1.BASIC ARITHMETIC AND LOGICAL OPERATIONS – 8051µC

ADDITION:

```
MOV DPTR,#4200
MOVX A,@DPTR
MOV B, A
INC DPTR
MOVX A,@DPTR
ADD A, B
INC DPTR
MOVX @DPTR,A
SJMP HLT
```

SUBTRACTION:

```
MOV DPTR,#4200
MOVX A,@DPTR
MOV B, A
INC DPTR
MOVX A,@DPTR
SUBB A, B
INC DPTR
MOVX @DPTR,A
SJMP HLT
```

MULTIPLICATION:

```
MOV A, #data1
MOV B, #data2
MUL AB
MOV DPTR, #4500H
MOVX@DPTR, A
INC DPTR
MOV A,B
MOVX@DPTR, A
SJMP HLT
```

DIVISION:

```
MOV A, #data1
MOV B, #data2
DIV AB
MOV DPTR, #4500H
MOVX@DPTR, A
INC DPTR
MOV A, B
MOVX@DPTR, A
SJMP HLT
```

AND:

```
MOV A, #data1
MOV B, #data2
ANL A B
MOV DPTR, #4500H
MOVX@DPTR, A
SJMP HLT
```

OR:

```
MOV A, #data1
MOV B, #data2
ORL A B
MOV DPTR, #4500H
MOVX@DPTR, A
SJMP HLT
```

Xor

MOV A, #data1
MOV B, #data2
XRL A B
MOV DPTR, #4500H
MOVX@DPTR, A
SJMP HLT

NAND

MOV A, #data1
MOV B, #data2
ANL A B
CPL A
MOV DPTR, #4500H
MOVX@DPTR, A
SJMP HLT

NOR

MOV A, #data1
MOV B, #data2
ORL A B
CPL A
MOV DPTR, #4500H
MOVX@DPTR, A
SJMP HLT

CPL and 2's CPL

MOV A, #data1
CPL A
MOV DPTR, #4500H
MOVX@DPTR, A
INC DPTR
INC A
MOVX@DPTR, A
SJMP HLT

2.Simple arithmetic operations using EdSim-51 Simulator

2b.1 Blinking of LEDs in EdSim-51 for desire pattern

```
start:mov a,p2
mov p1,p2
acall delay
cpl a
mov p1,a
acall delay
sjmp start
delay:
mov R1,#0fh
WAIT1:DJNZ R1,WAIT1
ret
End
```

2b.2 Blinking of LEDs in 8051 µC Development Kit for desire pattern

ADDRESS	LABEL	MNEMONICS	
4100	Start:	MOV P1,#0F	
4103		LCALL Delay	
4106		MOV A, P1	
4108		CPL A	
4109		MOV P1, A	
410B		LCALL Delay	
410E		SJMP Start	
4110	Delay:	MOV R1,#FF	
4112	L1	MOV R2,#FF	
4114	L2	DJNZ R2, L2	
4116		DJNZ R1, L1	
4118		RET	

2c.Data transfer between register and Memory using Simulator

Same order:

```
mov r0,#30h
mov r1,#40h
mov r2,#05
loop1:mov a, @r0
mov @r1,a
inc r0
inc r1
djnz r2,loop1
End
```

Reverse order:

```
mov r0,#30h
mov r1,#44h
mov r2,#05
loop1:mov a, @r0
mov @r1,a
inc r0
dec r1
djnz r2,loop1
End
```

3a.Basic and Arithmetic programming Using Embedded C

```
#include<reg51.h>
void main(void)
{
unsigned char a,b,c,d,Y;
while(1)
{
P1 = 0xff; //data
P2 = 0Xff; //data
a=P1;
b=P2;
c = a+b;
d = a-b;
Y = (c*d)/2;
P3 = Y;
}
}
```

3b.Using microcontroller peripherals to blink LED

```
#include <REG51.H>
void main(void)
{
unsigned int i;
while(1) {
P1=0x00; //pattern1
P2=0x00; //pattern1
for(i=0;i<65535;i++);
P1=0xff; //pattern2
P2=0xff; //pattern2
for(i=0;i<65535;i++);
}
}
```

