ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 351E
MICROCOMPUTER LABORATORY EXPERIMENT REPORT

EXPERIMENT NO	:	2
EXPERIMENT DATE	:	15.11.2024
LAB SESSION	:	FRIDAY - 14.30
GROUP NO	:	G9

GROUP MEMBERS:

150220723	:	Abdulsamet Ekinci
150210906	:	Emil Huseynov
150220902	:	Nahid Aliyev

FALL 2024

Contents

1 INTRODUCTION [10 points]

This experiment focused on general-purpose input/output (GPIO) programming using the MSP430G2553 microcontroller. Through three distinct tasks, we explored controlling LEDs with buttons, handling button debouncing, toggling states, and managing counters. This lab emphasized memory management, bitwise operations, and efficient control of GPIO ports, offering valuable experience in low-level assembly language programming.

2 MATERIALS AND METHODS [40 points]

This section outlines the hardware and software used, followed by detailed descriptions of the procedures for each task.

2.1 Hardware

- **MSP430G2553 Microcontroller:** A low-power microcontroller with 16-bit RISC architecture.
- **LEDs:** Connected to ports P1 and P2 for output visualization.
- **Buttons:** Configured as inputs for triggering and control logic.

2.2 Software

- **Code Composer Studio (CCS):** Used to write, debug, and flash assembly programs onto the MSP430.

2.3 Part 1

Objective: Turn on LED P1.4 when button P2.4 is pressed and hold the state indefinitely.

Procedure:

- Configured P1.4 as an output and P2.4 as an input:

```
mov.b #11111111b, &P1DIR    ; Set all bits of Port 1 as outputs
mov.b #00010000b, &P2DIR    ; Configure P2.4 as input
```

- Entered a loop to poll P2.4. When pressed, P1.4 was turned on:

```
loop:
    cmp.b #00010000b, &P2IN  ; Check P2.4 state
    jeq led_on               ; Jump if button pressed
    jmp loop                 ; Repeat
led_on:
    bis.b #00010000b, &P1OUT ; Turn on LED P1.4
```

- The program entered an infinite loop to maintain the state.

Challenges and Observations:

- Button press was detected successfully.
- No debouncing was necessary as the state remained stable.

2.4 Part 2

Objective: Toggle between LEDs P2.2 and P2.3 using button P1.5. Ensure no repeated toggling during a single press.

Procedure:

- Initialized P2.2 as ON and P2.3 as OFF:

```
mov.b #11111111b, &P2DIR  ; Set P2.2 and P2.3 as outputs
bis.b #00000100b, &P2OUT  ; Turn ON P2.2
bic.b #00001000b, &P2OUT  ; Turn OFF P2.3
```

- Polled P1.5 for button press transitions and toggled LEDs:

```
loop:
    cmp.b #00100000b, &P1IN  ; Check P1.5 state
    jeq toggle_leds          ; Jump if button pressed
    jmp loop                 ; Repeat

toggle_leds:
    xor.b #00001100b, &P2OUT ; Toggle P2.2 and P2.3
```

- Introduced a delay loop to prevent bouncing:

```
delay:
    mov.w #05000h, R15       ; Load delay count
delay_loop:
    dec.w R15
    jnz delay_loop
```

Challenges and Observations:

- Initial attempts showed bouncing issues, causing rapid toggling.
- Adding a delay loop resolved the problem, ensuring stable toggling.

2.5 Part 3

Objective: Count button presses on P2.1 using a 4-bit variable stored in memory. Display the count on Port 1 and reset at 16.

Procedure:

- Declared and initialized a memory variable for the counter:

```
count .data 0h                  ; Define and initialize count
```

- Incremented the counter on each valid button press:

```
loop:
    cmp.b #00000010b, &P2IN ; Check P2.1 state
    jeq increment_count     ; Jump if pressed
    jmp loop

increment_count:
    inc.b count             ; Increment counter
    cmp.b #10d, count       ; Check if count == 16
    jeq reset_count
    mov.b count, &P1OUT     ; Display on P1
    jmp loop

reset_count:
    mov.b #0h, count        ; Reset counter to 0
    mov.b count, &P1OUT
    jmp loop
```

- Used debouncing to ensure accurate counting.

Challenges and Observations:

- Initial counts were inconsistent due to bouncing.
- Debouncing resolved the issue, producing stable counts up to 16.

3 RESULTS [15 points]

- **Part 1:** LED P1.4 turned on upon pressing P2.4 and remained on indefinitely.
- **Part 2:** LEDs P2.2 and P2.3 toggled reliably with button P1.5, without repeated toggling during a single press.
- **Part 3:** Button press counts were displayed on Port 1 and reset correctly at 16.

4 DISCUSSION [25 points]

This experiment demonstrated practical GPIO programming and highlighted challenges such as debouncing. Key takeaways included:

- Understanding and handling button debouncing.
- Efficient use of memory and registers for managing counters.
- Leveraging assembly instructions for real-time hardware control.

5 CONCLUSION [10 points]

This experiment provided insights into GPIO operations, including LED control, toggling states, and managing counters. The MSP430 platform proved versatile for hands-on assembly programming.