**EXPERIMENT 3**

**A. Addition of two 8-bit data**

ORG 0000H ; Set the starting address to 0000H

MOV A,30H ; Move the value from memory location 30H into accumulator A

MOV B,31H ; Move the value from memory location 31H into register B

ADD A,B ; Add the contents of register B to the value in accumulator A. Result gets stored in A

MOV 32H,A ; Store the result of the addition (SUM) into memory location 32H

JNC **SKIP** ; If no carry occurs (CY=0), Jump to the label SKIP

MOV A,#01H ; If carry occurs (CY=1), load the immediate value 01H into accumulator A

SJMP **STOP** ; Unconditionally jump to the label STOP to terminate further processing

**SKIP:** MOV A,#00H ; At label SKIP, load the immediate value 00H into accumulator A (no carry case)

**STOP:** MOV 33H,A ; At label STOP, store the value of CARRY (either 00H or 01H) into memory location 33H

SJMP **$** ; Infinite loop to stop program execution

END ; End of the program

**EXPERIMENT 3**

**B. Subtraction of two 8-bit data**

ORG 0000H ; Set the starting address to 0000H

MOV A,40H ; Load the value from memory location 40H into accumulator A

MOV B,41H ; Load the value from memory location 41H into register B

CLR C ; Clear the carry flag (ensure no borrow occurs initially)

SUBB A,B ; Subtract the contents of register B from A

MOV 42H,A ; Store the result of the subtraction (DIFFERENCE) into memory location 42H

JNC **SKIP** ; If no borrow occurs (CY=0), jump to the label SKIP

MOV A,#01H ; If a borrow occurs (CY=1), load the immediate value 01H into accumulator A

SJMP **STOP** ; Unconditionally jump to the label STOP to terminate further processing

**SKIP:** MOV A,#00H ; At label SKIP, load the immediate value 00H into accumulator A (no borrow case)

**STOP:** MOV 43H,A ; At label STOP, store the value of BORROW (either 00H or 01H) into memory location 43H

SJMP **$** ; Infinite loop to stop program execution (halts at the STOP label)

END ; End of the program

**EXPERIMENT 3**

**C. Multiplication of two 8-bit data**

ORG 0000H ; Set the starting address to 0000H

MOV A,50H ; Load the value from memory location 50H into accumulator A

MOV B,51H ; Load the value from memory location 51H into register B

MUL AB ; Multiply the contents of A and B (A \* B) and stored in A (lower byte) and B (higher byte).

MOV 52H,A ; Store the lower byte of the multiplication result (from A) into memory location 52H

MOV 53H,B ; Store the higher byte of the multiplication result (from B) into memory location 53H

SJMP $ ; Infinite loop to stop program execution (halts at this line)

END ; End of the program

**EXPERIMENT 3**

**D. Multiplication of two 8-bit data**

ORG 0000H ; Set the starting address to 0000H

MOV A,60H ; Load the value from memory location 60H into accumulator A (dividend)

MOV B,61H ; Load the value from memory location 61H into register B (divisor)

DIV AB ; Perform division: divide the value in A by the value in B and quotient is stored in A, and the remainder is stored in B

MOV 62H,A ; Store the quotient (result of division) from A into memory location 62H

MOV 63H,B ; Store the remainder from B into memory location 63H

SJMP $ ; Infinite loop to stop program execution (halts here)

END ; End of the program

**EXPERIMENT 4**

**A. Interfacing of Switch and LED**

#include <AT89S52.h> // Include the header file for the AT89S52 microcontroller

#define SW P3\_2 // Define SW as the pin P3.2, which acts as the switch input

#define LED P2\_0 // Define LED as the pin P2.0, which controls the LED output

void main()

{

SW = 1; // Set the switch pin (SW) to high (inactive state in a pull-up configuration)

LED = 1; // Turn OFF the LED initially (assuming active-low LED configuration)

while(1) // Create an infinite loop to continuously check the switch state

{

if(SW == 0) // Check if the switch is pressed (SW is active-low)

