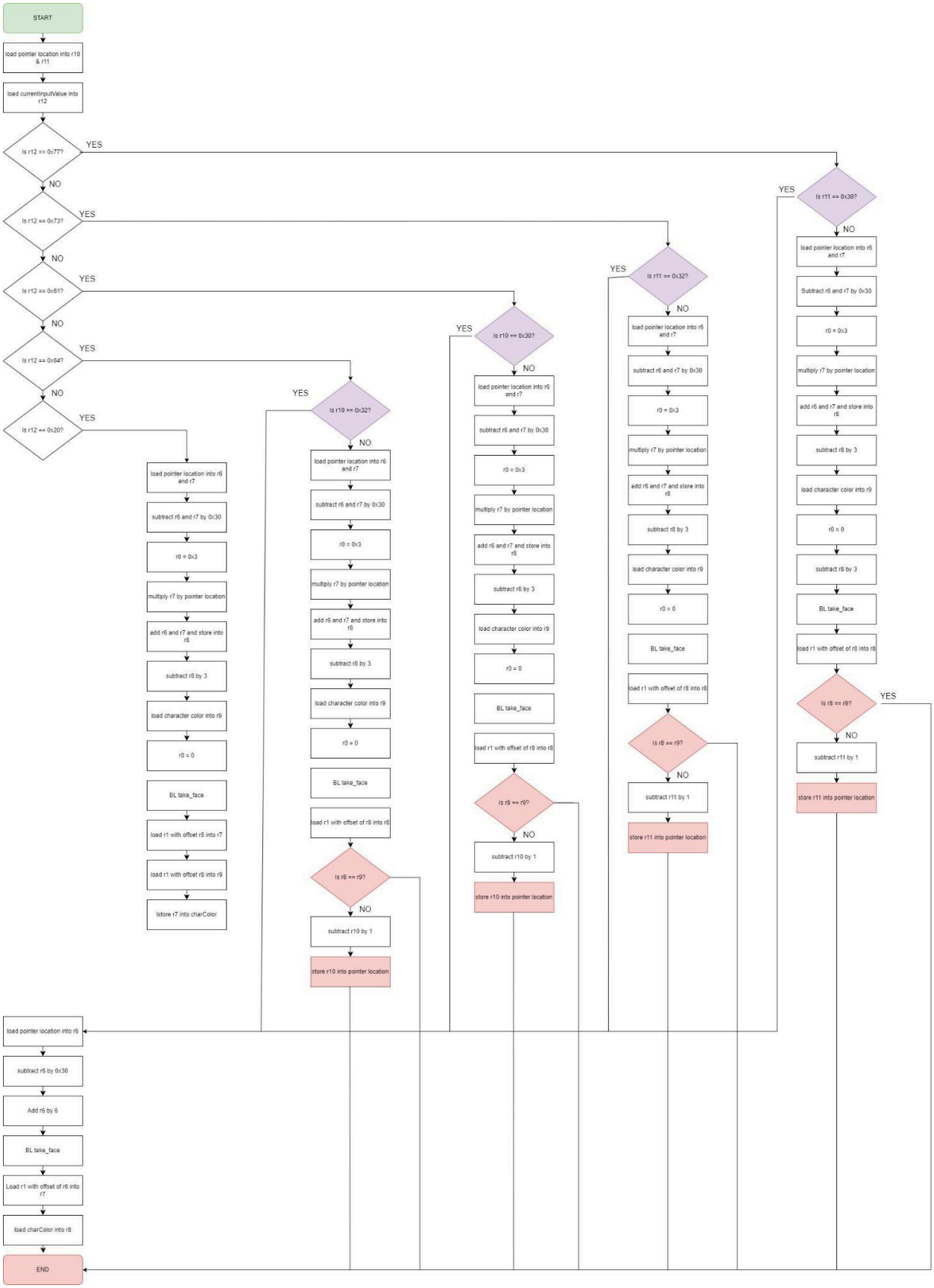
timer_handler



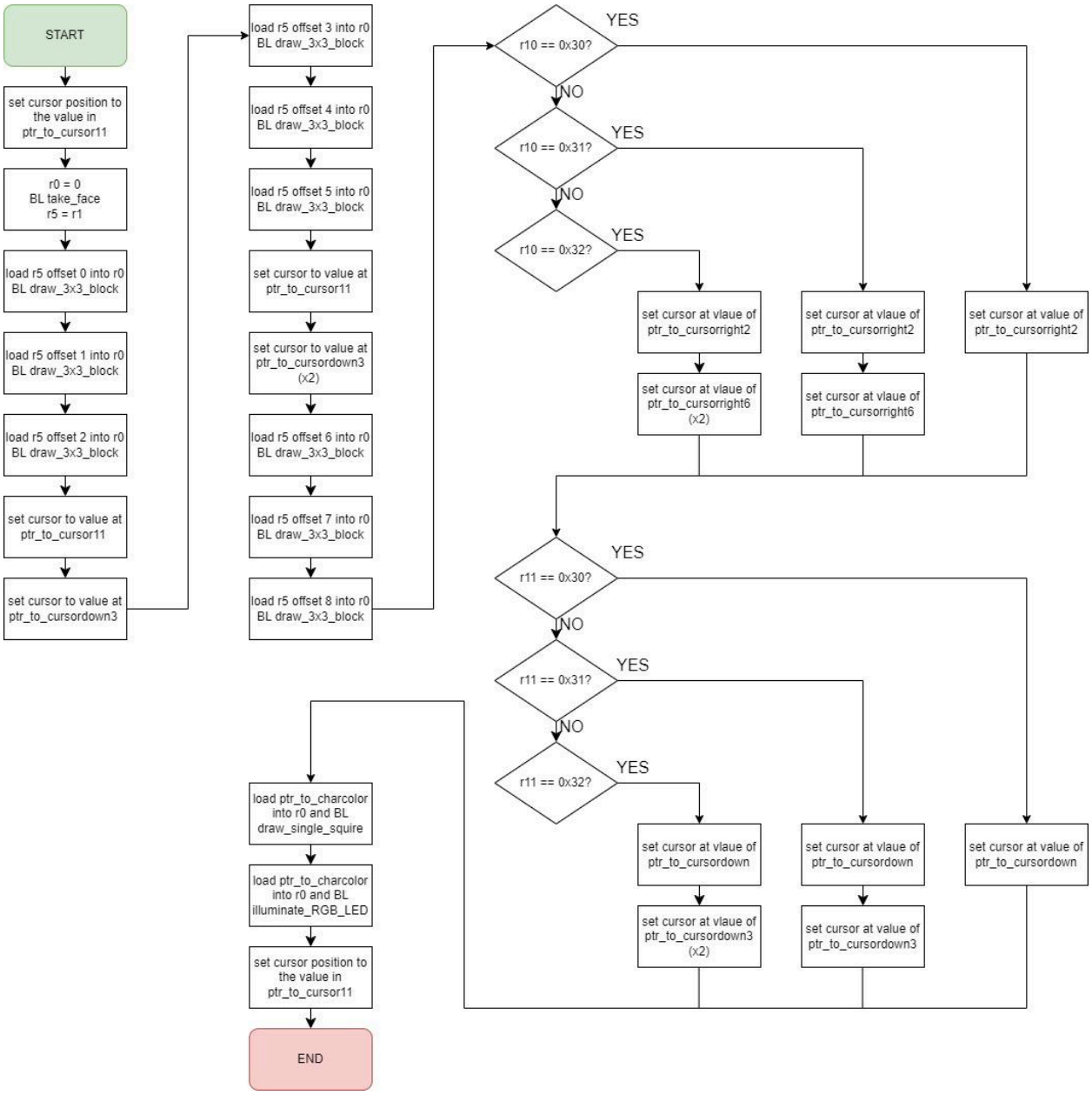
`set_updateOn`



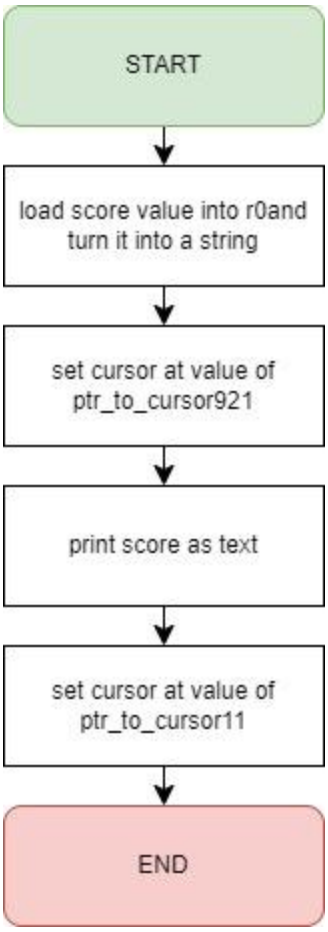
static_0


START
load pointer location into r10 & r11
load currentInputValue into r12
Is r12 == 0x77?
YES
NO
Is r12 == 0x73?
YES
NO
Is r12 == 0x61?
YES
NO
Is r12 == 0x64?
YES
NO
Is r12 == 0x20?
YES
load pointer location into r6 and r7
subtract r6 and r7 by 0x30
r0 = 0x3
multiply r7 by pointer location
add r6 and r7 and store into r8
subtract r8 by 3
load character color into r9
r0 = 0
BL take_face
load r1 with offset r8 into r7
load r1 with offset r8 into r9
Istore r7 into charColor
Is r10 == 0x32?
YES
NO
load pointer location into r6 and r7
subtract r6 and r7 by 0x30
r0 = 0x3
multiply r7 by pointer location
add r6 and r7 and store into r8
subtract r8 by 3
load character color into r9
r0 = 0
BL take_face
load r1 with offset r8 into r8
Is r8 == r9?
NO
subtract r10 by 1
store r10 into pointer location
Is r10 == 0x30?
YES
NO
load pointer location into r6 and r7
subtract r6 and r7 by 0x30
r0 = 0x3
multiply r7 by pointer location
add r6 and r7 and store into r8
subtract r8 by 3
load character color into r9
r0 = 0
BL take_face
load r1 with offset r8 into r8
Is r8 == r9?
NO
subtract r10 by 1
store r10 into pointer location
Is r11 == 0x32?
YES
NO
load pointer location into r6 and r7
subtract r6 and r7 by 0x30
r0 = 0x3
multiply r7 by pointer location
add r6 and r7 and store into r8
subtract r8 by 3
load character color into r9
r0 = 0
BL take_face
load r1 with offset r8 into r8
Is r8 == r9?
NO
subtract r11 by 1
store r11 into pointer location
Is r11 == 0x30?
YES
NO
load pointer location into r6 and r7
Subtract r6 and r7 by 0x30
r0 = 0x3
multiply r7 by pointer location
add r6 and r7 and store into r8
subtract r8 by 3
load character color into r9
r0 = 0
subtract r8 by 3
BL take_face
load r1 with offset of r8 into r8
Is r8 == r9?
YES
NO
subtract r11 by 1
store r11 into pointer location
load pointer location into r6
subtract r6 by 0x30
Add r6 by 6
BL take_face
Load r1 with offset of r6 into r7
load charColor into r8
END

