**Interrupt handling with a PIC Microcontroller**

STUDENT NAME: G.P.D. THAMARA

STUDENT NUMBER: EC/2021/005

**Source Code**

void interrupt() {

// Step 7: Interrupt service routine

if (INTCON.INTF == 1) { // Check external interrupt flag

// Step 8: Set PORTB values on interrupt

PORTB.RB1 = 1;

PORTB.RB2 = 0;

Delay\_ms(200); // 200 ms delay

// Step 9: Toggle values

PORTB.RB1 = 0;

PORTB.RB2 = 1;

Delay\_ms(200); // 200 ms delay

INTCON.INTF = 0; // Clear external interrupt flag

}

}

void main() {

// Step 2: Configuration

TRISB = 0b00000001; // RB0 as input, others as output

TRISA = 0b00000000; // All PORTA as output

CMCON = 0x07; // Disable comparators (disables RA5 special function)

OPTION\_REG = 0b10000000; // INT on rising edge, no prescaler

// Step 3: Enable interrupts

INTCON.GIE = 1; // Global interrupt enable

INTCON.PEIE = 1; // Peripheral interrupt enable

INTCON.INTE = 1; // Enable RB0 external interrupt

// Step 4: Infinite loop

while (1) {

// Step 5: Set PORT initial values

PORTB.RB2 = 0; // RB2 low

PORTA.RA0 = 1; // RA0 high

PORTA.RA1 = 0; // RA1 low

Delay\_ms(50); // 50 ms delay

// Step 6: Toggle

PORTA.RA0 = 0; // RA0 low

PORTA.RA1 = 1; // RA1 high

Delay\_ms(50); // 50 ms delay

INTCON.INTF = 0; // Clear interrupt flag (precaution)

}

}

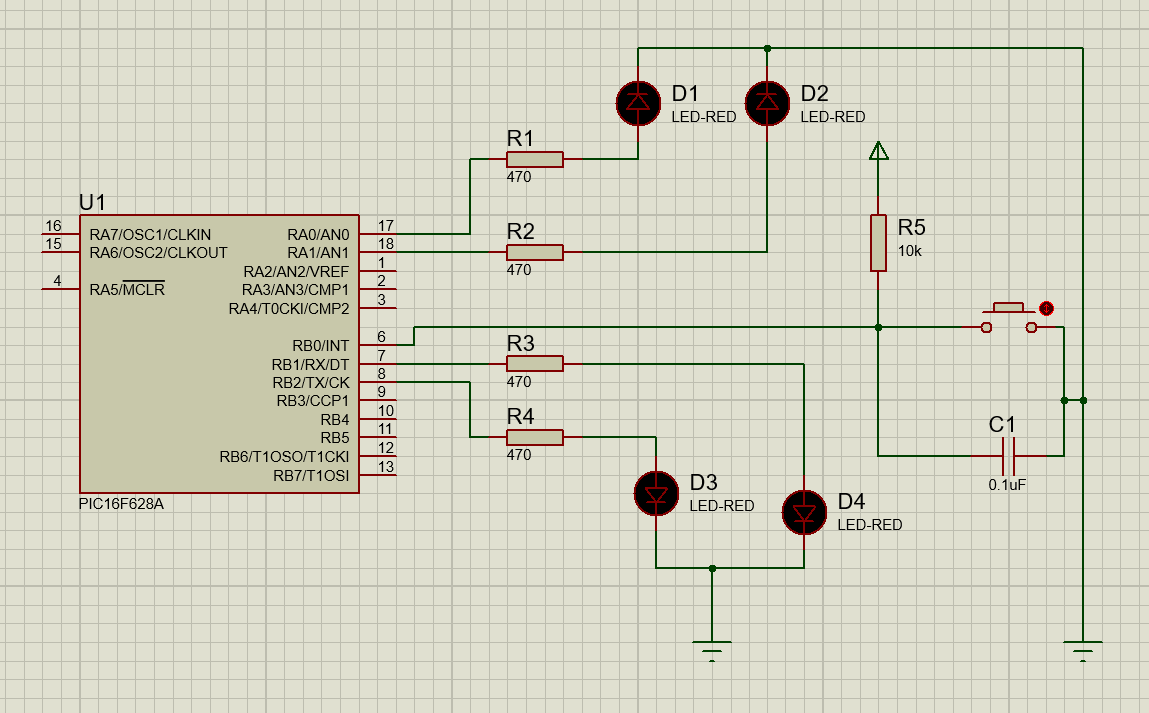
**A screenshot of a computer program

AI-generated content may be incorrect.**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**Circuit**

****

**Observations**

**A computer screen shot of a circuit

AI-generated content may be incorrect.**

**A computer screen shot of a circuit board

AI-generated content may be incorrect.**

**A computer screen shot of a circuit

AI-generated content may be incorrect.**

**Discussion**

This laboratory experiment focused on understanding and implementing interrupt handling mechanisms using the PIC16F628A microcontroller. Interrupts are critical features in embedded systems that allow the microcontroller to respond immediately to external or internal events without continuously polling for changes. The experiment emphasized the distinction between **internal interrupts** such as Timer0 overflow and USART communication and **external interrupts**, which originate from changes in external pins.

The primary objective of this experiment was to demonstrate the use of an **external interrupt** triggered through the **RB0/INT pin**. The system was configured so that when an external signal (e.g., button press) occurred on RB0, the microcontroller interrupted its main execution flow and entered a predefined **Interrupt Service Routine (ISR)** to handle the event. Key registers such as INTCON were used to enable and monitor the interrupt signals, while GIE and INTE bits ensured proper interrupt execution.

Through this setup, we gained hands-on experience in configuring and using interrupts, showcasing their efficiency in handling asynchronous events without affecting the main program’s continuous operation.