{

LED = 0; // Turn ON the LED (active-low)

}

else // If the switch is not pressed

{

LED = 1; // Turn OFF the LED (active-low)

}

}

}

**EXPERIMENT 4**

**B. Interfacing of RGB LED**

#include <AT89S52.h> // Include the header file for the AT89S52 microcontroller

#define RLED P0\_0 // Define RLED as the pin P0.0, which controls the Red LED

#define GLED P0\_1 // Define GLED as the pin P0.1, which controls the Green LED

#define BLED P0\_2 // Define BLED as the pin P0.2, which controls the Blue LED

void delay(int c) // Function to introduce a delay

{

int i, j; // Loop control variables

for(i = 0; i < c; i++) // Outer loop to control the delay duration

{

for(j = 0; j < 1275; j++) // Inner loop to introduce a smaller time delay

{} // Empty loop body used only for creating a time delay

}

}

void main()

{

RLED = 0; GLED = 0; BLED = 0; // Turn OFF the Red LED, Green LED & Blue LED initially

while(1) // Infinite loop to cycle through LED states

{

RLED = 1; GLED = 0; BLED = 0; // Turn ON the Red LED and Ensure the Green LED & Blue LED is OFF

delay(100); // Keep the Red LED on for some time

RLED = 0; GLED = 1; BLED = 0; // Turn ON the Green LED and Ensure the Red LED & Blue LED is OFF

delay(100); // Keep the red LED on for some time

RLED = 0; GLED = 0; BLED = 1; // Turn ON the Blue LED and Ensure the Red LED & Green LED is OFF

delay(100); // Keep the red LED on for some time

}

}

**EXPERIMENT 4**

**C. Interfacing of Buzzer**

#include <AT89S52.h> // Include the header file for the AT89S52 microcontroller

#define BUZZER P0\_3 // Define BUZZER as the pin P0.3, which controls the buzzer output

void delay(int c) // Function for creating a delay

{

int i, j; // Declare loop variables

for(i = 0; i < c; i++) // Outer loop to control the overall delay duration

{

for(j = 0; j < 1275; j++) // Inner loop to create a smaller unit of delay

{ } // Empty loop body - used purely for time delay

}

}

void main()

{

BUZZER = 0; // Initialize the buzzer pin to low (buzzer off)

while(1) // Infinite loop to continuously toggle the buzzer

{

BUZZER = 1; // Turn on the buzzer

delay(100); // Call the delay function to keep the buzzer ON for some time

BUZZER = 0; // Turn off the buzzer

delay(100); // Call the delay function to keep the buzzer OFF for some time

}

}

**EXPERIMENT 5**

**Microcontroller “A” - Transmitter**

#include <AT89S52.h> // Include the header file for the AT89S52 microcontroller

#define SW P3\_2 // Define SW as the pin P3.2, which acts as an external switch input

// Define a data array containing hexadecimal values and a terminator ('$')

char data[] = {0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99, 0xAA, 0xBB, 0xCC, 0xDD, 0xEE, 0xFF, '$'};

char \*ptr; // Pointer to traverse through the data array

char count = 0; // Counter variable (unused in this code)

void delay(int c) // Function to create a delay

{

int i, j; // Loop control variables

for (i = 0; i < c; i++) // Outer loop for total delay duration

{

for (j = 0; j < 250; j++) // Inner loop to create a smaller unit of delay

{ } // Empty loop body to generate the delay

}

}

void main()

{

ptr = &data[0]; // Initialize the pointer to point to the start of the data array

SW = 1; // Initialize the switch (SW) in its idle state (high level)

SCON = 0x40; // Configure the Serial Control register for mode 1 (8-bit UART, variable baud rate)

TR1 = 0; // Stop timer 1 (ensure it is not running initially)

TMOD = 0x20; // Configure Timer 1 in mode 2 (8-bit auto-reload mode)

