Write a program to transfer the data between two registers

```
; Program to transfer data between registers and perform basic operations

; Initialize registers
MOV A, #0x05 ; Move the value 05h into register A
MOV B, #0x03 ; Move the value 03h into register B

; Transfer data from register A to register B
MOV B, A    ; Move the value from register A into register B

; Perform addition
ADD A, B    ; Add the value in register B to register A

; Perform subtraction
SUBB A, B   ; Subtract the value in register B from register A

; Perform bitwise AND
ANL A, B    ; Perform bitwise AND operation between register A and B

; Perform bitwise OR
ORL A, B    ; Perform bitwise OR operation between register A and B

; Store the result in register C
MOV C, A    ; Move the result from register A into register C

END         ; End of program
```

In this program:

1. We initialize registers A and B with values 05h and 03h, respectively.
2. We transfer the value from register A to register B using the MOV instruction.
3. We perform addition, subtraction, bitwise AND, and bitwise OR operations using the ADD, SUBB, ANL, and ORL instructions, respectively.
4. We store the result of the operations in register C.

5. Finally, we end the program using the END instruction.

To run this program in EdSim51:

1. Open EdSim51 and create a new project.
2. Write the above code in the editor.
3. Assemble the program by clicking the "Assemble" button.
4. Run the program by clicking the "Run" button.
5. Observe the values in registers A, B, and C in the "Registers" window.

Note: EdSim51 uses the Intel 8051 microcontroller architecture, which has eight 8-bit registers (A, B, C, D, E, H, L, and PSW).

Perform the communication between Arduino and Raspberry PI using Zigbee.

Hardware Requirements:

- Arduino Board (e.g., Arduino Uno)
- Raspberry Pi Board (e.g., Raspberry Pi 3)
- Zigbee Modules (e.g., XBee Series 2)
- Breadboard and Jumper Wires

Software Requirements:

- Arduino IDE
- Raspberry Pi OS (e.g., Raspbian)
- Python Programming Language

Step 1: Connect Zigbee Modules to Arduino and Raspberry Pi

- Connect the Zigbee module to the Arduino board's serial pins (RX and TX).
- Connect the Zigbee module to the Raspberry Pi's USB port using a USB-to-TTL serial adapter.

Step 2: Configure Zigbee Modules

- Configure the Zigbee modules to operate in the same network and with the same baud rate (e.g., 9600).
- Set the destination address of the Arduino's Zigbee module to the Raspberry Pi's Zigbee module's address.

Step 3: Write Arduino Code

- Write a simple Arduino sketch to send data to the Raspberry Pi using the Zigbee module.
- Use the SoftwareSerial library to communicate with the Zigbee module.

Step 4: Write Raspberry Pi Code

- Write a Python script to receive data from the Arduino using the Zigbee module.
- Use the PySerial library to communicate with the Zigbee module.

Step 5: Test the Communication

- Upload the Arduino sketch and run the Python script.
- Send data from the Arduino to the Raspberry Pi using the Zigbee modules.
- Verify that the data is received correctly on the Raspberry Pi.

Example Arduino Code:

```
#include <SoftwareSerial.h>

SoftwareSerial zigbee(2, 3); // RX, TX

void setup() {
  zigbee.begin(9600);
}

void loop() {
  zigbee.println("Hello from Arduino!");
  delay(1000);
}
```

Example Raspberry Pi Code (Python):

```
import serial

zigbee = serial.Serial('/dev/ttyUSB0', 9600)

while True:
  data = zigbee.readline()
  print(data.decode())
```

Note: Replace '/dev/ttyUSB0' with the actual device file of your USB-to-TTL serial adapter.

Perform the communication between Arduino and Raspberry PI using Zigbee.

Hardware Requirements:

- Arduino Board (e.g., Arduino Uno)
- Raspberry Pi Board (e.g., Raspberry Pi 3)
- Zigbee Modules (e.g., XBee Series 2)
- Breadboard and Jumper Wires

Software Requirements:

- Arduino IDE
- Raspberry Pi OS (e.g., Raspbian)
- Python Programming Language

Step 1: Connect Zigbee Modules to Arduino and Raspberry Pi

- Connect the Zigbee module to the Arduino board's serial pins (RX and TX).
- Connect the Zigbee module to the Raspberry Pi's USB port using a USB-to-TTL serial adapter.

Step 2: Configure Zigbee Modules

- Configure the Zigbee modules to operate in the same network and with the same baud rate (e.g., 9600).
- Set the destination address of the Arduino's Zigbee module to the Raspberry Pi's Zigbee module's address.

Step 3: Write Arduino Code

- Write a simple Arduino sketch to send data to the Raspberry Pi using the Zigbee module.
- Use the SoftwareSerial library to communicate with the Zigbee module.

Step 4: Write Raspberry Pi Code

- Write a Python script to receive data from the Arduino using the Zigbee module.
- Use the PySerial library to communicate with the Zigbee module.

Step 5: Test the Communication

- Upload the Arduino sketch and run the Python script.
- Send data from the Arduino to the Raspberry Pi using the Zigbee modules.
- Verify that the data is received correctly on the Raspberry Pi.

