Do more and more
Sharif University of Technology

May 27, 2024

Keywords: Assembly, C, Proteus

1 Introduction

In this Question We will follow the HW03 file to design a system using 3 Arduino; one as the main board and two as the sensor device handler.

1.0.1 Question 1

The use of I2C in a multi-master, multi-slave configuration can pose challenges related to bus arbitration and stability. While I2C supports multi-master setups, issues like arbitration conflicts can arise when multiple masters attempt to access the bus simultaneously. In I2C multi-master mode, conflicts are resolved through a process called arbitration. When multiple masters attempt to access the bus simultaneously, only one master should complete the transmission to prevent data corruption. The I2C protocol includes mechanisms to handle arbitration conflicts and ensure data integrity. Here is the strategy used to resolve conflicts in I2C multi-

- master mode:
  Automatic Multi-Master Transaction Handling: The I2C component automatically checks for arbitration conflicts. If one master loses arbitration, the transaction is considered complete, and appropriate completion status flags are set.

- Manual Multi-Master Transaction Handling:
  After each byte transfer, the master must check for the return condition. This manual approach involves monitoring the bus to ensure that only one master is actively transmitting at a time. If a master loses arbitration, it needs to switch to a slave mode to avoid data corruption.

- Bus Contention Management:
  Bus Contention Management: In multi-master setups, the bus should be checked

to see if it is busy before initiating a transaction. If another master is already communicating with a slave, the program must wait until the current operation is complete before starting a new transaction to avoid conflicts.

By implementing these strategies, the I2C protocol effectively manages conflicts in multi-master mode, ensuring that only one master accesses the bus at a time to maintain communication stability and data integrity.

1.0.2 Question 2

To connect two devices using I2C, follow these steps:

1. Identify Device Addresses:
   Ensure that each device has a unique I2C address. This is crucial for distinguishing between devices on the bus.

2. Connect Power and Ground:
   Connect the VCC and GND pins of both devices together to provide power and establish a common ground.

3. Connect SDA and SCL Lines:
   Connect the SDA (data) and SCL (clock) lines of the devices. These lines are used for communication between the devices.

4. Add Pull-Up Resistors:
   Include pull-up resistors between the VCC line and the SDA line, as well as between the VCC line and the SCL line. Pull-up resistors help maintain signal integrity on the bus.

5. Designate Master and Slave:
   Determine which device will act as the master and which will be the slave. The master generates the clock signal and initiates communication, while the slave responds to commands from the master.

6. Check Wiring and Addressing:
   Verify the wiring connections and device addresses to ensure proper communication. Use tools like an I2C scanner to confirm that the devices are detected on the bus.

By following these steps, you can successfully connect two devices using the I2C bus, enabling communication between them while ensuring data integrity and reliable operation. Now it's time to understand the software steps to run the I2C protocol. To communicate using the I2C protocol, the following steps are involved:

1. Start Condition:
   The communication begins with a start condition where the SDA line switches from a high voltage level to a low voltage level before the SCL line switches from high to low.

2. Address Frame:
   After the start condition, the master sends the address of the slave it wants to communicate with. Each slave compares this address with its own and sends an ACK bit if the address matches.

3. Read or Write Bit:
   Following the address frame, a single bit specifies whether the master is sending data to the slave (low voltage level) or requesting data from it (high voltage level).

4. Data Frame:
   Each data frame is 8 bits long and sent with the most significant bit first. The data frame contains the actual data being transmitted.

5. ACK/NACK Bit:
   After each frame, an acknowledge (ACK) or no-acknowledge (NACK) bit is sent. If the frame was successfully received, an ACK bit is returned to the sender.

6. Stop Condition:
   The communication ends with a stop condition where the SDA line switches from a low voltage level to a high voltage level after the SCL line switches from low to high.

By following these steps, devices can effectively communicate using the I2C protocol, ensuring data integrity and reliable transmission between the master and slave devices.

1.0.3 Question 3

Baud rate is a crucial parameter in UART (Universal Asynchronous Receiver-Transmitter) communication. It determines the speed at which data is transmitted between devices using the UART protocol.
Some common baud rates used in UART communication include:

- 9600 bps (bits per second)
- 19200 bps
- 38400 bps

- 57600 bps
- 115200 bps

UART is an asynchronous protocol, meaning it does not use a clock signal for synchronization. The baud rate is used to synchronize the transmitter and receiver, ensuring that data is sampled correctly. Mismatched baud rates between devices can lead to timing discrepancies, causing data corruption and transmission errors. Ensuring the same baud rate on both ends is crucial for reliable communication.

1.0.4 Question 4

Here are the pros and cons of UART and I2C communication protocols in LaTeX format:

UART Pros and Cons

Pros:

- **Simplicity and versatility**: UART is simple and versatile, making it easy to implement in various systems.
- **Widespread adoption**: UART has been a standard for decades, leading to widespread support and compatibility.
- **Asynchronous communication**: Allows devices with different clock speeds to communicate effectively.
- **No master-slave configuration**: Devices can communicate in a peer-to-peer fashion.
- **Efficient for short distances**: Suitable for on-board communication in electronic systems.
- **Low overhead**: Relatively low overhead compared to more complex protocols.
- **Real-time communication**: Supports real-time communication for timely data transmission.
- **Flexible baud rate selection**: Users can select baud rates according to application requirements.
- **Simple error detection**: Can incorporate basic error detection using parity bits.

Cons:

- **Limited cable length**: Generally limited in cable length, especially in noisy environments.