TH1 = 0xFD; // Set reload value for Timer 1 to generate baud rate of 9600

TL1 = 0xFD; // Load the same value into TL1 for initial operation

TR1 = 1; // Start Timer 1 to enable baud rate generation

IE = 0x90; // Enable interrupts: Enable serial interrupt and global interrupt

SBUF = 0x00; // Initialize the Serial Buffer register with 0

while (1); // Infinite loop to wait for interrupts

}

**Microcontroller “A” - Serial Interrupt Service Routine (ISR)**

void serial\_isr(void) \_\_interrupt(4) // Interrupt vector 4 is dedicated to serial communication

{

TI = 0; // Clear the Transmit Interrupt flag

while (SW == 1); // Wait until the switch (SW) is pressed (active low)

delay(1); // Debounce delay

while (SW == 0); // Wait until the switch (SW) is released (active high)

if (\*ptr != '$') // If the current character in the data array is not the terminator ('$')

{

SBUF = \*ptr; // Load the current character into the Serial Buffer register for transmission

\*ptr++; // Move the pointer to the next character in the array

}

else // If the terminator ('$') is reached

{

ptr = &data[0]; // Reset the pointer to the start of the data array

SBUF = 0x00; // Transmit a NULL character (end of transmission)

}

}

**Microcontroller “B” - Receiver**

#include <AT89S52.h> // Include the header file for the AT89S52 microcontroller

#define LED P0 // Define LED as Port P0, which is connected to the LEDs

void main()

{

LED = 0xFF; // Initialize all LEDs to off (assuming active-low configuration)

SCON = 0x50; // Configure the Serial Control register for mode 1 (8-bit UART, variable baud rate)

TR1 = 0; // Stop Timer 1 initially

TMOD = 0x20; // Configure Timer 1 in mode 2 (8-bit auto-reload mode)

TH1 = 0xFD; // Set the reload value for Timer 1 to generate a baud rate of 9600

TL1 = 0xFD; // Load the same value into TL1 for initial operation

TR1 = 1; // Start Timer 1 to enable baud rate generation

IE = 0x90; // Enable interrupts (serial interrupt and global interrupt)

while (1); // Infinite loop; program runs indefinitely, waiting for interrupts

}

**Microcontroller “B” - Serial Interrupt Service Routine (ISR)**

void serial\_isr(void) \_\_interrupt(4) // Interrupt vector 4 is dedicated to serial communication

{

RI = 0; // Clear the Receive Interrupt flag

LED = ~SBUF; // Invert the received byte (SBUF) and write it to 8-bit LED Array

}

**EXPERIMENT 6**

**A. Interfacing of Stepper Motor – FULL STEP SEQUENCE**

#include <AT89S52.h> // Include the header file for the AT89S52 microcontroller

#define STEPPER\_MOTOR P0 // Define STEPPER\_MOTOR as the port P0, which controls the stepper motor

void delay(int c) // Function for creating a delay

{

int i, j; // Declare loop variables

for(i = 0; i < c; i++) // Outer loop to control the total delay duration

{

for(j = 0; j < 250; j++) // Inner loop to create a smaller unit of delay

{ } // Empty loop body - used solely to create a time delay

}

}

void main()

{

char pattern = 0x88; // Initialize pattern for stepper motor. 0x88 represents the initial step position.

STEPPER\_MOTOR = 0x00; // Set all pins of STEPPER\_MOTOR to low (motor off initially)

while(1) // Infinite loop to rotate the stepper motor continuously

{

STEPPER\_MOTOR = pattern; // Send the current pattern to the stepper motor port to activate specific coils

delay(1); // Wait for a short time to allow the stepper motor to complete the step

pattern = (pattern >> 1) | (pattern << 7); // Rotate the bit pattern to activate successive coils

}

}

**EXPERIMENT 6**

**B. Interfacing of Stepper Motor – HALF STEP SEQUENCE**

#include <AT89S52.h> // Include the header file for the AT89S52 microcontroller

#define STEPPER\_MOTOR P0 // Define STEPPER\_MOTOR as port P0, which is connected to the stepper motor

void delay(int c) // Function for creating a delay

{

int i, j; // Declare loop variables

for(i = 0; i < c; i++) // Outer loop to control the total delay duration

{

for(j = 0; j < 250; j++) // Inner loop to create a smaller unit of delay

{ } // Empty loop body - used solely to create a time delay

}

}

void main()

