Assignment 3

a) Explain in detail the serial communication interruptions in PIC.
b) Produce 2 example codes to show how to use and program them.
c) Explain 2 applications where Serial communication interrupts can be used.

Font size = 10
Font type = Times New Roman
Line Spacing = 1.5

Q1

SERIAL COMMUNICATION INTERRUPTIONS IN PIC

Serial communication in PIC microcontrollers involves sending and receiving data one bit at a time over a single wire using the **USART** (**Universal Synchronous Asynchronous Receiver Transmitter**) module. Interrupts help the microcontroller handle serial communication more efficiently by responding to events automatically instead of constantly checking for them.

Interrupts are like alarms that notify the microcontroller when:

- Data is received (Receive Interrupt).
- Data can be sent (Transmit Interrupt).

When an interrupt occurs, the microcontroller stops what it is doing, runs a special code block called an **Interrupt Service Routine** (**ISR**) to handle the event, and then goes back to its normal tasks.

Types of Serial Communication Interrupts

- **Receive Interrupt (RCIF):**
  This interrupt is triggered when new data arrives in the RCREG (receive buffer).
  The microcontroller automatically jumps to the ISR to read the received data.
- **Transmit Interrupt (TXIF):**
  This interrupt is triggered when the TXREG (transmit buffer) is empty and ready to accept new data.
  The microcontroller can write new data to TXREG to send it.

Enabling Serial Interrupts

- Setting the GIE bit in the INTCON register to enable all interrupts.
- Setting the PEIE bit in the INTCON register to enable peripheral interrupts.
- Setting the RCIE bit in the PIE1 register to enable receive interrupts.
- Setting the TXIE bit in the PIE1 register to enable transmit interrupts.
- Creating a subroutine that runs when an interrupt occurs. Use BTFSS or BTFSC to check and handle specific interrupt flags (RCIF or TXIF).

During an interrupt, the microcontroller:

- Saves the current program counter.
- Jumps to the ISR (defined at address 0x08).
- Executes the interrupt-handling code.
- Returns to the main program using the RETFIE instruction.

This ensures the microcontroller can handle data communication without interrupting other ongoing tasks for long.

Q2

Example Codes

(a)

```
DATA EQU 0X20
   ORG 0x00
   GOTO MAIN

   ORG 0x08
   BTFSS PIR1, RCIF        ; Skip if RCIF is not set
   RETFIE                  ; Return if no receive interrupt
   GOTO ISR                ; Jump to ISR

MAIN:
   BSF STATUS, RP0         ; Bank 1
   MOVLW B'10010000'       ; RX enabled, asynchronous mode
   MOVWF RCSTA             ; Load into RCSTA
   MOVLW B'00100100'       ; TX enabled, 8-bit mode, BRGH = 1
   MOVWF TXSTA             ; Load into TXSTA
   MOVLW 25                ; Set baud rate (9600 for 4 MHz)
   MOVWF SPBRG             ; Load into SPBRG
   BCF STATUS, RP0         ; Bank 0
   BSF PIE1, RCIE          ; Enable Receive Interrupt
   BSF INTCON, PEIE        ; Enable Peripheral Interrupts
   BSF INTCON, GIE         ; Enable Global Interrupts

AGAIN:
   BRA AGAIN

ISR:
   MOVF RCREG, W           ; Move received data into WREG
   MOVWF DATA              ; Store data in a file register
   BCF PIR1, RCIF          ; Clear the Receive Interrupt flag
   RETFIE
```

