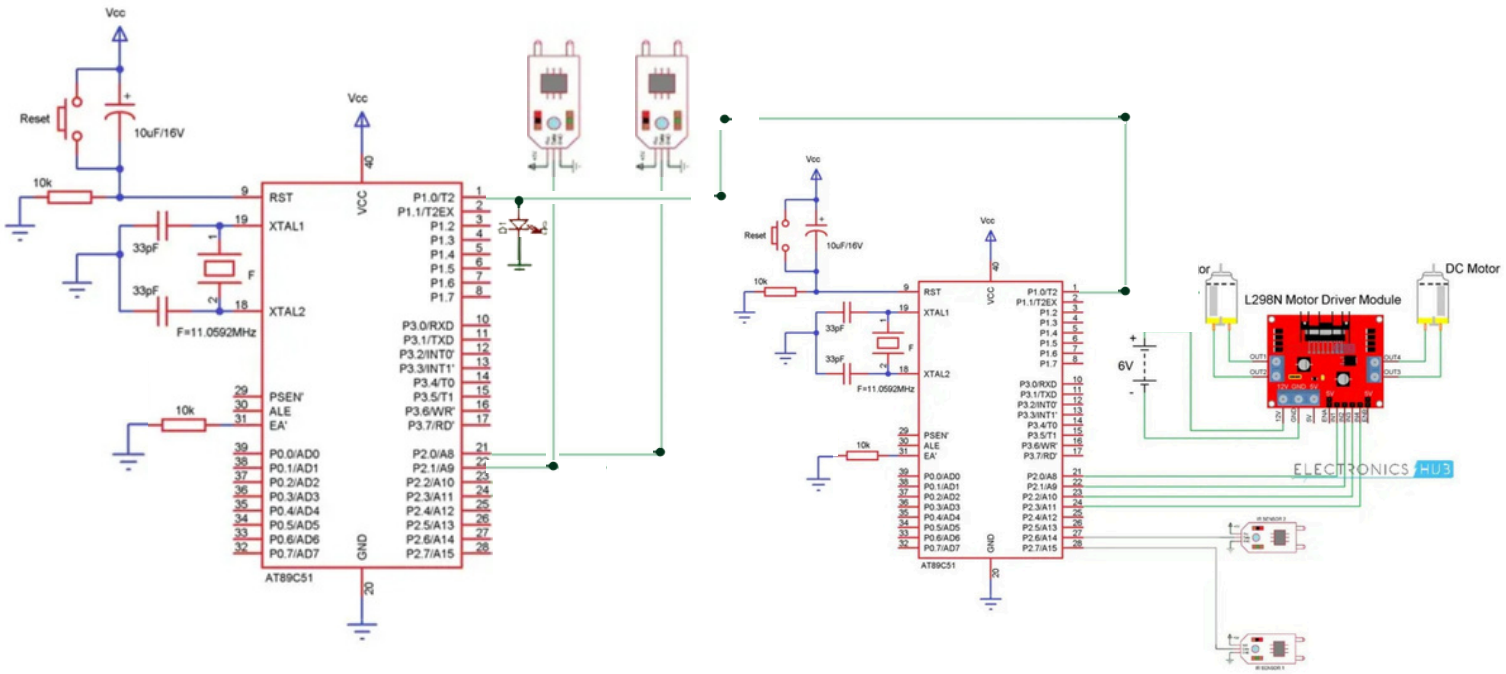
The Schematic Diagram of the circuit:



Components used in the circuit(for further clarification):

- 8051 Microcontroller x 2
- Development Board for 8051 Microcontroller (preferred)
- 10KΩ Resistors x 4
- 10µF Capacitors x 2
- 11.0592MHz Crystal x 2
- 33pF Capacitors x 4
- Push Button
- Motor driver Module (L298N)
- Robot Chassis with Motors
- IR Sensors x 4
- LED x 1

- **Some points that will elucidate your understanding of this line follower circuit:**

1- Why using a motor driver instead of directly connecting the motors to the microcontroller?

DC motors require a much higher current, typically in the range of hundreds of milliamps or more, and often operate at voltages higher than the microcontroller's operating voltage, such as 6V or 12V. Connecting motors directly to the AT89S52 can either damage the microcontroller or result in the motors failing to move.

2- Why choosing a specific value of 33pF for the crystal oscillator?

To ensure a stable and accurate clock signal for the microcontroller to prevent timing errors in instruction execution and communication. This guarantees reliable operation of timing-critical functions, such as reading IR sensors or controlling motors.

3- Why connecting the EA bar pin to the ground?