- **Lack of inherent flow control**: Lacks inherent flow control mechanisms, leading to potential data overrun issues.
- **Single-point communication**: Typically supports communication between two devices at a time.
- **Limited data frame structure**: Simple data frame structure may be a limitation in some applications.
- **Synchronization challenges**: Achieving synchronization between transmitter and receiver can be challenging.
- **Not suitable for high-speed communication**: May not be ideal for high-speed communication compared to other protocols.
- **No built-in addressing**: Lacks built-in addressing mechanisms for multiple devices on a bus.
- **Power consumption**: May consume more power compared to some low-power communication protocols.

I2C Pros and Cons
Pros:

- **Flexibility**: Supports multi-master and multi-slave communication, adding functionality.
- **Addressing feature**: Inherent ability to use chip addressing, simplifying device addition.
- **Simplicity**: Requires only two bidirectional signal lines for communication.
- **Better error handling**: Relies on ACK/NACK for robust error detection and correction.
- **Adaptable**: Can work well with both slow and fast ICs.

Pros:

- **Conflicts**: Possibility of address conflicts due to chip addressing.
- **Slower speeds**: Open-drain design limits speed compared to push-pull protocols.
- **Requires more space**: Requires space for pull-up resistors on the PCB.

1.0.5 Question 5

Software Serial and Hardware Serial in Arduino serve the purpose of serial communication, but they differ in their implementation and capabilities. Here is a breakdown of the differences between Software Serial and Hardware Serial based on the provided sources:

Software Serial:

- **Definition:** Software Serial is a library in Arduino that emulates the behavior of hardware serial communication using software, allowing communication on digital pins other than the hardware serial pins.
- **Usage**: It is used when additional serial communication ports are needed beyond the hardware-supported ones or when the hardware serial pins are already in use.
- **Limitations:**
  - Only one software serial port can receive data at a time, limiting simultaneous communication.
  - Specific pins on certain Arduino models have limitations for RX support.
  - Speeds up to 115200 bps are achievable with Software Serial.
- **Example:** The SoftwareSerial library is included in Arduino IDE 1.0 and above, and its usage involves creating a SoftwareSerial object with specified RX and TX pins.

Hardware Serial:

- **Definition:** Hardware Serial refers to the built-in hardware support for serial communication on specific pins of the Arduino board, such as pins 0 and 1 on the Arduino Uno.
- **Usage:** It is the primary serial communication method on Arduino boards, providing reliable and efficient communication.
- **Capabilities:**
  - Hardware Serial is more optimized and reliable than Software Serial.
  - Multiple hardware serial ports are available on some Arduino boards like the Arduino Mega, accessed using Serial, Serial1, Serial2, etc.
- **Example:** Hardware Serial is used for communication with the computer via USB-to-UART converters on Arduino Uno, facilitating communication for programming and debugging.

Comparison:

- Efficiency: Hardware Serial is more efficient and reliable due to its hardware-based implementation, making it suitable for critical and high-speed communication. - **Flexibility**: Software Serial offers flexibility by allowing serial communication on any digital pins, but it comes with limitations in terms of speed and simultaneous communication.
- Usage: Hardware Serial is typically preferred for primary communication tasks, while Software Serial is used for additional communication needs or when hardware serial pins are occupied.

In summary, Hardware Serial provides optimized and reliable serial communication, while Software Serial offers flexibility for additional communication ports with some limitations in speed and simultaneous communication capabilities.

1.0.6 Question 6

LDR (Light Dependent Resistor) or Photoresistor

The **Light Dependent Resistor** (**LDR**) commonly known as **Photoresistor** is a special type of resistor whose resistance varies with the amount of light falling on it. It has the following key features:

- **No polarity**: LDRs have no specific polarity, so they can be connected in any direction.
- **Breadboard friendly**: LDRs can be easily used on breadboards or perf boards.
- **Symbol**: The symbol for an LDR is similar to a resistor but includes inward arrows indicating light signals.
- **Light sensing**: The primary function of an LDR is to sense light intensity.
- **Resistance variation**: In dark conditions, the resistance of an LDR is in the range of megaohms, while in bright light, the resistance decreases to a few ohms.
- **Applications**: LDRs are commonly used as light sensors to detect day/night cycles, in automatic lighting control systems, smoke detectors, burglar alarms, and batch counting systems.

LDR Operation

The resistance of an LDR decreases as the light intensity increases. This property allows it to be used as a light sensor. When the LDR is placed in a dark environment, its resistance is very high (in the megaohm range). As light falls on the LDR, its resistance gradually decreases, reaching a few ohms in bright light conditions.

textbfUsing an LDR

LDRs can be easily integrated into various circuits, including those with microcontrollers like Arduino or PIC, as well as analog ICs like op-amps. A simple circuit using an LDR involves connecting it in a potential divider configuration with a fixed resistor.[1][3]

LDR Characteristics

- **Response time**: LDRs have a fast response time, typically around 10 ms for the resistance to drop when exposed to light after darkness, and up to 1 second for the resistance to rise back to the dark value after the removal of light.[2]
- **Memory effect**: LDRs exhibit a memory effect, where their resistance is influenced by the lighting conditions they were previously exposed to. This effect can be minimized by storing LDRs in light before use.[3]

In summary, the LDR is a versatile and cost-effective light sensor that can be easily integrated into various electronic circuits and systems for light detection and monitoring applications.

