ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 351E

MICROCOMPUTER LABORATORY EXPERIMENT REPORT

EXPERIMENT NO	:	5
EXPERIMENT DATE	:	13.12.2024
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 assembly programming for the MSP430 microcontroller, specifically implementing a 7-segment display control system and initializing interrupts. The objective was to learn low-level control of hardware through assembly, such as managing I/O ports for the 7-segment display and creating a simple timer-based counter system. Additionally, the experiment included implementing a button-controlled mechanism to toggle counting modes and manage the counter. This experiment aimed to enhance our understanding of hardware programming, particularly in embedded systems.

2 MATERIALS AND METHODS [40 points]

This section describes the hardware, software, and methods used in the experiment.

2.1 Hardware

- **MSP430G2553 Microcontroller:** A low-power microcontroller with a 16-bit RISC architecture, used to control the system and interact with hardware components.
- **7-Segment Display:** Connected to Port 1 to display numbers.
- **Push Buttons:** Configured on Port 2 to button connected t Pin 6 for changing mode (odd/even).

2.2 Software

- **Code Composer Studio (CCS):** Used for writing, debugging, and flashing assembly code onto the MSP430 microcontroller.

2.3 Part 1

Objective

The primary objective of Part 1 is to control a 7-segment display using the MSP430 microcontroller. The display sequentially cycles through numbers 0 to 9, and upon reaching 9, it resets to 0 and repeats the cycle. This is achieved using assembly language to manipulate registers, memory, and I/O ports.

System Design and Logic

To control the 7-segment display, the system requires precise bit manipulation to activate specific segments for each digit. Each digit from 0 to 9 corresponds to a unique binary pattern where each bit represents the state (ON/OFF) of a segment on the display.

7-Segment Mapping Logic: Each digit on the 7-segment display is represented by activating the following segments:

- 0: a, b, c, d, e, f
- 1: b, c
- 2: a, b, d, e, g
- 3: a, b, c, d, g
- 4: b, c, f, g
- 5: a, c, d, f, g
- 6: a, c, d, e, f, g
- 7: a, b, c
- 8: a, b, c, d, e, f, g (all segments on)
- 9: a, b, c, d, f, g

These segment states are stored in an array as binary values where each bit corresponds to the state of a segment (1 = ON, 0 = OFF). For example, the binary value for digit 0 is 000111111b, representing active segments (a, b, c, d, e, f) and inactive segment (g).

Technical Explanation

Data Section

```
array:
    .byte   000111111b, 000000110b, 001011011b, 001001111b
    .byte   001100110b, 001101101b, 001111101b, 000000111b
    .byte   001111111b, 001101111b
```

Explanation: This section defines the binary segment patterns for the digits 0 to 9. Each pattern is an 8-bit binary value where each bit corresponds to a segment on the 7-segment display. The array is used later in the main program loop to fetch the pattern for each digit.

Initialization

```
mov.w   #__STACK_END, SP        ; Initialize stack pointer
mov.w   #WDTPW|WDTHOLD, &WDTCTL  ; Stop the watchdog timer
mov.b   #0d, P2SEL               ; Configure Port 2 as GPIO
mov.w   #0xFF, &P1DIR            ; Set Port 1 as output
mov.w   #0x00, &P1SEL            ; Set Port 1 as GPIO
mov.w   #0x00, &P1OUT            ; Clear Port 1 output
mov.w   #0x00, R12               ; Initialize the counter to 0
```

Explanation:

- **Stack Pointer Initialization:** The SP is set to the end of the stack to ensure proper memory management.
- **Watchdog Timer:** The watchdog timer is stopped using the WDTPW—WDTHOLD combination to prevent unnecessary resets.
- **Port 2 Configuration:** Port 2 is set as GPIO to allow button control, but this part is not used in Part 1.
- **Port 1 Configuration:**
  - P1DIR sets Port 1 as an output to control the 7-segment display.
  - P1SEL sets Port 1 to GPIO mode instead of alternative functions.
  - P1OUT clears Port 1, turning off all segments of the 7-segment display at the start.
- **Counter (R12) Initialization:** The register R12 is used as a counter to track which digit to display (0 to 9).