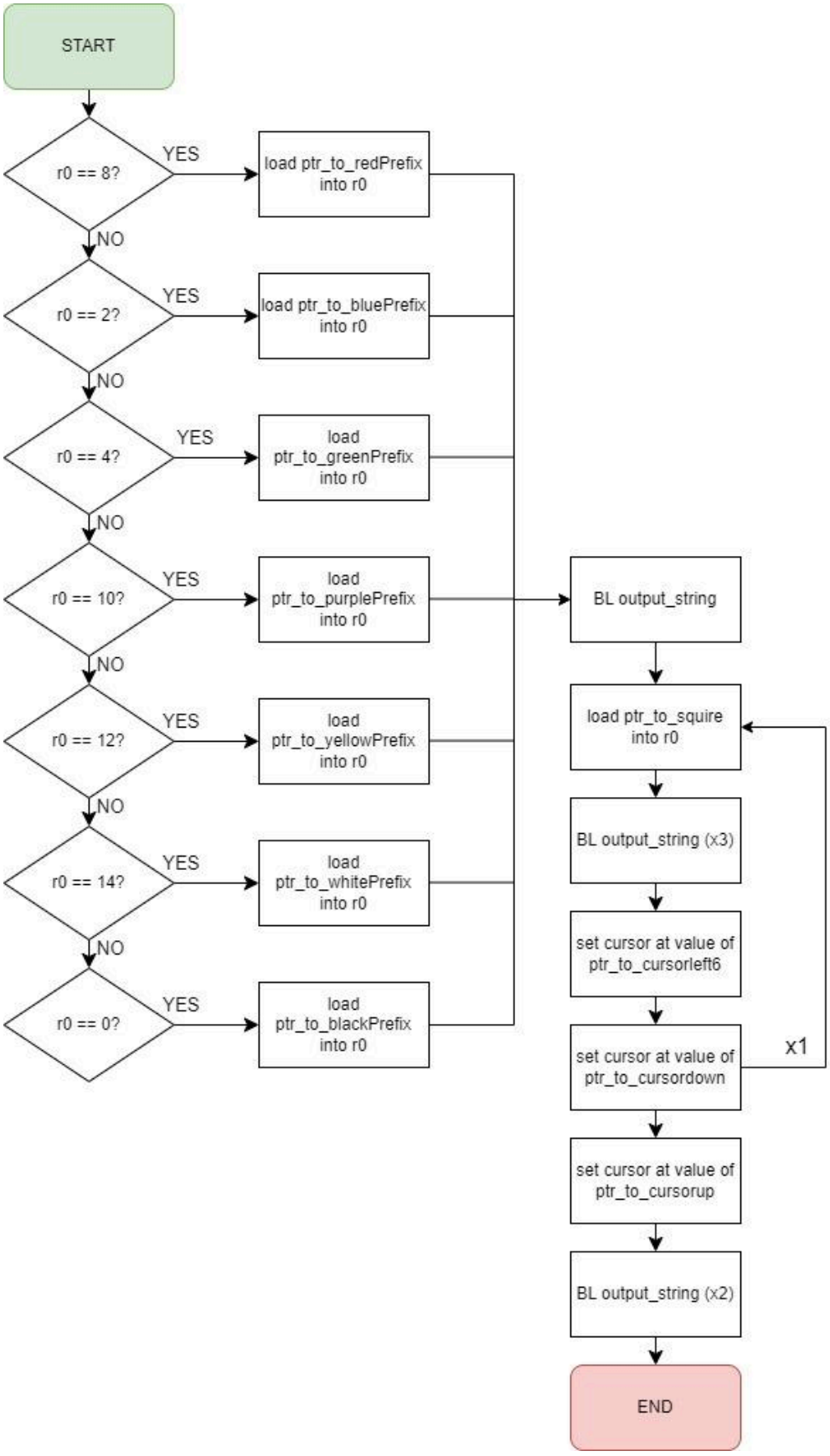
static_face_draw



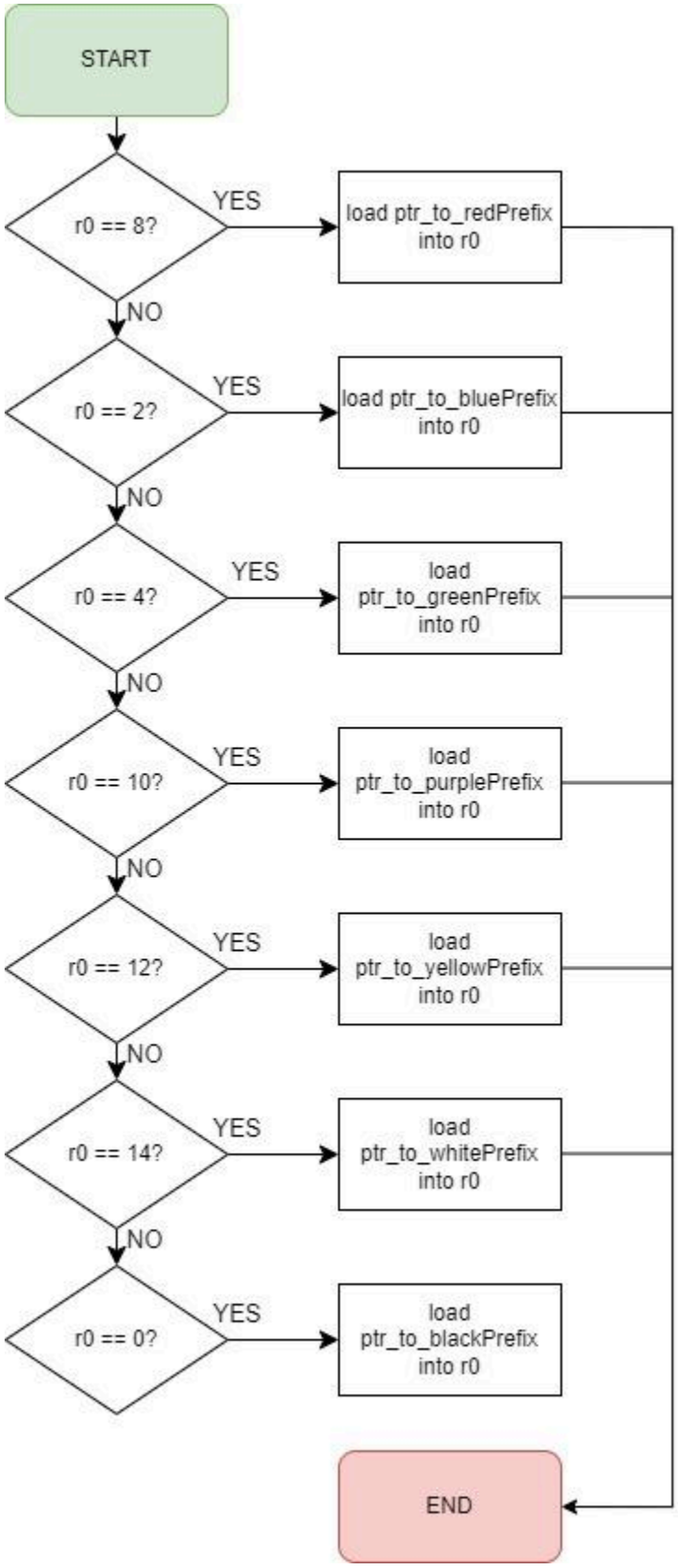
step_counter_draw



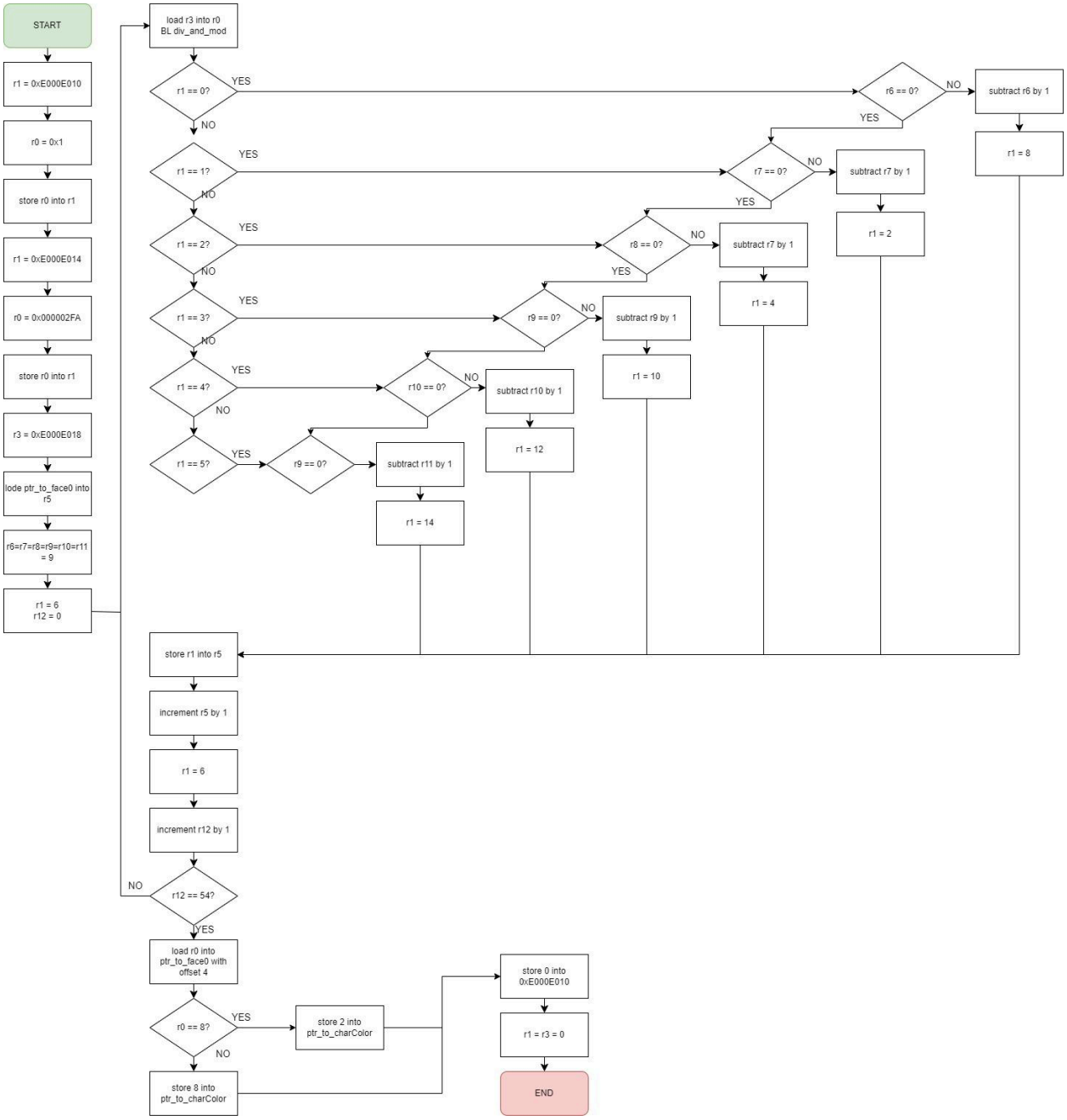