1.0.7 Simulation

Here I had simulated the whole project in the Proteus and here is the schematic including the code. **LDR Sensor:**

```
 #include <Arduino.h>

float lux = 0.00 , ADC_value=0.0048828125
, LDR_value;
void setup() {
  pinMode(A0 , INPUT);
  Serial.begin(9600);
}
void loop() {
  LDR_value = analogRead(A0);
  lux = (250.000000/(ADC_value*LDR_value))
        -50.000000;
  String out = "l"+ String(lux) + "l" ;
  Serial.println(out);
  delay(25);
}
```

Tempreture and Humidity sensor:

```
#include "Wire.h"
```

```
#include "SHT2x.h"
SHT2x sht;
void setup()
{
  Serial.begin(9600);
}
void loop()
{
  bool success  = sht.read();
  if (success)
  {
    String temp =String((sht.getTemperature()));
    String hum =String((sht.getHumidity()));
    String csv = temp + "," + hum;
    Serial.println(csv);
  }

}
}
```

Main Board:

```
// include the library code:
#include <LiquidCrystal.h>
#include <Arduino.h>
#include <AltSoftSerial.h>

// these part is coupied from internet
const int rs = 12, en = 11,
d4 = 5, d5 = 4, d6 = 3, d7 = 2;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);

AltSoftSerial altSerial1;
void setup() {
  altSerial1.begin(9600);
  Serial.begin(9600);
  lcd.begin(20, 4);
}
```

```
void loop()
{
  float hum = 0 ;
  float temp = 0 ;
  float lux = 0 ;
  float value = 0;
  if (Serial.available() > 0){
     String receivedValue = Serial.
                readStringUntil('\n');
     Serial.println(receivedValue);
    if (receivedValue.substring(0 , 1) == "l"
    &&receivedValue
        .substring(receivedValue.length()- 2 , x
    receivedValue.length()-1 ) =="l")
    {

     String v = receivedValue
                .substring(1 , receivedValue.length()-1 );
     value = v.toFloat();
     lux = value;
    }
  }
  if (altSerial1.available() > 0)
  {
    String receivedValue = altSerial1.readStringUntil('\n');

    int commaIndex = receivedValue.indexOf(',');

    String number1 = receivedValue.substring(0, commaIndex);

    String number2 = receivedValue.substring(commaIndex + 1);

     temp = number1.toFloat();

     hum = number2.toFloat();
  }
  lcd.setCursor(0, 0);
  lcd.print("LUX: ");
  lcd.print(lux) ;
  lcd.setCursor(0, 1);
  lcd.print("temp: ");
  lcd.print(temp);

  lcd.setCursor(0, 2);
  lcd.print("hum: ");
  lcd.print(hum);
  if (hum>=80)
  {
  lcd.setCursor(0, 3);
  lcd.print("no watering");
  }else if(hum <= 50){
  lcd.setCursor(0, 3);
  lcd.print("action: 15 cc/min ");

  }else if (temp <= 25 && lux <= 600 ){
  lcd.setCursor(0, 3);
  lcd.print("action: 10 cc/min ");

  }else if (temp <= 25 && lux >= 600 ){
  lcd.setCursor(0, 3);
  lcd.print("action: 5 cc/min ");

  }else if (temp >= 25 && lux >= 600){
  lcd.setCursor(0, 3);
  lcd.print("action: 5 cc/min ");

  }else if (temp >= 25 && lux <= 600){
  lcd.setCursor(0, 3);
  lcd.print("action: 10 cc/min ");
  }
}
}
```