Example Arduino Code:

```
#include <SoftwareSerial.h>

SoftwareSerial zigbee(2, 3); // RX, TX

void setup() {
  zigbee.begin(9600);
}
```

```
void loop() {
  zigbee.println("Hello from Arduino!");
  delay(1000);
}
```

Example Raspberry Pi Code (Python):

```
import serial

zigbee = serial.Serial('/dev/ttyUSB0', 9600)

while True:
  data = zigbee.readline()
  print(data.decode())
```

Note: Replace '/dev/ttyUSB0' with the actual device file of your USB-to-TTL serial adapter.

Perform the communication between Arduino and Raspberry PI using GSM.

Hardware Requirements:

- Arduino Board (e.g., Arduino Uno)
- Raspberry Pi Board (e.g., Raspberry Pi 3)
- GSM Modules (e.g., SIM900A for Arduino and SIM800L for Raspberry Pi)
- Breadboard and Jumper Wires
- SIM Cards with SMS and GPRS capabilities

Software Requirements:

- Arduino IDE
- Raspberry Pi OS (e.g., Raspbian)
- Python Programming Language
- AT Command Library for Arduino (e.g., Sim900a.h)

Step 1: Connect GSM Modules to Arduino and Raspberry Pi

- Connect the GSM module to the Arduino board's serial pins (RX and TX).
- Connect the GSM module to the Raspberry Pi's USB port using a USB-to-TTL serial adapter.

Step 2: Configure GSM Modules

- Configure the GSM modules to operate in the same network and with the same baud rate (e.g., 9600).
- Set the APN (Access Point Name) for GPRS connectivity.

Step 3: Write Arduino Code

- Write a simple Arduino sketch to send SMS or make HTTP requests using the GSM module.
- Use the AT Command Library to communicate with the GSM module.

Step 4: Write Raspberry Pi Code

- Write a Python script to receive SMS or HTTP requests using the GSM module.
- Use the PySerial library to communicate with the GSM module.

Step 5: Test the Communication

- Upload the Arduino sketch and run the Python script.
- Send SMS or make HTTP requests from the Arduino to the Raspberry Pi using the GSM modules.
- Verify that the data is received correctly on the Raspberry Pi.

Example Arduino Code:

```
#include <Sim900a.h>

Sim900a gsm;

void setup() {
  gsm.begin(9600);
}

void loop() {
  gsm.sendSMS("Raspberry Pi", "Hello from Arduino!");
  delay(1000);
}
```

Example Raspberry Pi Code (Python):

```
import serial

gsm = serial.Serial('/dev/ttyUSB0', 9600)

while True:
  data = gsm.readline()
  print(data.decode())
```

Note: Replace '/dev/ttyUSB0' with the actual device file of your USB-to-TTL serial adapter.

Write a program to perform logical operations using Embedded C.

Here is an example program in Embedded C that performs logical operations using simple variables:

```c
#include <stdio.h>

int main() {
    int a = 5;
    int b = 3;

    // Logical AND
    int and_result = a && b;

    // Logical OR
    int or_result = a || b;

    // Logical NOT
    int not_result = !a;

    printf("a = %d, b = %d\n", a, b);
    printf("a AND b = %d\n", and_result);
    printf("a OR b = %d\n", or_result);
    printf("NOT a = %d\n", not_result);

    return 0;
}
```

In this program, we perform the following logical operations:

1. Logical AND (&&): Evaluates to 1 if both a and b are non-zero.
2. Logical OR (||): Evaluates to 1 if either a or b is non-zero.
3. Logical NOT (!): Evaluates to 1 if a is zero, and 0 if a is non-zero.

Note: The results of the logical operations are printed to the console using printf.

Write a program to perform the communication between IoT and Zigbee.

IoT (ESP8266/ESP32) Code:

```
#include <WiFi.h>
#include <SoftwareSerial.h>

SoftwareSerial zigbee(2, 3); // RX, TX

void setup() {
  Serial.begin(115200);
  zigbee.begin(9600);
  WiFi.begin("your_wifi_ssid", "your_wifi_password");
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi...");
  }
  Serial.println("Connected to WiFi");
}

void loop() {
  if (zigbee.available()) {
    char c = zigbee.read();
    Serial.print(c);
  }
  if (Serial.available()) {
    char c = Serial.read();
    zigbee.print(c);
  }
}
```

Zigbee (XBee) Code:

```
#include <SoftwareSerial.h>

SoftwareSerial iot(2, 3); // RX, TX
```

```
void setup() {
  Serial.begin(9600);
  iot.begin(9600);
}

void loop() {
  if (iot.available()) {
    char c = iot.read();
    Serial.print(c);
  }
  if (Serial.available()) {
    char c = Serial.read();
    iot.print(c);
  }
}
```

In this program, we establish a serial communication between the IoT device (ESP8266/ESP32) and the Zigbee module (XBee). The IoT device connects to a WiFi network and acts as a bridge between the Zigbee network and the internet.