draw_3x3_block



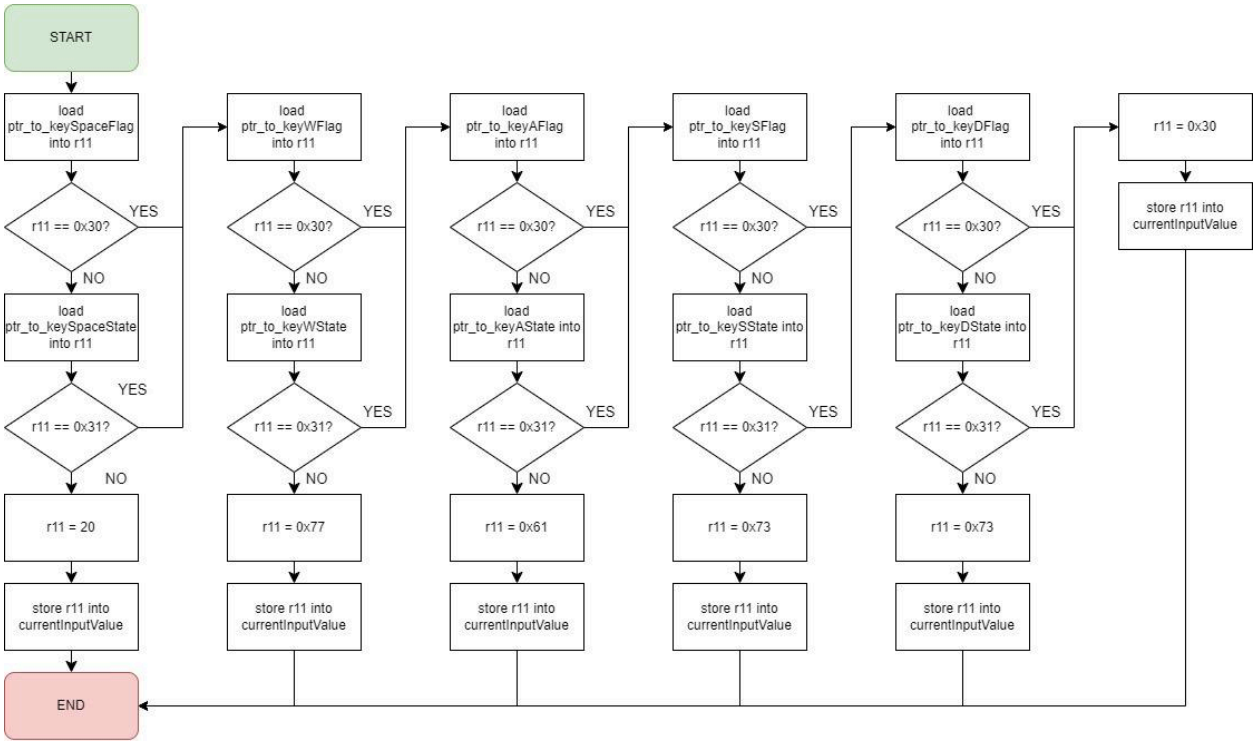
draw_single_squire



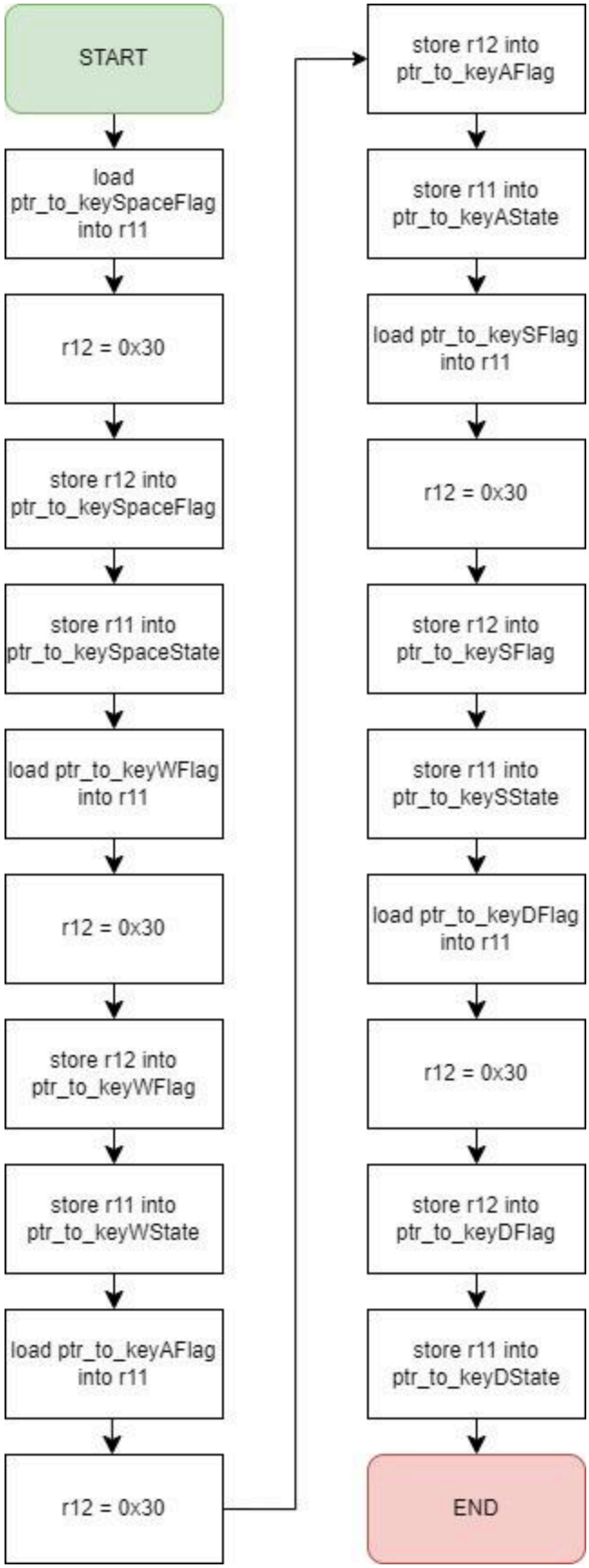