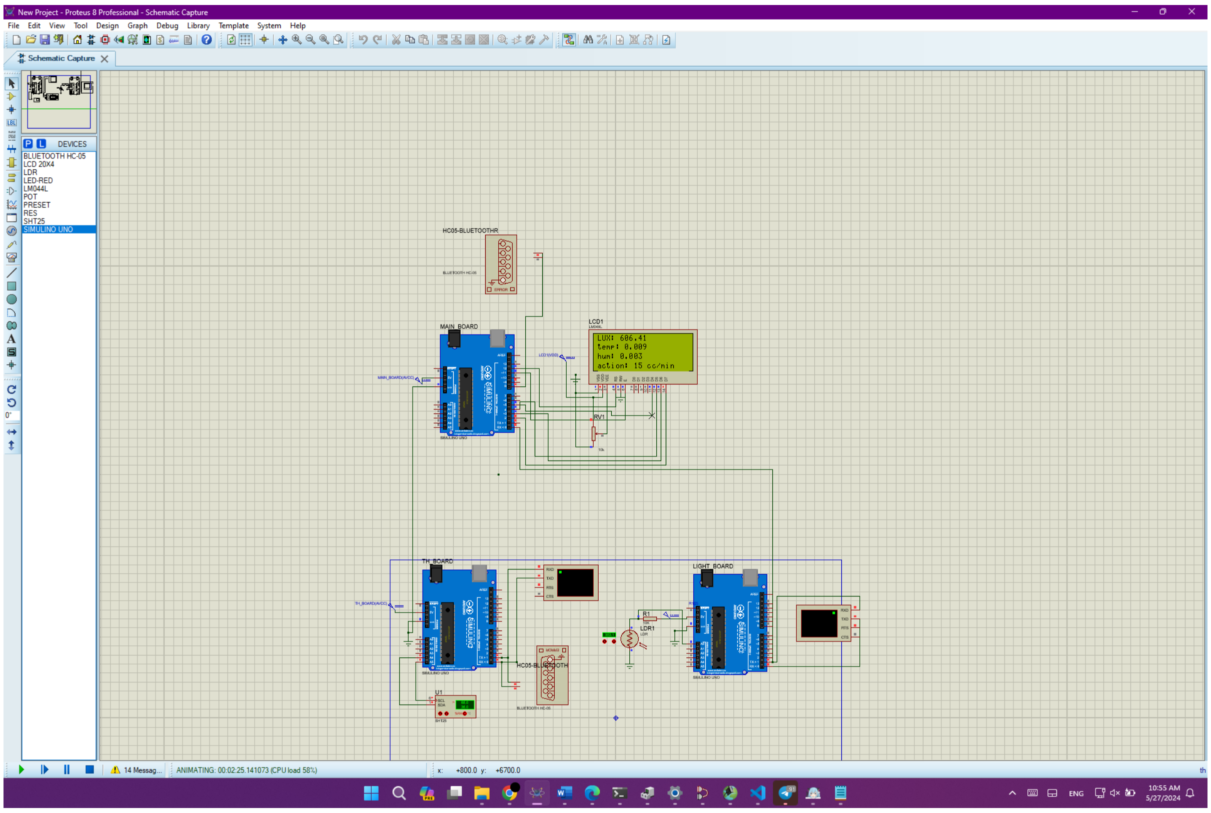
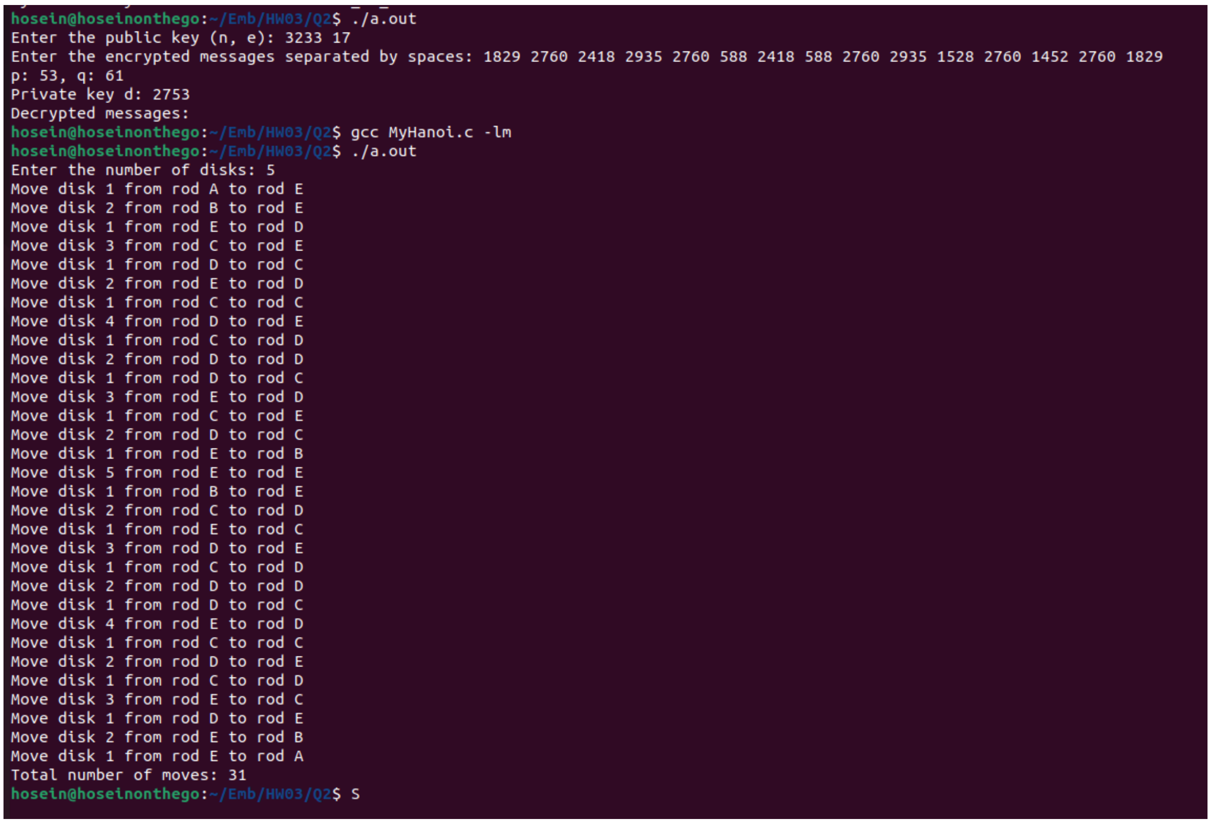


Figure 1: Main Board



```
hosein@hoseinonthego:~/Emb/HW03/Q2$ ./a.out
Enter the public key (n, e): 3233 17
Enter the encrypted messages separated by spaces: 1829 2760 2418 2935 2760 588 2418 588 2760 2935 1528 2760 1452 2760 1829
p: 53, q: 61
Private key d: 2753
Decrypted messages:
hosein@hoseinonthego:~/Emb/HW03/Q2$ gcc MyHanoi.c -lm
hosein@hoseinonthego:~/Emb/HW03/Q2$ ./a.out
Enter the number of disks: 5
Move disk 1 from rod A to rod E
Move disk 2 from rod B to rod E
Move disk 1 from rod E to rod D
Move disk 3 from rod C to rod E
Move disk 1 from rod D to rod C
Move disk 2 from rod E to rod D
Move disk 1 from rod C to rod C
Move disk 4 from rod D to rod E
Move disk 1 from rod C to rod D
Move disk 2 from rod D to rod D
Move disk 1 from rod D to rod C
Move disk 3 from rod E to rod D
Move disk 1 from rod C to rod E
Move disk 2 from rod D to rod C
Move disk 1 from rod E to rod B
Move disk 5 from rod E to rod E
Move disk 1 from rod B to rod E
Move disk 2 from rod C to rod D
Move disk 1 from rod E to rod C
Move disk 3 from rod D to rod E
Move disk 1 from rod C to rod D
Move disk 2 from rod D to rod D
Move disk 1 from rod D to rod C
Move disk 4 from rod E to rod D
Move disk 1 from rod C to rod C
Move disk 2 from rod D to rod E
Move disk 1 from rod C to rod D
Move disk 3 from rod E to rod C
Move disk 1 from rod D to rod E
Move disk 2 from rod E to rod B
Move disk 1 from rod E to rod A
Total number of moves: 31
hosein@hoseinonthego:~/Emb/HW03/Q2$ S
```