In the 8051 microcontroller, the EA (External Access) pin decides whether the microcontroller fetches program code from internal memory or external memory. Since we will ignore the internal ROM and fetch the code from external memory, we grounded the EA bar pin.

- **Continuation:**

When light reflects (e.g., from a white surface):
- The IR light emitted by the IR LED reflects back and is received by the photodiode.
- The voltage at the comparator input (receiver side) is high.
- The LM393 comparator outputs LOW (0).

-> Output = 0 (Active LOW state).

When light does NOT reflect (e.g., from a black surface):
- The IR light is absorbed (black absorbs IR) or not reflected back to the photodiode.
- The voltage at the comparator input is low.
- The LM393 comparator outputs HIGH (1).

-> Output = 1.

4- What type of comparator that when it receives a high input, it outputs low?

An inverting comparator is a type of comparator that has two inputs: a non-inverting input (+) and an inverting input (-). When the voltage at the inverting input (-) is higher than the voltage at the non-inverting input (+), the output of the comparator becomes LOW (0). On the other hand, when the voltage at the inverting input (-) is lower than the voltage at the non-inverting input (+), the output becomes HIGH (1). This behavior is referred to as active LOW, meaning that the output goes LOW (0) when a specific condition is met.

🚗 FIRST CODE: "Car" Microcontroller

```
#include <reg51.h>

// Define bit addresses using sbit
sbit MOT1    = P2^0;   // Motor 1
sbit MOT2    = P2^1;   // Motor 2
sbit MOT3    = P2^2;   // Motor 3
sbit MOT4    = P2^3;   // Motor 4
sbit S_LEFT  = P2^4;   // Left Sensor
sbit S_RIGHT = P2^5;   // Right Sensor
sbit RADAR   = P1^0;   // RADAR STATE

// Function prototypes
void forward(void);
void left(void);
void right(void);
void stop(void);

void main(void) {
    // Initialize ports with pull-ups
    S_LEFT  = 1;    // Enable pull-up for left sensor
    S_RIGHT = 1;    // Enable pull-up for right sensor
    RADAR   = 1;    // Enable pull-up for radar

    while(1) {       // Infinite loop
        if(RADAR) {  // If radar detects obstacle
            stop();
        }
        else {
            if(S_LEFT) {          // Left sensor activated
                if(S_RIGHT) {     // Both sensors activated
                    stop();
                }
                else {           // Only left sensor activated
                    left();
                }
            }
            else {
                if(S_RIGHT) {    // Only right sensor activated
                    right();
                }
                else {           // No sensors activated
                    forward();
                }
            }
        }
    }
}

// Motor control functions
void forward(void) {
    MOT1 = 0;
    MOT2 = 1;
    MOT3 = 1;
    MOT4 = 0;
}

void left(void) {
    MOT1 = 0;
    MOT2 = 1;
    MOT3 = 0;
    MOT4 = 0;
}

void right(void) {
    MOT1 = 0;
    MOT2 = 0;
```

🚗 FIRST CODE: "Car" Microcontroller

```
    MOT3 = 1;
    MOT4 = 0;
}

void stop(void) {
    MOT1 = 0;
    MOT2 = 0;
    MOT3 = 0;
    MOT4 = 0;
}
```

🛰️ SECOND CODE: "Radar" Microcontroller

```
#include <reg51.h>

// Define pins
sbit sensor1 = P2^0;  // IR Sensor 1 (active low)
sbit sensor2 = P2^1;  // IR Sensor 2 (active low)
sbit Stop_Signal  = P1^0;  // LED (active high)

void delay_1ms(void);
void delay_ms(unsigned int ms);

void main() {
    unsigned int time_count;

    // Setup
    Stop_Signal = 0;
    sensor1 = 1;
    sensor2 = 1;

    while (1) {
        // Wait for first sensor to be triggered
        if (sensor1 == 0) {
            time_count = 0;

            // Measure delay until sensor2 is triggered
            while (sensor2 == 1) {
                delay_1ms();
                time_count++;

                // Timeout safeguard (1 second max wait)
                if (time_count > 1000) break;
            }

            // If time between sensors is = 180 ms, light up LED
            if (time_count <= 180) {
                Stop_Signal = 1;
                delay_ms(500); // LED ON for 500 ms
                Stop_Signal = 0;
            }

            // Wait until car leaves both sensors
            while (sensor1 == 0 || sensor2 == 0);
        }
    }
}

// ~1 ms delay at 12 MHz
void delay_1ms(void) {
    unsigned int i;
    for (i = 0; i < 1275; i++);
}

// General delay function
void delay_ms(unsigned int ms) {
    unsigned int i;
    for (i = 0; i < ms; i++)
        delay_1ms();
}
```