4.Programming an Arduino

a. To blink built in LED

```
void setup() {
  pinMode(13, OUTPUT);  // Configure pin 13 as an output
}
void loop() {
  // Turn ON the LED using a HIGH signal
  digitalWrite(13, HIGH);
  delay(1000);                    // Wait for 1 second
  // Turn OFF the LED using a LOW signal
  digitalWrite(13, LOW);
  delay(1000);                    // Wait for 1 second
}
```

b. To blink external LED

```
int myled = 9;
void setup() {
  pinMode(myled, OUTPUT);      // Configure pin 9 as an output

}
void loop() {
  digitalWrite(myled, HIGH);   // Turn the LED ON
  delay(1000);                // Wait for 1 second
  digitalWrite(myled, LOW);    // Turn the LED OFF
  delay(1000);                // Wait for 1 second
}
```

c. IR sensor interfacing

```
int ledPin = 13;
int sensorPin = 8;  //IR sensor Output connected to pin 8 in arduino
int obstacleDetected= LOW; // LOW refers to no obstacle
void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(sensorPin, INPUT); //pin 49 receives IR sensor output
  Serial.begin(9600);
}
void loop()
{
  obstacleDetected= digitalRead(sensorPin);
  if (obstacleDetected == LOW) {
    Serial.println(“Stop! obstacle detected”);
    digitalWrite(ledPin, HIGH);
  }
  else {
    Serial.println(“No Obstacle, Go ahead!”);
    digitalWrite(ledPin, LOW);
  }
  delay(200); }
```

d.Ultrasonic Sensor

```
const int trigPin = 9;
const int echoPin = 10;
long duration;
int distance;

void setup() {
  Serial.begin(9600);
  pinMode(trigPin, OUTPUT);  // Set the trigger pin as output
  pinMode(echoPin, INPUT);   // Set the echo pin as input
}

void loop() {
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  duration = pulseIn(echoPin, HIGH);
  distance = duration * 0.017;
  Serial.print("Distance: ");
  Serial.print(distance);
  Serial.println(" cm");

  delay(500);
}
```

e.DHT Sensor

```
#include "DHT.h"

#define DHTPIN 2       // Data pin connected to digital pin 2
#define DHTTYPE DHT11  // Change to DHT22 if using DHT22
DHT dht(DHTPIN, DHTTYPE);

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

```
void loop() {
  delay(2000); // Delay between reads (DHT11 needs ~2s)

  float humidity = dht.readHumidity();
  float temperature = dht.readTemperature();
  if (isnan(humidity) || isnan(temperature)) {
    Serial.println("Failed to read from DHT sensor!");
    return;
  }

  Serial.print("Humidity: ");
  Serial.print(humidity);
  Serial.print(" %\t");

  Serial.print("Temperature: ");
  Serial.print(temperature);
  Serial.println(" °C");
}
```

5.IoT enabled real-time monitoring of sensor data using ESP32 (DHT)

```
#include "ThingSpeak.h"
#include <WiFi.h>
#include "DHT.h"

char ssid[] = "OnePlusNord4"; // Your WiFi SSID
char pass[] = "password"; // Your WiFi password
WiFiClient client;

unsigned long myChannelField = number; // Channel ID
const int TemperatureField = 1; // Field for temperature data
const int HumidityField = 2; // Field for humidity data
const char* myWriteAPIKey = "paste here"; // Your write AP
IKey

const int out = 23; // Pin for temperature sensor data
float temperature = 0; // Initialize temperature
DHT dht(23, DHT11);

void setup()
{ Serial.begin(115200);
  pinMode(out, INPUT); // Set pin mode to input for temperature
  sensor ThingSpeak.begin(client);
  dht.begin();
  delay(500); }

void loop()
{
  if (WiFi.status() != WL_CONNECTED)
  {Serial.print("Attempting to connect to SSID: ");
    Serial.println(ssid);
    while (WiFi.status() != WL_CONNECTED)
    { WiFi.begin(ssid, pass);
      Serial.print(".");
      delay(5000);
    }
    Serial.println("\nConnected.");
  }

  // Read sensor values
  float temperature = dht.readTemperature();
  float humidity = dht.readHumidity();

  Serial.print("Temperature: ");
  Serial.print(temperature);
  Serial.println(" °C");
  Serial.print("Humidity ");
  Serial.print(humidity);
  Serial.println(" g.m-3");
// Write temperature to ThingSpeak
  ThingSpeak.writeField(myChannelField, TemperatureField, temperature,
myWriteAPIKey); // Write temperature to ThingSpeak
  ThingSpeak.writeField(myChannelField, HumidityField, humidity, myWriteAPIKey); //
Write humidity to ThingSpeak
  delay(100);
}
```

(Ultrasonic)

```
#include <WiFi.h>
#include "ThingSpeak.h"

// WiFi credentials
char ssid[] = "";
char pass[] = "";

// ThingSpeak configuration
unsigned long myChannelField = ;
const int DistanceField = 1;  // Field 1 will store distance
const char* myWriteAPIKey = "";

// Ultrasonic sensor pins
const int TRIG_PIN = 23;
const int ECHO_PIN = 22;

WiFiClient client;

void setup() {
  Serial.begin(115200);

  // Configure ultrasonic pins
  pinMode(TRIG_PIN, OUTPUT);
  pinMode(ECHO_PIN, INPUT);