Figure 2: RSA and Hanoi Tower

2 RSA Arm Ass

RSA Algorithm:

```
     .arch armv5t
.fpu softvfp
.eabi_attribute 20, 1
.eabi_attribute 21, 1
.eabi_attribute 23, 3
.eabi_attribute 24, 1
.eabi_attribute 25, 1
.eabi_attribute 26, 2
.eabi_attribute 30, 6
.eabi_attribute 34, 0
.eabi_attribute 18, 4
.file "MyRSA.c"
.text
.section .rodata
.align 2
.LC0:
.ascii "Enter the public key (n, e): \000"
.align 2
.LC1:
.ascii "%d %d\000"
.align 2
.LC2:
.ascii "Enter the encrypted messages separated by spaces: \000"
.align 2
.LC3:
.ascii " %[^\012]s\000"
.align 2
.LC4:
.ascii " \000"
.align 2
.LC5:
.ascii "p: %d, q: %d\012\000"
.align 2
.LC6:
.ascii "Private key d: %d\012\000"
.align 2
```

```
.LC7:
.ascii "Decrypted messages:\000"
.align 2
.LC8:
.ascii "%d \000"
.text
.align 2
.global main
.syntax unified
.arm
.type main, %function
main:
@ args = 0, pretend = 0, frame = 1448
@ frame_needed = 1, uses_anonymous_args = 0
push {r4, fp, lr}
add fp, sp, #8
sub sp, sp, #1440
sub sp, sp, #12
ldr r3, .L8
ldr r3, [r3]
str r3, [fp, #-16]
mov r3, #0
mov r3, #0
str r3, [fp, #-1440]
ldr r0, .L8+4
bl printf
sub r2, fp, #1440
sub r2, r2, #12
sub r3, fp, #1440
sub r3, r3, #12
sub r3, r3, #4
mov r1, r3
ldr r0, .L8+8
bl __isoc99_scanf
ldr r0, .L8+12
bl printf
sub r3, fp, #1016
mov r1, r3
ldr r0, .L8+16
```

```
bl __isoc99_scanf
sub r3, fp, #1016
ldr r1, .L8+20
mov r0, r3
bl strtok
str r0, [fp, #-1436]
b .L2
.L3:
ldr r4, [fp, #-1440]
add r3, r4, #1
str r3, [fp, #-1440]
ldr r0, [fp, #-1436]
bl atoi
mov r2, r0
lsl r3, r4, #2
sub r3, r3, #12
add r3, r3, fp
str r2, [r3, #-1404]
ldr r1, .L8+20
mov r0, #0
bl strtok
str r0, [fp, #-1436]
.L2:
ldr r3, [fp, #-1436]
cmp r3, #0
bne .L3
ldr r0, [fp, #-1456]
sub r2, fp, #1424
sub r2, r2, #12
sub r2, r2, #8
sub r3, fp, #1424
sub r3, r3, #12
sub r3, r3, #12
mov r1, r3
bl factorizeNumber
ldr r3, [fp, #-1448]
ldr r2, [fp, #-1444]
mov r1, r3
ldr r0, .L8+24
```

```
bl printf
ldr r3, [fp, #-1448]
sub r3, r3, #1
ldr r2, [fp, #-1444]
sub r2, r2, #1
mul r3, r2, r3
str r3, [fp, #-1428]
ldr r3, [fp, #-1452]
ldr r1, [fp, #-1428]
mov r0, r3
bl findModularInverse
str r0, [fp, #-1424]
ldr r1, [fp, #-1424]
ldr r0, .L8+28
bl printf
ldr r0, .L8+32
bl puts
mov r3, #0
str r3, [fp, #-1432]
b .L4
.L5:
ldr r3, [fp, #-1432]
lsl r3, r3, #2
sub r3, r3, #12
add r3, r3, fp
ldr r3, [r3, #-1404]
ldr r2, [fp, #-1456]
ldr r1, [fp, #-1424]
mov r0, r3
bl performModularExponentiation
str r0, [fp, #-1420]
ldr r1, [fp, #-1420]
ldr r0, .L8+36
bl printf
ldr r3, [fp, #-1432]
add r3, r3, #1
str r3, [fp, #-1432]
.L4:
ldr r2, [fp, #-1432]
ldr r3, [fp, #-1440]
cmp r2, r3
blt .L5
mov r3, #0
ldr r2, .L8
ldr r1, [r2]
ldr r2, [fp, #-16]
eors r1, r2, r1
mov r2, #0
beq .L7
bl __stack_chk_fail
.L7:
mov r0, r3
sub sp, fp, #8
@ sp needed
pop {r4, fp, pc}
.L9:
.align 2
.L8:
.word __stack_chk_guard
.word .LC0
.word .LC1
.word .LC2
.word .LC3
.word .LC4
.word .LC5
.word .LC6
.word .LC7
.word .LC8
.size main, .-main
.global __aeabi_idivmod
.align 2
.global calculateGCD
.syntax unified
.arm
.type calculateGCD, %function
calculateGCD:
@ args = 0, pretend = 0, frame = 16
@ frame_needed = 1, uses_anonymous_args = 0
```

```
push {fp, lr}
add fp, sp, #4
sub sp, sp, #16
str r0, [fp, #-16]
str r1, [fp, #-20]
b .L11
.L12:
ldr r3, [fp, #-20]
str r3, [fp, #-8]
ldr r3, [fp, #-16]
ldr r1, [fp, #-20]
mov r0, r3
bl __aeabi_idivmod
mov r3, r1
str r3, [fp, #-20]
ldr r3, [fp, #-8]
str r3, [fp, #-16]
.L11:
ldr r3, [fp, #-20]
cmp r3, #0
bne .L12
ldr r3, [fp, #-16]
mov r0, r3
sub sp, fp, #4
@ sp needed
pop {fp, pc}
.size calculateGCD, .-calculateGCD
.global __aeabi_idiv
.align 2
.global findModularInverse
.syntax unified
.arm
.type findModularInverse, %function
findModularInverse:
@ args = 0, pretend = 0, frame = 32
@ frame_needed = 1, uses_anonymous_args = 0
push {fp, lr}
add fp, sp, #4
sub sp, sp, #32
```

```
str r0, [fp, #-32]
str r1, [fp, #-36]
ldr r3, [fp, #-36]
str r3, [fp, #-16]
mov r3, #0
str r3, [fp, #-24]
mov r3, #1
str r3, [fp, #-20]
ldr r3, [fp, #-36]
cmp r3, #1
bne .L17
mov r3, #0
b .L16
.L18:
ldr r1, [fp, #-36]
ldr r0, [fp, #-32]
bl __aeabi_idiv
mov r3, r0
str r3, [fp, #-12]
ldr r3, [fp, #-36]
str r3, [fp, #-8]
ldr r3, [fp, #-32]
ldr r1, [fp, #-36]
mov r0, r3
bl __aeabi_idivmod
mov r3, r1
str r3, [fp, #-36]
ldr r3, [fp, #-8]
str r3, [fp, #-32]
ldr r3, [fp, #-24]
str r3, [fp, #-8]
ldr r3, [fp, #-12]
ldr r2, [fp, #-24]
mul r3, r2, r3
ldr r2, [fp, #-20]
sub r3, r2, r3
str r3, [fp, #-24]
ldr r3, [fp, #-8]
str r3, [fp, #-20]
```

```
.L17:
ldr r3, [fp, #-32]
cmp r3, #1
bgt .L18
ldr r3, [fp, #-20]
cmp r3, #0
bge .L19
ldr r2, [fp, #-20]
ldr r3, [fp, #-16]
add r3, r2, r3
str r3, [fp, #-20]
.L19:
ldr r3, [fp, #-20]
.L16:
mov r0, r3
sub sp, fp, #4
@ sp needed
pop {fp, pc}
.size findModularInverse, .-findModularInverse
.align 2
.global performModularExponentiation
.syntax unified
.arm
.type performModularExponentiation, %function
performModularExponentiation:
@ args = 0, pretend = 0, frame = 24
@ frame_needed = 1, uses_anonymous_args = 0
push {fp, lr}
add fp, sp, #4
sub sp, sp, #24
str r0, [fp, #-16]
str r1, [fp, #-20]
str r2, [fp, #-24]
mov r3, #1
str r3, [fp, #-8]
ldr r3, [fp, #-16]
ldr r1, [fp, #-24]
mov r0, r3
bl __aeabi_idivmod
```

```
mov r3, r1
str r3, [fp, #-16]
b .L21
.L23:
ldr r3, [fp, #-20]
cmp r3, #0
and r3, r3, #1
rsblt r3, r3, #0
cmp r3, #1
bne .L22
ldr r3, [fp, #-8]
ldr r2, [fp, #-16]
mul r3, r2, r3
ldr r1, [fp, #-24]
mov r0, r3
bl __aeabi_idivmod
mov r3, r1
str r3, [fp, #-8]
.L22:
ldr r3, [fp, #-20]
asr r3, r3, #1
str r3, [fp, #-20]
ldr r3, [fp, #-16]
mov r2, r3
mul r2, r3, r2
mov r3, r2
ldr r1, [fp, #-24]
mov r0, r3
bl __aeabi_idivmod
mov r3, r1
str r3, [fp, #-16]
.L21:
ldr r3, [fp, #-20]
cmp r3, #0
bgt .L23
ldr r3, [fp, #-8]
mov r0, r3
sub sp, fp, #4
@ sp needed
```

```
pop {fp, pc}
.size performModularExponentiation, .-performModularExponentiation
.global __aeabi_i2d
.global __aeabi_dcmpge
.align 2
.global factorizeNumber
.syntax unified
.arm
.type factorizeNumber, %function
factorizeNumber:
@ args = 0, pretend = 0, frame = 32
@ frame_needed = 1, uses_anonymous_args = 0
push {fp, lr}
add fp, sp, #4
sub sp, sp, #32
str r0, [fp, #-24]
str r1, [fp, #-28]
str r2, [fp, #-32]
ldr r0, [fp, #-24]
bl __aeabi_i2d
mov r2, r0
mov r3, r1
mov r0, r2
mov r1, r3
bl sqrt
str r0, [fp, #-12]
str r1, [fp, #-8]
mov r3, #2
str r3, [fp, #-16]
b .L26
.L29:
ldr r3, [fp, #-24]
ldr r1, [fp, #-16]
mov r0, r3
bl __aeabi_idivmod
mov r3, r1
cmp r3, #0
bne .L27
ldr r3, [fp, #-28]
ldr r2, [fp, #-16]
str r2, [r3]
ldr r1, [fp, #-16]
ldr r0, [fp, #-24]
bl __aeabi_idiv
mov r3, r0
mov r2, r3
ldr r3, [fp, #-32]
str r2, [r3]
b .L25
.L27:
ldr r3, [fp, #-16]
add r3, r3, #1
str r3, [fp, #-16]
.L26:
ldr r0, [fp, #-16]
bl __aeabi_i2d
mov r2, r0
mov r3, r1
sub r1, fp, #12
ldmia r1, {r0-r1}
bl __aeabi_dcmpge
mov r3, r0
cmp r3, #0
bne .L29
.L25:
sub sp, fp, #4
@ sp needed