Note: Replace "your_wifi_ssid" and "your_wifi_password" with your actual WiFi credentials.

This program assumes that the XBee module is configured to operate in transparent mode, where it simply forwards data between the IoT device and the Zigbee network. The IoT device can then send and receive data to/from the Zigbee network using the serial communication established in the program.

Write a program to compare two numbers in 8051 using simulator

```
; Compare two numbers in 8051

ORG 0000H

MOV A, #0x05 ; Move 05h into accumulator
MOV B, #0x03 ; Move 03h into register B

; Compare A and B
CMP A, B

; Jump to label if A is greater than B
JNC LABEL1

; Jump to label if A is less than B
JC LABEL2

; If A is equal to B, continue execution

LABEL1:
MOV A, #0x01 ; Move 01h into accumulator if A is greater than B
SJMP END

LABEL2:
MOV A, #0x02 ; Move 02h into accumulator if A is less than B
SJMP END

END:
NOP ; No operation
```

In this program:

1. We move two numbers (05h and 03h) into the accumulator (A) and register B, respectively.
2. We compare the values in A and B using the CMP instruction.
3. If A is greater than B, we jump to label LABEL1.

4. If A is less than B, we jump to label LABEL2.
5. If A is equal to B, we continue execution.
6. At LABEL1 and LABEL2, we move different values into the accumulator based on the comparison result.
7. Finally, we end the program with a NOP instruction.

To run this program in a simulator:

1. Open the simulator software (e.g., EdSim51).
2. Create a new project and paste the above code into the editor.
3. Assemble the program by clicking the "Assemble" button.
4. Run the program by clicking the "Run" button.
5. Observe the values in the accumulator and registers in the simulator's debug window.

Note: The simulator will execute the program step-by-step, allowing you to examine the values in the registers and accumulator at each step.

Write a program to perform comparison operations using Embedded C.

```
#include <stdio.h>

void compare(int a, int b) {
    if (a > b) {
        printf("%d is greater than %d\n", a, b);
    } else if (a < b) {
        printf("%d is less than %d\n", a, b);
    } else {
        printf("%d is equal to %d\n", a, b);
    }

    if (a >= b) {
        printf("%d is greater than or equal to %d\n", a, b);
    } else {
        printf("%d is less than or equal to %d\n", a, b);
```

```
    }
}

int main() {
    int num1 = 10;
    int num2 = 20;

    compare(num1, num2);

    return 0;
}
```

Write a program to perform the communication between IoT and GSM.

```
#include <SoftwareSerial.h>

// Define the pins for the GSM module
#define GSM_TX 2
#define GSM_RX 3

SoftwareSerial gsm(GSM_TX, GSM_RX);

void setup() {
  Serial.begin(115200);
  gsm.begin(9600);
}

void loop() {
  // Send SMS using GSM module
  gsm.println("AT+CMGF=1");
  delay(1000);
  gsm.println("AT+CMGS=\"+1234567890\"");
  delay(1000);
  gsm.println("Hello from IoT!");
```

```
  delay(1000);
  gsm.println((char)26);

  // Receive SMS using GSM module
  if (gsm.available()) {
    String message = gsm.readString();
    Serial.println(message);
  }

  // Make a call using GSM module
  gsm.println("ATD+1234567890;");
  delay(1000);

  // Receive data from IoT device (e.g., sensor data)
  int sensorData = analogRead(A0);
  gsm.println(sensorData);
}
```

In this program:

1. We define the pins for the GSM module and create a SoftwareSerial object to communicate with it.
2. In the setup() function, we initialize the serial communication with the GSM module.
3. In the loop() function, we:
   - Send an SMS using the GSM module.
   - Receive an SMS using the GSM module.
   - Make a call using the GSM module.
   - Send sensor data from the IoT device to the GSM module.

Note: Replace "+1234567890" with the actual phone number you want to send SMS or make a call to. Also, make sure to install the necessary libraries and configure the GSM module correctly.

Write a python programming to perform data transfer operation in arduino.

```
import serial

# Open the serial port
ser = serial.Serial('COM3', 9600)  # Replace COM3 with your Arduino's serial port

# Send data to Arduino
data = "Hello, Arduino!"
ser.write(data.encode())

# Receive data from Arduino
received_data = ser.readline().decode()
print("Received from Arduino:", received_data)

# Close the serial port
ser.close()
```

In this program:

1. We import the PySerial library.
2. We open the serial port connected to the Arduino board.
3. We send a string "Hello, Arduino!" to the Arduino board.
4. We receive data from the Arduino board and print it.
5. We close the serial port.

Note: Replace 'COM3' with the actual serial port of your Arduino board.

Arduino Code:

```
void setup() {
  Serial.begin(9600);
}
```

```
void loop() {
  if (Serial.available()) {
    String data = Serial.readString();
    Serial.println("Received from Python: " + data);
    Serial.println("Hello, Python!");
  }
}
```

In this Arduino code:

1. We initialize the serial communication at 9600 baud.
2. We check if data is available in the serial buffer.
3. If data is available, we read it and print a message.
4. We send a response back to the Python program.