Main Loop

```
main_loop:
    cmp.w   #10, R12             ; Check if the counter has reached 10
    jge     reset_counter        ; If true, reset the counter
    mov.b   array(R12), &P1OUT   ; Display the corresponding digit
    call    #Delay               ; Wait for 1 second
    add.w   #1, R12              ; Increment the counter
    jmp     main_loop            ; Repeat the process
```

Explanation:

- **Counter Check:** The counter (R12) is compared to 10. If it reaches 10, the program jumps to reset_counter to reset the counter.
- **Display Digit:** The current value of R12 is used as an index to access the array, which contains the binary patterns for digits 0 to 9. The binary value corresponding to the digit is stored in P1OUT, causing the 7-segment display to show that digit.
- **Delay:** The delay is called to hold the current digit for 1 second.
- **Counter Increment:** After displaying the current digit, R12 is incremented to display the next digit.
- **Loop:** The loop repeats to display the next digit.

Counter Reset

```
reset_counter:
    clr.w   R12             ; Clear the counter
    jmp     main_loop       ; Return to the main loop
```

Explanation: When the counter reaches 10, it is cleared (set to 0) using the clr.w instruction. The program then returns to main_loop to begin the cycle again from 0.

Delay Function

```
Delay:
    mov.w   #0Ah, R14       ; Outer loop count
L2:
    mov.w   #07A00h, R15    ; Inner loop count
L1:
    dec.w   R15             ; Decrement inner loop
    jnz     L1              ; Repeat until zero
    dec.w   R14             ; Decrement outer loop
    jnz     L2              ; Repeat outer loop
    ret
```

Explanation:

- **Outer Loop (R14):** Controls the number of iterations of the inner loop.
- **Inner Loop (R15):** Counts down from 07A00h to 0.
- **Nested Loops:** The combination of outer and inner loops creates a time delay, allowing the displayed digit to be visible long enough for human perception.
- **Return:** The function returns control to the main loop.

2.4 Part 2

Objective

The primary objective of Part 2 is to extend the functionality of Part 1 by introducing an interrupt-driven mode change. The program controls the 7-segment display and cycles through numbers 0 to 9. However, in Part 2, the display mode is toggled between even and odd cycles using an interrupt from a button connected to Port 2, Pin 6. When the button is pressed, the mode toggles between displaying even and odd numbers, and the display resets accordingly.

System Design and Logic

The system builds on the logic from Part 1, where the digits 0 to 9 are displayed on the 7-segment display. The key enhancement in Part 2 is the addition of an interrupt service routine (ISR) that toggles the display mode between even and odd cycles.

7-Segment Display Mode Logic:

- In the **odd mode**, the display cycles through odd numbers: 1, 3, 5, 7, and 9.
- In the **even mode**, the display cycles through even numbers: 0, 2, 4, 6, and 8.

Technical Explanation

Data section is as same as Part 1.

Initialization

```
mov.w   #__STACK_END, SP        ; Initialize stack pointer
mov.w   #WDTPW|WDTHOLD, &WDTCTL  ; Stop the watchdog timer
mov.b   #0d, P2SEL               ; Configure Port 2 as GPIO
mov.w   #0xFF, &P1DIR            ; Set Port 1 as output
mov.w   #0x00, &P1SEL            ; Set Port 1 as GPIO
mov.w   #0x00, &P1OUT            ; Clear Port 1 output
mov.w   #0x00, R12               ; Initialize the counter to 0
mov.w   #0x00, R4                ; Initialize the mode (even mode)
```

Explanation:

- **Stack Pointer Initialization:** The stack pointer is initialized to __STACK_END to define the memory location for the stack.
- **Watchdog Timer:** The watchdog timer is disabled to prevent unnecessary resets.
- **Port 2 Configuration:** Port 2 is configured as GPIO for the interrupt-driven button input. However, in this part, the button is used to toggle the mode.

- **Port 1 Configuration:** Port 1 is configured as output to control the 7-segment display, similar to Part 1.
- **Counter (R12) and Mode (R4) Initialization:** Register R12 is used as the counter for the digits, and register R4 is used to track the current display mode (even or odd).

