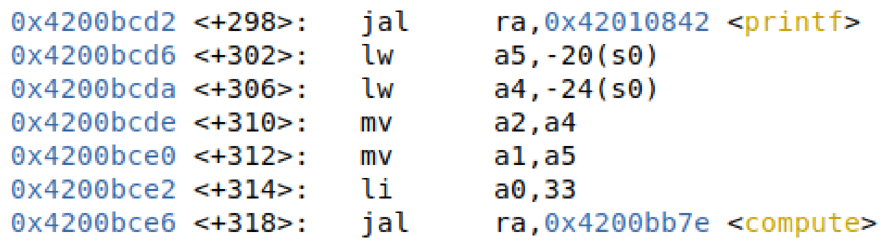
Rohan Kapur
ECE 121 Spring 2023
Due: 4/19/23
Lab 2 Report

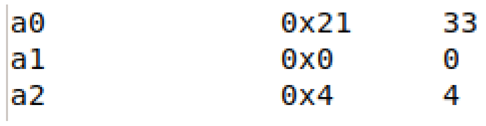
Lab 2.1

The arguments passed to "compute" are **33, 0, and 4** stored in argument/return value registers a0, a1, and a2. I found these out by using gdb, where I first ran "thb app_main" to set a temporary hardware breakpoint at the app_main function in gdb, let program run ("continue") until it hit that breakpoint, then disassembled the code at that point by running "disas" which allowed me to find the instruction address (app_main+318) where the compute function is invoked after the arguments are stored in registers a0, a1, and a2.



```
0x4200bcd2 <+298>:   jal     ra,0x42010842 <printf>
0x4200bcd6 <+302>:   lw      a5,-20(s0)
0x4200bcda <+306>:   lw      a4,-24(s0)
0x4200bcde <+310>:   mv      a2,a4
0x4200bce0 <+312>:   mv      a1,a5
0x4200bce2 <+314>:   li      a0,33
0x4200bce6 <+318>:   jal     ra,0x4200bb7e <compute>
```

I then set another hardware breakpoint for that address ("thb 0x4200bce6"), let the program continue running again until it got to that breakpoint, then finally inspected the register values at that point by running "info registers", which yielded the following that showed the values stored in the argument registers a0, a1, and a2 when "compute" was invoked



```
a0               0x21     33
a1               0x0      0
a2               0x4      4
```

Lab 2.2

References used:

- I2C esp32c3 documentation: https://docs.espressif.com/projects/esp-idf/en/latest/esp32c3/api-reference/peripherals/i2c.html
- Rust ESP32-C3 board GitHub page: https://github.com/esp-rs/esp-rust-board
- SHTC3 sensor datasheet: https://www.mouser.com/datasheet/2/682/Sensirion_04202018_HT_DS_SHTC3_Preliminiary_D2-1323493.pdf

I initially started with code generated by ChatGPT with the query: "Hi. I have a rust esp32c3 development board with an shtc3 temp and humidity sensor. Can you give me working code to continuously output the temperature (in celsius and fahrenheit) and humidity (in %) every 2 seconds?" I ended up scrapping it, however, when it wasn;t working, and mainly used the references above when creating the working code. The chatGPT code mainly acted as a boilerplate then.