make_random_cube



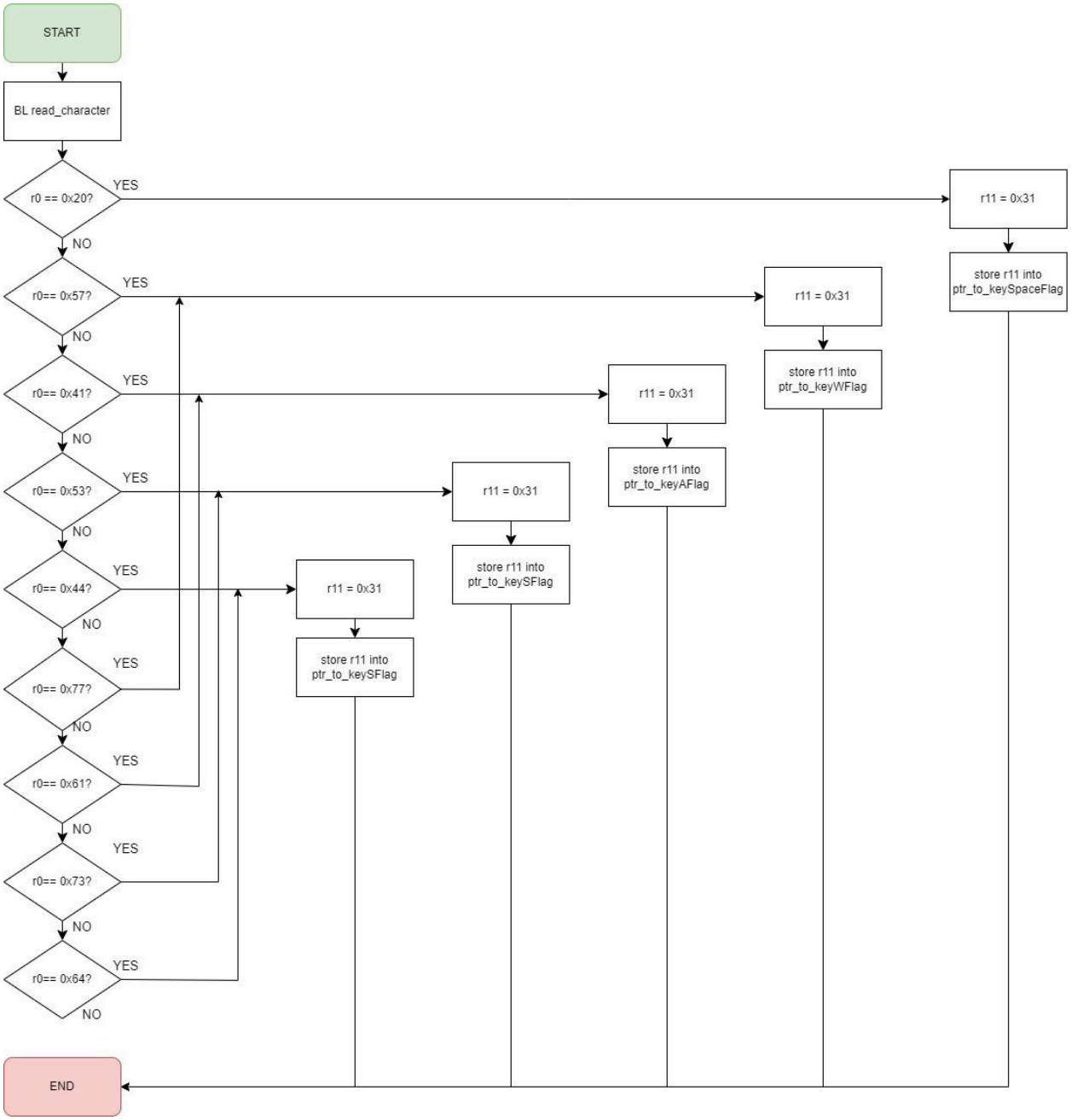
get_current_input_value



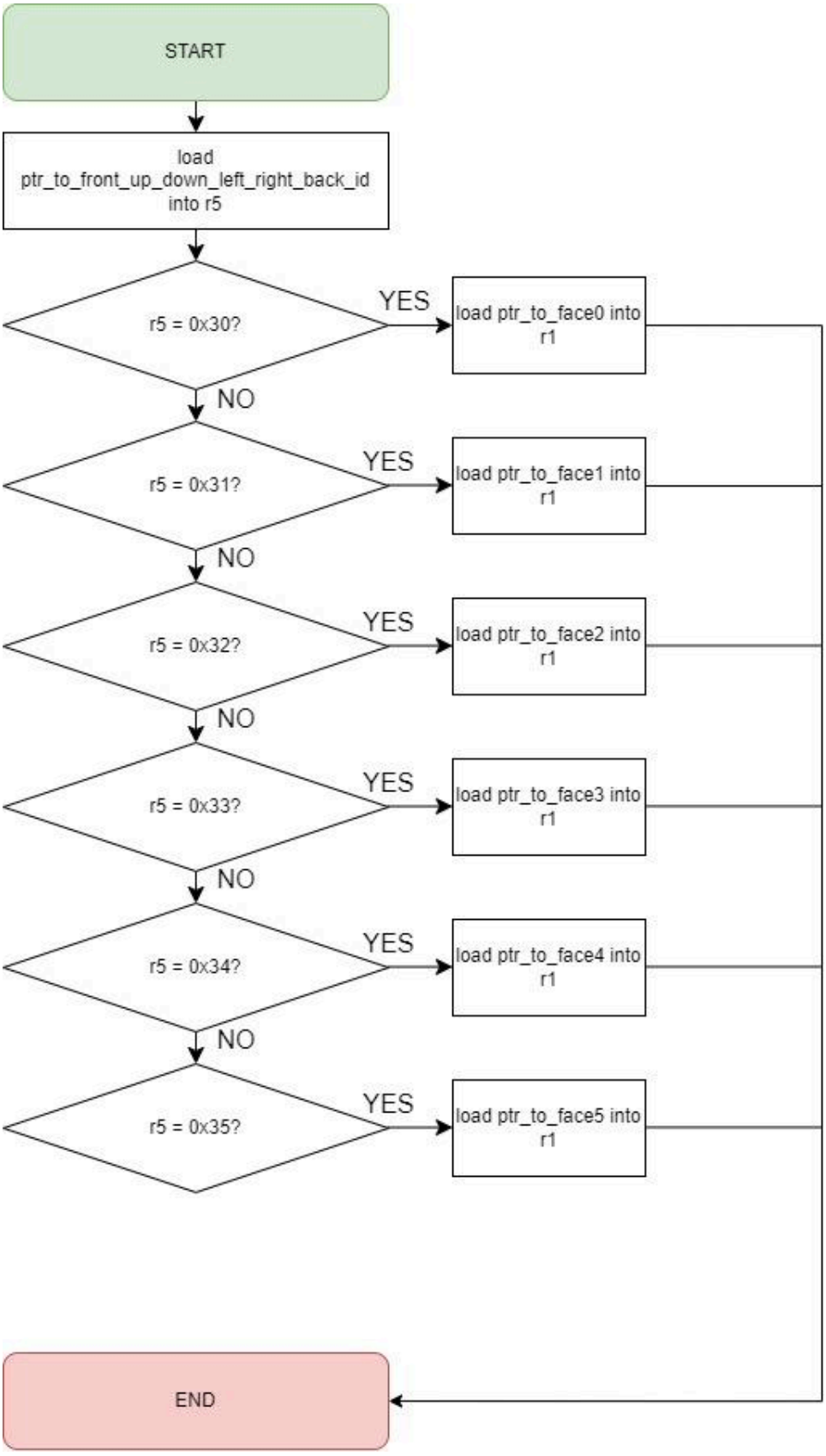
`update_input_state`