  // Connect to WiFi
  WiFi.begin(ssid, pass);
  Serial.print("Connecting to WiFi");
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.print(".");
  }
  Serial.println("\nWiFi connected.");

  // Initialize ThingSpeak
  ThingSpeak.begin(client);
}

void loop() {
  // Ensure WiFi connection
  if (WiFi.status() != WL_CONNECTED) {
    Serial.println("Reconnecting to WiFi...");
    WiFi.begin(ssid, pass);
    while (WiFi.status() != WL_CONNECTED) {
      delay(1000);
      Serial.print(".");
    }
    Serial.println("\nReconnected.");
  }

  // Trigger pulse
  digitalWrite(TRIG_PIN, LOW);
  delayMicroseconds(2);
  digitalWrite(TRIG_PIN, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIG_PIN, LOW);

  // Read echo time
  long duration_us = pulseIn(ECHO_PIN, HIGH);

  // Calculate distance (speed of sound = 343 m/s)
  float distance_cm = 0.017 * duration_us;

  // Print to Serial Monitor
  Serial.print("Distance: ");
  Serial.print(distance_cm);
  Serial.println(" cm");

  // Send to ThingSpeak
  int statusCode = ThingSpeak.writeField(myChannelField, DistanceField, distance_cm,
myWriteAPIKey);
  delay(1000);  // ThingSpeak allows update every 15 sec
}
```

IR Sensor

```
#include <WiFi.h>
#include "ThingSpeak.h"

// WiFi credentials
char ssid[] = "YOUR_WIFI_SSID";
char pass[] = "YOUR_WIFI_PASSWORD";

// ThingSpeak configuration
unsigned long myChannelField = YOUR_CHANNEL_ID;
const int IRField = 1;  // Field 1 will store IR sensor reading
const char* myWriteAPIKey = "YOUR_API_KEY";

// IR sensor pin
const int IR_PIN = 22;  // Connect digital output of IR sensor here

WiFiClient client;

void setup() {
  Serial.begin(115200);
  pinMode(IR_PIN, INPUT);

  // Connect to WiFi
  WiFi.begin(ssid, pass);
  Serial.print("Connecting to WiFi");
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.print(".");
  }
  Serial.println("\nWiFi connected.");

  // Initialize ThingSpeak
  ThingSpeak.begin(client);
}

void loop() {
  // Ensure WiFi connection
  if (WiFi.status() != WL_CONNECTED) {
    Serial.println("Reconnecting to WiFi...");
    WiFi.begin(ssid, pass);
    while (WiFi.status() != WL_CONNECTED) {
      delay(1000);
      Serial.print(".");
    }
    Serial.println("\nReconnected.");
  }

  // Read IR sensor
  int irValue = digitalRead(IR_PIN); // 0 = object detected, 1 = no object

  // Print to Serial Monitor
  Serial.print("IR Sensor: ");
  Serial.println(irValue == 0 ? "Object Detected" : "No Object");

  // Send to ThingSpeak
  int statusCode = ThingSpeak.writeField(myChannelField, IRField, irValue,
myWriteAPIKey);

  if (statusCode == 200) {
    Serial.println("Data sent to ThingSpeak.");
  } else {
    Serial.print("Failed to send data. HTTP error code: ");
    Serial.println(statusCode);
  }

  delay(15000);  // ThingSpeak update limit
}
```

6. IoT-enabled real-time monitoring of sensor data using Raspberry PI (IR)

(pip install rpi-lgpio)

```
import requests
import time
import RPi.GPIO as GPIO

THINGSPEAK_WRITE_API_KEY = 'your_api_key_here'

THINGSPEAK_URL=f'https://api.thingspeak.com/update?api_key={THINGSPEAK_WRITE
_API_KEY}'

# GPIO setup
GPIO.cleanup()
GPIO.setmode(GPIO.BOARD)
GPIO.setup(8, GPIO.IN)

# Function to send data to ThingSpeak
def send_data_to_thingspeak(field_value):
    data = {'field1': field_value}
    try:
        response = requests.post(THINGSPEAK_URL, data=data)
        response.raise_for_status()  # Raise error for bad responses
        print("Data sent successfully to ThingSpeak.")
        print("Response:", response.text)
    except requests.exceptions.RequestException as e:
        print(f"Failed to send data to ThingSpeak: {e}")

# Main loop: read IR sensor and send data
try:
    while True:
        if GPIO.input(8) == 0:  # Object detected (assuming active LOW)
            print("IR sensor detected the object")
            send_data_to_thingspeak(1)  # Send 1
        else:  # No object detected
            print("IR sensor did not detect the object")
            send_data_to_thingspeak(0)  # Send 0

        time.sleep(15)  # Wait 15 seconds between sends to respect ThingSpeak rate limits

except KeyboardInterrupt:
    print("Program stopped by user.")
finally:
    GPIO.cleanup()
```

LED