{

char pattern[9] = {0x08, 0x0C, 0x04, 0x06, 0x02, 0x03, 0x01, 0x09}; // Define the stepping pattern

STEPPER\_MOTOR = 0x00; // Initialize the motor port to low (motor off initially)

while(1) // Infinite loop to drive the stepper motor continuously

{

char k; // Declare a loop control variable for iterating through the pattern

for(k = 0; k < 8; k++) // Loop through all 8 patterns to complete one rotation cycle

{

STEPPER\_MOTOR = pattern[k]; // Send the current pattern to the motor port to activate specific coils

delay(1); // Wait briefly to allow the motor to complete the step

}

}

}

**EXPERIMENT 7**

**Temperature Monitoring and Control using 8051 Microcontroller with ADC0809**

#include <AT89S52.h> // Header file for AT89S52 microcontroller

#define LCD\_BUS P2 // Define LCD data bus

#define RS P0\_5 // LCD Register Select pin

#define RW P0\_6 // LCD Read/Write pin

#define EN P0\_7 // LCD Enable pin

#include <LCD.h> // Header file for LCD functions

#define ADC\_BUS P3 // Define ADC data bus

#define ALE P1\_1 // Address Latch Enable pin for ADC0809

#define SOC P1\_2 // Start of Conversion pin for ADC0809

#define EOC P1\_3 // End of Conversion pin for ADC0809

#define OE P1\_4 // Output Enable pin for ADC0809

#define ADDRA P1\_5 // Address A pin for ADC0809

#define ADDRB P1\_6 // Address B pin for ADC0809

#define ADDRC P1\_7 // Address C pin for ADC0809

#include <ADC0809.h> // Header file for ADC functions

#define MM P0\_1 // Motor driver control pin (Cooling fan OFF)

#define MP P0\_0 // Motor driver control pin (Cooling fan ON)

void delay(char c) // Function to introduce delay

{

char i, j;

for(i = 0; i < c; i++) // Loop for required delay cycles

{

for(j = 0; j < 250; j++) // Inner loop adds delay effect

{}

}

}

void motor\_speed(char s) // Function to control motor speed using PWM logic

{

MP = 1; // Activate motor

delay(s); // Run motor for 's' amount of time

MP = 0; // Turn off motor

delay(100 - s); // Pause motor operation to adjust PWM duty cycle

}

void main()

{

char t, o; // Variables to store temperature digits

char val, temp, pwm;

lcd\_init(); // Initialize the LCD

lcd\_print("TEMPERATURE: "); // Print message on LCD

lcd\_data(0xDF); // Print degree symbol

lcd\_data('C'); // Print 'C' for Celsius

MM = 0; // Cooling fan off

MP = 0; // Motor stopped

while(1) // Infinite loop for continuous monitoring

{

val = read\_adc(0); // Read temperature sensor value from ADC channel 0

temp = (val \* 196) / 100; // Convert ADC value to temperature in Celsius

t = ((temp % 100) / 10) | 0x30; // Extract tens digit

o = (temp % 10) | 0x30; // Extract ones digit

lcd\_cmd(0x8C); // Move cursor position for temperature display

lcd\_data(t); // Display tens digit on LCD

lcd\_data(o); // Display ones digit on LCD

pwm = 20 + (temp - 25) \* 3; // Calculate PWM duty cycle for fan speed control

if(temp <= 25) // If temperature is below or equal to 25°C

{

MM = 0; // Turn off cooling fan

MP = 0; // Stop motor

}

else if(pwm >= 100) // If PWM exceeds 100%, run fan at full speed

{

MM = 0; // Ensure cooling fan motor state

MP = 1; // Run motor at full speed

}

else // If temperature is above 25°C but within controlled range

{

motor\_speed(pwm); // Adjust motor speed based on temperature

}

}

}