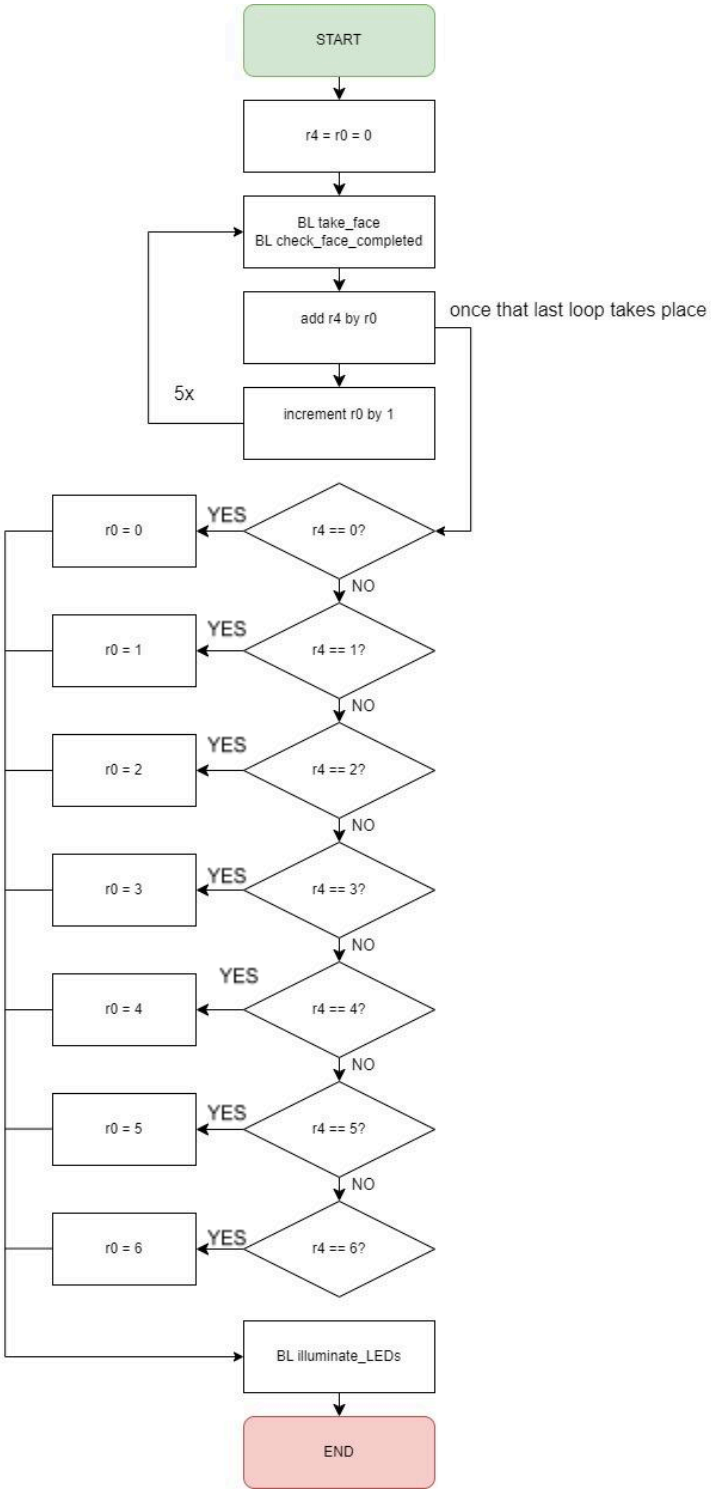
get_input_flag



take_face


START
load ptr_to_front_up_down_left_right_back_id into r5
r5 = 0x30?
YES
load ptr_to_face0 into r1
NO
r5 = 0x31?
YES
load ptr_to_face1 into r1
NO
r5 = 0x32?
YES
load ptr_to_face2 into r1
NO
r5 = 0x33?
YES
load ptr_to_face3 into r1
NO
r5 = 0x34?
YES
load ptr_to_face4 into r1
NO
r5 = 0x35?
YES
load ptr_to_face5 into r1
END

`check_all_face_completed`



`check_face_completed`