pop {fp, pc}
.size factorizeNumber, .-factorizeNumber
.ident "GCC: (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0"
.section .note.GNU-stack,"",%progbits
```

```
     .arch armv5t
.fpu softvfp
.eabi_attribute 20, 1
.eabi_attribute 21, 1
.eabi_attribute 23, 3
```

```
.eabi_attribute 24, 1
.eabi_attribute 25, 1
.eabi_attribute 26, 2
.eabi_attribute 30, 6
.eabi_attribute 34, 0
.eabi_attribute 18, 4
.file "MyHanoi.c"
.text
.align 2
.global createStack
.syntax unified
.arm
.type createStack, %function
createStack:
@ args = 0, pretend = 0, frame = 16
@ frame_needed = 1, uses_anonymous_args = 0
push {fp, lr}
add fp, sp, #4
sub sp, sp, #16
str r0, [fp, #-16]
mov r0, #12
bl malloc
mov r3, r0
str r3, [fp, #-8]
ldr r3, [fp, #-8]
ldr r2, [fp, #-16]
str r2, [r3]
ldr r3, [fp, #-8]
mvn r2, #0
str r2, [r3, #4]
ldr r3, [fp, #-8]
ldr r3, [r3]
lsl r3, r3, #2
mov r0, r3
bl malloc
mov r3, r0
mov r2, r3
ldr r3, [fp, #-8]
str r2, [r3, #8]
```

```
ldr r3, [fp, #-8]
mov r0, r3
sub sp, fp, #4
@ sp needed
pop {fp, pc}
.size createStack, .-createStack
.align 2
.global isFull
.syntax unified
.arm
.type isFull, %function
isFull:
@ args = 0, pretend = 0, frame = 8
@ frame_needed = 1, uses_anonymous_args = 0
@ link register save eliminated.
str fp, [sp, #-4]!
add fp, sp, #0
sub sp, sp, #12
str r0, [fp, #-8]
ldr r3, [fp, #-8]
ldr r2, [r3, #4]
ldr r3, [fp, #-8]
ldr r3, [r3]
sub r3, r3, #1
cmp r2, r3
moveq r3, #1
movne r3, #0
and r3, r3, #255
mov r0, r3
add sp, fp, #0
@ sp needed
ldr fp, [sp], #4
bx lr
.size isFull, .-isFull
.align 2
.global isEmpty
.syntax unified
.arm
.type isEmpty, %function
```

```
isEmpty:
@ args = 0, pretend = 0, frame = 8
@ frame_needed = 1, uses_anonymous_args = 0
@ link register save eliminated.
str fp, [sp, #-4]!
add fp, sp, #0
sub sp, sp, #12
str r0, [fp, #-8]
ldr r3, [fp, #-8]
ldr r3, [r3, #4]
cmn r3, #1
moveq r3, #1
movne r3, #0
and r3, r3, #255
mov r0, r3
add sp, fp, #0
@ sp needed
ldr fp, [sp], #4
bx lr
.size isEmpty, .-isEmpty
.align 2
.global push
.syntax unified
.arm
.type push, %function
push:
@ args = 0, pretend = 0, frame = 8
@ frame_needed = 1, uses_anonymous_args = 0
push {fp, lr}
add fp, sp, #4
sub sp, sp, #8
str r0, [fp, #-8]
str r1, [fp, #-12]
ldr r0, [fp, #-8]
bl isFull
mov r3, r0
cmp r3, #0
bne .L10
ldr r3, [fp, #-8]
ldr r2, [r3, #8]
ldr r3, [fp, #-8]
ldr r3, [r3, #4]
add r1, r3, #1
ldr r3, [fp, #-8]
str r1, [r3, #4]
ldr r3, [fp, #-8]
ldr r3, [r3, #4]
lsl r3, r3, #2
add r3, r2, r3
ldr r2, [fp, #-12]
str r2, [r3]
b .L7
.L10:
nop
.L7:
sub sp, fp, #4
@ sp needed
pop {fp, pc}
.size push, .-push
.align 2
.global pop
.syntax unified
.arm
.type pop, %function
pop:
@ args = 0, pretend = 0, frame = 8
@ frame_needed = 1, uses_anonymous_args = 0
push {fp, lr}
add fp, sp, #4
sub sp, sp, #8
str r0, [fp, #-8]
ldr r0, [fp, #-8]
bl isEmpty
mov r3, r0
cmp r3, #0
beq .L12
mvn r3, #0
b .L13
```

```
.L12:
ldr r3, [fp, #-8]
ldr r2, [r3, #8]
ldr r3, [fp, #-8]
ldr r3, [r3, #4]
sub r0, r3, #1
ldr r1, [fp, #-8]
str r0, [r1, #4]
lsl r3, r3, #2
add r3, r2, r3
ldr r3, [r3]
.L13:
mov r0, r3
sub sp, fp, #4
@ sp needed
pop {fp, pc}
.size pop, .-pop
.section .rodata
.align 2
.LC0:
.ascii "Move disk %d from rod %c to rod %c\012\000"
.text
.align 2
.global moveDisk
.syntax unified
.arm
.type moveDisk, %function
moveDisk:
@ args = 0, pretend = 0, frame = 16
@ frame_needed = 1, uses_anonymous_args = 0
push {fp, lr}
add fp, sp, #4
sub sp, sp, #16