Main Loop

```
main_loop:
    cmp.w   #10, R12             ; Check if the counter has reached 10
    jge     reset_counter        ; If true, reset the counter
    cmp.w   #0, R4               ; Check the mode (even or odd)
    jeq     even_mode            ; Jump to even_mode if mode is even
odd_mode:
    mov.b   array(R12), &P1OUT   ; Display the corresponding odd number
    call    #Delay               ; Wait for 1 second
    add.w   #2, R12              ; Increment counter by 2 (odd mode)
    jmp     main_loop            ; Repeat the process
even_mode:
    mov.b   array(R12), &P1OUT   ; Display the corresponding even number
    call    #Delay               ; Wait for 1 second
    add.w   #2, R12              ; Increment counter by 2 (even mode)
    jmp     main_loop            ; Repeat the process
```

Explanation:

- **Counter Check:** The counter (R12) is compared to 10. If it reaches 10, the program jumps to `reset_counter` to reset the counter.
- **Mode Check:** The program checks the current mode (stored in R4). If the mode is even (R4 = 0), the program enters `even_mode`; otherwise, it enters `odd_mode`.
- **Display Digit:** In each mode, the corresponding binary pattern for the number (odd or even) is fetched from the `array` and displayed on the 7-segment display.
- **Delay and Counter Update:** The delay is called to hold the current digit for 1 second, and the counter is incremented by 2 to cycle through the odd or even digits.
- **Loop:** The loop continues, displaying the next digit in the selected mode.

Counter Reset

```
reset_counter:
    clr.w   R12             ; Clear the counter
    jmp     main_loop       ; Return to the main loop
```

Explanation:

When the counter reaches 10, it is cleared (set to 0) using the `clr.w` instruction. The program then returns to `main_loop` to begin the cycle again from 0.

Interrupt Service Routine (ISR)

```
P2_ISR:
    dint                        ; Disable interrupts
    mov.w   R4, R5              ; Copy the current mode (even/odd)
    xor.w   #1, R5              ; Toggle the mode (even/odd)
    mov.w   R5, R4              ; Store the new mode
    bis.w   #0x40, &P2IFG       ; Clear the interrupt flag for P2.6
    eint                        ; Enable interrupts again
    reti                        ; Return from ISR
```

Explanation:

- **Disable Interrupts:** Interrupts are disabled using `dint` to prevent re-entry into the ISR while it is being executed.
- **Toggle Mode:** The current mode is copied to R5, and an XOR operation with 1 toggles the mode between even (0) and odd (1).
- **Clear Interrupt Flag:** The interrupt flag for Port 2, Pin 6 is cleared by writing to `P2IFG`.
- **Enable Interrupts:** Interrupts are re-enabled using `eint`, and the ISR ends with `reti`, returning to the main program.

Interrupt Vector Setup

```
__interrupt_VECTOR__:
    .word   P2_ISR              ; Set Port 2 interrupt vector to ISR
```

Explanation:

The interrupt vector table is set to jump to the `P2_ISR` when an interrupt occurs on Port 2, Pin 6 (button press).

Delay Function

```
Delay:
    mov.w   #0Ah, R14       ; Outer loop count
L2:
    mov.w   #07A00h, R15    ; Inner loop count
L1:
    dec.w   R15             ; Decrement inner loop
    jnz     L1              ; Repeat until zero
    dec.w   R14             ; Decrement outer loop
    jnz     L2              ; Repeat outer loop
    ret
```

Explanation:

- **Outer Loop (R14):** Controls the number of iterations of the inner loop.
- **Inner Loop (R15):** Counts down from 07A00h to 0.
- **Nested Loops:** The combination of outer and inner loops creates a time delay, allowing the displayed digit to be visible long enough for human perception.
- **Return:** The function returns control to the main loop.

3 RESULTS AND DISCUSSION [30 points]

The experiment successfully implemented both the 7-segment display and initializing interrupts. The 7-segment display correctly showed numbers from 0 to 9, and initializing interrupts between even and odd counting modes using button presses. The counting logic worked correctly, and the button interaction was reliable once debounced.

However, some challenges were encountered:

- Debugging the counter reset mechanism and ensuring that the counter did not exceed 9 in even and odd modes was non-trivial.

4 CONCLUSION [20 points]

This experiment provided hands-on experience with low-level programming on the MSP430 microcontroller. The successful implementation of both the 7-segment display control and initializing interrupts , along with the interaction between button presses and counting modes, deepened our understanding of hardware-level programming, interrupt handling, and GPIO configuration. Although there were challenges, particularly in managing button presses and counter logic, the experiment was a valuable learning experience in embedded system design.