(b)

```
   ORG 0x00
   GOTO MAIN

   ORG 0x08
   BTFSS PIR1, TXIF        ; Skip if TXIF is not set
   RETFIE                  ; Return if no transmit interrupt
   GOTO ISR                ; Jump to ISR

MAIN:
   BSF STATUS, RP0         ; Bank 1
   MOVLW B'10000000'       ; TX enabled, asynchronous mode
   MOVWF RCSTA             ; Load into RCSTA
   MOVLW B'00100100'       ; TX enabled, 8-bit mode, BRGH = 1
   MOVWF TXSTA             ; Load into TXSTA
   MOVLW 25                ; Set baud rate (9600 for 4 MHz)
   MOVWF SPBRG             ; Load into SPBRG
   BCF STATUS, RP0         ; Bank 0
   BSF PIE1, TXIE          ; Enable Transmit Interrupt
   BSF INTCON, PEIE        ; Enable Peripheral Interrupts
   BSF INTCON, GIE         ; Enable Global Interrupts

AGAIN:
   BRA AGAIN

ISR:
   MOVLW 'A'               ; Load data to be transmitted
   MOVWF TXREG             ; Write data to TXREG to transmit
   BCF PIR1, TXIF          ; Clear the Transmit Interrupt flag
   RETFIE
```

- RCREG: Holds the received data.
- TXREG: Holds the data to be transmitted.
- PIR1: Contains the RCIF flag (Receive Interrupt Flag) and TXIF flag (Transmit Interrupt Flag).
- PIE1: Enables the RCIE (Receive Interrupt Enable) and TXIE (Transmit Interrupt Enable).
- RX: Pin for receiving data. Enabled via RCSTA (MOVLW B'10010000').
- TX: Pin for transmitting data. Enabled via TXSTA (MOVLW B'00100100').
- RCSTA: Receive configuration and control. Configures RX.
- TXSTA: Transmit configuration and control. Configures TX (TXEN enabled, BRGH set high).
- BRGH: Controls baud rate speed. Set to 1 in TXSTA for high-speed communication.
- SPBRG: Register for setting baud rate. MOVLW 25 sets the SPBRG value to 25, which corresponds to a baud rate of 9600 when the system clock is 4 MHz.
- Baud Rate: Speed of communication (bits per second). Determined by SPBRG and the system clock frequency. $SPBRG = \frac{\text{Foscillator}}{16 \times BR} - 1$

For BR= 9600 at 4MHZ, $SPBRG = \frac{4 \times 10^6}{16 \times 9600} - 1 = 25$

Q3

Applications of Serial Communication Interrupts

Serial communication interrupts are widely used in real-world applications to efficiently handle data transmission and reception.

Communication b/w Microcontroller and PC

Imagine a microcontroller is connected to a PC via UART. The PC sends commands (e.g., "Turn ON the LED") to the microcontroller, and the microcontroller sends back responses (e.g., "LED is ON"). Here is how interrupts help:

- Receive Interrupt: When the PC sends a command, the microcontroller detects the incoming data through a receive interrupt. It processes the command and performs the required action.
- Transmit Interrupt: After executing the command, the microcontroller sends a response back to the PC using a transmit interrupt.

Benefits

- The microcontroller doesn’t waste time constantly checking for incoming data.
- It can handle multiple tasks while waiting for data from the PC.

Real-Life Use

- Home automation systems (e.g., turning devices ON/OFF via a PC or smartphone).
- Serial debugging tools for communicating between microcontrollers and computers.

Data Logging with Sensors

Imagine a microcontroller collects data from sensors (e.g., temperature or humidity) and sends it to an external storage device or another microcontroller for logging or analysis. Here is how interrupts help:

- Receive Interrupt: If the sensor sends data periodically (e.g., temperature readings every second), the microcontroller uses a receive interrupt to read the data when it arrives.
- Transmit Interrupt: The microcontroller sends the sensor data to a storage device or another system (e.g., a cloud server) using a transmit interrupt.

Benefits

- The microcontroller can focus on other tasks (e.g., controlling actuators or monitoring other sensors) instead of waiting for sensor data.
- It ensures reliable and timely data collection and transmission.

Real-Life Use

- Weather monitoring stations (collecting and transmitting weather data).
- Industrial automation (monitoring and logging machine parameters like speed or pressure).

Efficiency:

Interrupts allow the microcontroller to focus on other tasks and only react when data is available or ready.

Accuracy:

Interrupts ensure no data is missed, as they trigger immediately when data is received or the buffer is ready for transmission.

Flexibility:

Interrupts enable multitasking, allowing the microcontroller to handle serial communication alongside other critical functions (e.g., motor control, signal processing).