str r0, [fp, #-8]
str r1, [fp, #-12]
str r2, [fp, #-16]
str r3, [fp, #-20]
ldr r0, [fp, #-8]
bl pop
```

```
mov r3, r0
mov r1, r3
ldr r0, [fp, #-12]
bl push
ldr r3, [fp, #-8]
ldr r3, [r3]
add r2, r3, #65
ldr r3, [fp, #-8]
ldr r3, [r3, #4]
sub r3, r2, r3
sub r1, r3, #2
ldr r3, [fp, #-12]
ldr r3, [r3]
add r2, r3, #65
ldr r3, [fp, #-12]
ldr r3, [r3, #4]
sub r3, r2, r3
sub r3, r3, #1
mov r2, r1
ldr r1, [fp, #-16]
ldr r0, .L15
bl printf
ldr r3, [fp, #-20]
ldr r3, [r3]
add r2, r3, #1
ldr r3, [fp, #-20]
str r2, [r3]
nop
sub sp, fp, #4
@ sp needed
pop {fp, pc}
.L16:
.align 2
.L15:
.word .LC0
.size moveDisk, .-moveDisk
.align 2
.global towerOfHanoi
.syntax unified
```

```
.arm
.type towerOfHanoi, %function
towerOfHanoi:
@ args = 4, pretend = 0, frame = 16
@ frame_needed = 1, uses_anonymous_args = 0
push {fp, lr}
add fp, sp, #4
sub sp, sp, #24
str r0, [fp, #-8]
str r1, [fp, #-12]
str r2, [fp, #-16]
str r3, [fp, #-20]
ldr r3, [fp, #-8]
cmp r3, #1
bne .L18
ldr r3, [fp, #4]
mov r2, #1
ldr r1, [fp, #-16]
ldr r0, [fp, #-12]
bl moveDisk
b .L17
.L18:
ldr r3, [fp, #-8]
sub r0, r3, #1
ldr r3, [fp, #4]
str r3, [sp]
ldr r3, [fp, #-16]
ldr r2, [fp, #-20]
ldr r1, [fp, #-12]
bl towerOfHanoi
ldr r3, [fp, #4]
ldr r2, [fp, #-8]
ldr r1, [fp, #-16]
ldr r0, [fp, #-12]
bl moveDisk
ldr r3, [fp, #-8]
sub r0, r3, #1
ldr r3, [fp, #4]
str r3, [sp]
```

```
ldr r3, [fp, #-12]
ldr r2, [fp, #-16]
ldr r1, [fp, #-20]
bl towerOfHanoi
.L17:
sub sp, fp, #4
@ sp needed
pop {fp, pc}
.size towerOfHanoi, .-towerOfHanoi
.section .rodata
.align 2
.LC1:
.ascii "Enter the number of disks: \000"
.align 2
.LC2:
.ascii "%d\000"
.align 2
.LC3:
.ascii "Total number of moves: %d\012\000"
.text
.align 2
.global main
.syntax unified
.arm
.type main, %function
main:
@ args = 0, pretend = 0, frame = 32
@ frame_needed = 1, uses_anonymous_args = 0
push {fp, lr}
add fp, sp, #4
sub sp, sp, #40
ldr r3, .L25
ldr r3, [r3]
str r3, [fp, #-8]
mov r3, #0
mov r3, #0
str r3, [fp, #-28]
ldr r0, .L25+4
bl printf
```

```
sub r3, fp, #32
mov r1, r3
ldr r0, .L25+8
bl __isoc99_scanf
ldr r3, [fp, #-32]
mov r0, r3
bl createStack
str r0, [fp, #-20]
ldr r3, [fp, #-32]
mov r0, r3
bl createStack
str r0, [fp, #-16]
ldr r3, [fp, #-32]
mov r0, r3
bl createStack
str r0, [fp, #-12]
ldr r3, [fp, #-32]
str r3, [fp, #-24]
b .L21
.L22:
ldr r1, [fp, #-24]
ldr r0, [fp, #-20]
bl push
ldr r3, [fp, #-24]
sub r3, r3, #1
str r3, [fp, #-24]
.L21:
ldr r3, [fp, #-24]
cmp r3, #0
bgt .L22
ldr r0, [fp, #-32]
sub r3, fp, #28
str r3, [sp]
ldr r3, [fp, #-12]
ldr r2, [fp, #-16]
ldr r1, [fp, #-20]
bl towerOfHanoi
ldr r3, [fp, #-28]
mov r1, r3
```

```
ldr r0, .L25+12
bl printf
ldr r3, [fp, #-20]
ldr r3, [r3, #8]
mov r0, r3
bl free
ldr r0, [fp, #-20]
bl free
ldr r3, [fp, #-16]
ldr r3, [r3, #8]
mov r0, r3
bl free
ldr r0, [fp, #-16]
bl free
ldr r3, [fp, #-12]
ldr r3, [r3, #8]
mov r0, r3
bl free
ldr r0, [fp, #-12]
bl free
mov r3, #0
ldr r2, .L25
ldr r1, [r2]
ldr r2, [fp, #-8]
eors r1, r2, r1
mov r2, #0
beq .L24
bl __stack_chk_fail
.L24:
mov r0, r3
sub sp, fp, #4
@ sp needed
pop {fp, pc}
.L26:
.align 2
.L25:
.word __stack_chk_guard
.word .LC1
.word .LC2
```

```
.word .LC3
.size main, .-main
.ident "GCC: (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0"
.section .note.GNU-stack,"",%progbits
```

It is notable that I wrote the Hanoi tower question using a non-recursive function because it is not possible to have recursion in assembly and it should handle by