LAB5

Interrupts using msp430 microcontroller and interrupt based Knight rider

Engr.Shahzada Fahim Jan

Interrupt:

- An external event that interrupt the microcontroller that a device need its service
- Whenever a device needs its service, the device notify the microcontroller by sending it an interrupt signal
- Upon receiving The microcontroller interrupt whatever it is doing and service the device
- The program associated with interrupt is called ISR(Interrupt Service Routine)

Interrupt Service Routine:

- For every interrupt, there must be an interrupt service routine
- When an interrupt is invoked, the microcontroller runs the interrupt service routine.
- The **Global Interrupt Enable (GIE) flag** must be set to allow interrupts.

Steps in executing an interrupt

- Upon activation of interrupt, the microcontroller goes through the following steps
- 1.It finishes the instruction it is executing and save the address of the next instruction(PC) on the stack
- 2.It jumps to a fixed location in memory called Interrupt vector table that holds the address of the Interrupt Service Routine.
- 3.get the address of ISR from Interrupt vector table and jumps to it and starts to execute the ISR until it reaches the last instruction RETI
- 4.Upon executing RETI it return to the place where it was interrupted

1. Hardware Triggers

•Caused by **external hardware components**.

•**Sensitivity Levels:**

- **Edge Triggered:** Activated by a change in signal.
  - **Rising Edge:** Triggered when a signal transitions from **low to high (0 → 1)**.
  - **Falling Edge:** Triggered when a signal transitions from **high to low (1 → 0)**.
- **Level Triggered:** Activated when a signal stays at a specific logic level (**High or Low**).

2. Software Triggers

•Triggered by a **software event** within a program.

•Often used in debugging, testing, or simulating interrupts.

3. CPU Exceptions

•Caused by an **internal CPU state**, such as:

- Division by zero.
- Accessing invalid memory.
- Illegal instruction execution.

•These exceptions are handled by the CPU's **exception handling mechanism**.

1. Maskable Interrupts

- **Can be blocked (masked)** using a flag:
  - **Global Interrupt Flag (GIE)**: Controls whether interrupts are globally enabled or disabled.
  - **Local Flags in Peripheral Interfaces**: Individual peripherals have their own interrupt enable/disable flags.
- **Disabled upon RESET**
- **Automatically disabled upon entering an Interrupt Service Routine (ISR)**

2. Non-maskable Interrupts (NMI)

- **Cannot be masked (disabled)**, meaning they are always executed.
- **Reserved for system-critical events** such as:
  - Power failure detection.

```c
#include <msp430fr4133.h>
void main(void) {
    WDTCTL = WDTPW | WDTHOLD;
    PM5CTL0 &= ~LOCKLPM5;
    P1DIR |= BIT0;
    P1OUT &= ~BIT0;
    P1DIR &= ~BIT2;
    P1REN |= BIT2;
    P1OUT |= BIT2;
    P1IE  |= BIT2;
    P1IES |= BIT2;
    P1IFG &= ~BIT2;
    __bis_SR_register(GIE);     // Enable global interrupts
    while (1) {
            __no_operation();
    }
}
```

```
// ISR for Port 1
#pragma vector = PORT1_VECTOR
__interrupt void Port_1(void) {
    P1OUT ^= BIT0;
    P1IFG &= ~BIT2;
}
```

TASKS:

Write Msp430 program which toggle LED P1.0 when an interrupt occur at P1.2

Toggle LED p1.0 and P4.0 when an interrupt occur at P1.2

Detect short or long press of button if button is pressed for longer time the LED should remain ON if the button is pressed for a shorter time it should toggle

Write code for interrupt based knight rider