```
import RPi.GPIO as GPIO
import time

# Use BCM pin numbering
GPIO.setmode(GPIO.BCM)

# Set up GPIO 17 as an output
led_pin = 17
GPIO.setup(led_pin, GPIO.OUT)

# Blink the LED
try:
    while True:
        GPIO.output(led_pin, GPIO.HIGH)  # LED ON
        time.sleep(1)                  # Wait 1 second
        GPIO.output(led_pin, GPIO.LOW)   # LED OFF
        time.sleep(1)                  # Wait 1 second
except KeyboardInterrupt:
    print("Program stopped")

# Clean up GPIO settings
GPIO.cleanup()
```

ULTRASONIC RPI

```
import requests
import time
import RPi.GPIO as GPIO

# ThingSpeak API setup
THINGSPEAK_WRITE_API_KEY = 'your_api_key_here'
```

```
THINGSPEAK_URL =
f'https://api.thingspeak.com/update?api_key={THINGSPEAK_WRITE_API_KEY}
'

# Ultrasonic sensor GPIO pins
TRIG = 23
ECHO = 24

# GPIO setup
GPIO.cleanup()
GPIO.setmode(GPIO.BCM)
GPIO.setup(TRIG, GPIO.OUT)
GPIO.setup(ECHO, GPIO.IN)

# Function to measure distance
def measure_distance():
    GPIO.output(TRIG, True)
    time.sleep(0.00001)
    GPIO.output(TRIG, False)

    while GPIO.input(ECHO) == 0:
        start_time = time.time()
    while GPIO.input(ECHO) == 1:
        end_time = time.time()

    duration = end_time - start_time
    distance = (duration * 34300) / 2
    return round(distance, 2)

# Function to send data to ThingSpeak
def send_data_to_thingspeak(field_value):
    data = {'field1': field_value}
    try:
        response = requests.post(THINGSPEAK_URL, data=data)
        response.raise_for_status()
        print("Data sent to ThingSpeak:", response.text)
    except requests.exceptions.RequestException as e:
        print("Failed to send data:", e)

# Main loop
try:
    while True:
        dist = measure_distance()
        print(f"Measured Distance: {dist} cm")
        send_data_to_thingspeak(dist)
        time.sleep(15)  # Respect ThingSpeak rate limits (min 15 seconds)
```

```
except KeyboardInterrupt:
    print("Program stopped by user.")
finally:
    GPIO.cleanup()
```

7. RPI and ESP32 Communication

Server

```
import socket

# Define server address and port
server_ip = '0.0.0.0'  # Listen on all interfaces
server_port = 9101

# Create a TCP socket
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Bind the socket to the IP and port
server_socket.bind((server_ip, server_port))

# Start listening for connections
server_socket.listen(1)
print(f"Listening for connections on {server_ip}:{server_port}...")

# Accept a connection
client_socket, addr = server_socket.accept()
print(f"Connection established with {addr}")

try:
    while True:
        # Receive data from ESP32
        data = client_socket.recv(1024).decode('utf-8')
        if data:
            print(f"Received from ESP32: {data}")

            # Send a response to ESP32
            response = input("Enter a message to send to ESP32: ")
            client_socket.send(response.encode('utf-8'))
```

```
except KeyboardInterrupt:
    print("\nServer stopped by user.")

finally:
    client_socket.close()
    server_socket.close()
    print("Sockets closed.")
```

client ESP 32

```
#include <WiFi.h>

// Your Wi-Fi credentials
const char* ssid = "Nord4";
const char* password = "nord1234";

// Raspberry Pi IP and port
const char* server_ip = "192.168.254.230"; // Replace with your actual Pi IP
const uint16_t server_port = 9101;

WiFiClient client;

void setup() {
  Serial.begin(115200);

  // Connect to Wi-Fi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi...");
  }
  Serial.println("Connected to WiFi");

  // Connect to Raspberry Pi TCP server
  if (client.connect(server_ip, server_port)) {
    Serial.println("Connected to Raspberry Pi server.");
  } else {
    Serial.println("Connection to Raspberry Pi failed.");
  }
}
```

```
void loop() {
  if (!client.connected()) {
    Serial.println("Disconnected. Reconnecting...");
    if (client.connect(server_ip, server_port)) {
      Serial.println("Reconnected to server.");
    } else {
      Serial.println("Reconnect failed.");
      delay(2000);
      return;
    }
  }

  // Send data to Raspberry Pi
  client.println("Hello from ESP32!");

  // Wait for a response from the Raspberry Pi
  if (client.available()) {
    String response = client.readStringUntil('\n');
    Serial.println("Received: " + response);
  }

  delay(2000);  // Wait before sending the next message
}
```