Initial Code from chatGPT (which I wholly revamped):

```
#include <stdio.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "driver/i2c.h"

#define SHTC3_ADDR 0x70
#define SHTC3_CMD_MEASURE_H 0x5C
#define SHTC3_CMD_MEASURE_T 0x7C

#define I2C_MASTER_SCL_IO 22
#define I2C_MASTER_SDA_IO 21
#define I2C_MASTER_NUM I2C_NUM_0
#define I2C_MASTER_FREQ_HZ 100000

void i2c_master_init()
{
   i2c_config_t conf = {
       .mode = I2C_MODE_MASTER,
       .sda_io_num = I2C_MASTER_SDA_IO,
       .sda_pullup_en = GPIO_PULLUP_ENABLE,
       .scl_io_num = I2C_MASTER_SCL_IO,
       .scl_pullup_en = GPIO_PULLUP_ENABLE,
       .master.clk_speed = I2C_MASTER_FREQ_HZ,
   };
   i2c_param_config(I2C_MASTER_NUM, &conf);
   i2c_driver_install(I2C_MASTER_NUM, conf.mode, 0, 0, 0);
}

float read_temperature()
{
   uint8_t cmd[2] = {SHTC3_CMD_MEASURE_T, 0x00};
   i2c_cmd_handle_t cmd_handle = i2c_cmd_link_create();
   i2c_master_start(cmd_handle);
   i2c_master_write_byte(cmd_handle, SHTC3_ADDR << 1 | I2C_MASTER_WRITE, true);
   i2c_master_write(cmd_handle, cmd, sizeof(cmd), true);
   i2c_master_stop(cmd_handle);
   i2c_master_cmd_begin(I2C_MASTER_NUM, cmd_handle, 1000 / portTICK_RATE_MS);
   i2c_cmd_link_delete(cmd_handle);
```

```
    vTaskDelay(20 / portTICK_RATE_MS); // Wait for conversion to finish

    cmd[0] = 0xE0;
    cmd_handle = i2c_cmd_link_create();
    i2c_master_start(cmd_handle);
    i2c_master_write_byte(cmd_handle, SHTC3_ADDR << 1 | I2C_MASTER_WRITE, true);
    i2c_master_write(cmd_handle, cmd, sizeof(cmd), true);
    i2c_master_start(cmd_handle);
    i2c_master_write_byte(cmd_handle, SHTC3_ADDR << 1 | I2C_MASTER_READ, true);
    uint8_t buf[3];
    i2c_master_read(cmd_handle, buf, sizeof(buf), I2C_MASTER_LAST_NACK);
    i2c_master_stop(cmd_handle);
    i2c_master_cmd_begin(I2C_MASTER_NUM, cmd_handle, 1000 / portTICK_RATE_MS);
    i2c_cmd_link_delete(cmd_handle);

    float temperature = (buf[0] * 256.0f + buf[1]) * 175.0f / 65535.0f - 45.0f;
    return temperature;
}

float read_humidity()
{
    uint8_t cmd[2] = {SHTC3_CMD_MEASURE_H, 0x00};
    i2c_cmd_handle_t cmd_handle = i2c_cmd_link_create();
    i2c_master_start(cmd_handle);
    i2c_master_write_byte(cmd_handle, SHTC3_ADDR << 1 | I2C_MASTER_WRITE, true);
    i2c_master_write(cmd_handle, cmd, sizeof(cmd), true);
i2c_master_stop(cmd_handle);
i2c_master_cmd_begin(I2C_MASTER_NUM, cmd_handle, 1000 / portTICK_RATE_MS);
i2c_cmd_link_delete(cmd_handle);

vTaskDelay(20 / portTICK_RATE_MS); // Wait for conversion to finish

cmd[0] = 0xE5;
cmd_handle = i2c_cmd_link_create();
i2c_master_start(cmd_handle);
i2c_master_write_byte(cmd_handle, SHTC3_ADDR << 1 | I2C_MASTER_WRITE, true);
i2c_master_write(cmd_handle, cmd, sizeof(cmd), true);
i2c_master_start(cmd_handle);
i2c_master_write_byte(cmd_handle, SHTC3_ADDR << 1 | I2C_MASTER_READ, true);
```

```
uint8_t buf[3];
i2c_master_read(cmd_handle, buf, sizeof(buf), I2C_MASTER_LAST_NACK);
i2c_master_stop(cmd_handle);
i2c_master_cmd_begin(I2C_MASTER_NUM, cmd_handle, 1000 / portTICK_RATE_MS);
i2c_cmd_link_delete(cmd_handle);

float humidity = (buf[0] * 256.0f + buf[1]) * 100.0f / 65535.0f;
return humidity;
}

void task_sensor(void *pvParameter)
{
while (1) {
float temperature = read_temperature();
float humidity = read_humidity();
    printf("Temperature: %.1f C (%.1f F)\n", temperature, temperature * 1.8f + 32.0f);
    printf("Humidity: %.1f %%\n", humidity);

    vTaskDelay(2000 / portTICK_RATE_MS);
}
}

void app_main()
{
i2c_master_init();
xTaskCreate(task_sensor, "sensor", 4096, NULL, 5, NULL);
}
```