ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 351E
MICROCOMPUTER LABORATORY EXPERIMENT REPORT

EXPERIMENT NO	:	1
EXPERIMENT DATE	:	08.11.2024
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]

In this experiment, we explored basic assembly coding techniques using the MSP430 microcontroller and focused on controlling LEDs connected to ports P1 and P2. The primary objective was to understand assembly-level programming by working with registers, bitwise operations, and loop constructs on the MSP430. We initially ran a provided code to observe LED patterns and then modified it to achieve a specified sequential LED pattern. This experiment helped us familiarize ourselves with the assembly language syntax and the MSP430's architecture, particularly the use of registers and output control through bitwise operations.

2 MATERIALS AND METHODS [40 points]

This section details the hardware setup, software environment, and code structure used to accomplish the experiment objectives.

- **Hardware**
  - **MSP430G2553 Microcontroller**: A low-power, cost-effective microcontroller from Texas Instruments, featuring a 16-bit RISC CPU and multiple general-purpose I/O ports. Ports P1 and P2 on this microcontroller were used to control LEDs.
  - **LED Setup**: LEDs were connected to the output ports P1 and P2, with each bit representing an individual LED's state (on or off).
- **Software**
  - **Code Composer Studio (CCS)**: Texas Instruments' integrated development environment (IDE) was used to write, compile, and debug the assembly code. We created an "Empty Assembly-only Project" in CCS and selected "MSP430G2553" as the target microcontroller.

2.1 Part 1

The code initializes the direction and output registers for ports P1 and P2, enabling all bits as outputs and initially turning off all LEDs. R6 and R7 are loaded with specific bit patterns to define LED states in each loop cycle, while R8 acts as a loop counter. Two primary loops, `Mainloop1` and `Mainloop2`, control the lighting pattern on the LEDs, with `Mainloop2` introducing a sequential LED activation effect.

The following code was used in **Part 1** to control LED patterns on the MSP430. This initial code setup creates a moving LED pattern by rotating the bits in R6 and R7, thereby shifting the LED lights. The code includes a delay loop to slow down the LED transitions for visible effect.

Listing 1: Assembly Code for Part 1

```
; Setup direction registers for Port 1 and Port 2
SetupP1
    mov.b #11111111b, &P1DIR        ; Set all bits of P1 direction regis
    mov.b #11111111b, &P2DIR        ; Set all bits of P2 direction regis
    mov.b #00000000b, &P1OUT        ; Set all bits of P1 output register
    mov.b #00000000b, &P2OUT        ; Set all bits of P2 output register

    mov.b #00001000b, R6            ; Load initial LED pattern for P1 in
    mov.b #00010000b, R7            ; Load initial LED pattern for P2 in
    mov.b #0d, R8                   ; Initialize counter R8 to 0

Mainloop1
    bis.b R6, &P1OUT                ; Bitwise OR of R6 with P1OUT (turn
    bis.b R7, &P2OUT                ; Bitwise OR of R7 with P2OUT (turn

    inc R8                          ; Increment counter R8
    rra R6                          ; Rotate R6 right (change LED patter
    rla R7                          ; Rotate R7 left (change LED pattern

    mov.w #00500000, R15            ; Load a large value into R15 for de
L1
    dec.w R15                       ; Decrement R15 (delay loop)
    jnz L1                          ; Jump back to L1 if R15 is not zero

    cmp #4d, R8                     ; Compare R8 with 4 (decimal)
    jeq SetupP1                     ; If R8 == 4, restart from SetupP1 t
    jmp Mainloop1                   ; Repeat Mainloop1
```

This code sets up an LED pattern on both Port 1 and Port 2, where the LEDs are toggled in a rotating sequence. The code utilizes:

- **Bitwise OR Operations** (`bis.b`) to control the individual bits of the LED outputs.

- **Rotate Right (`rra`) and Rotate Left (`rla`) Operations** to shift the active LED in each loop cycle.
- **Delay Loop (`L1`)** to make the LED transitions visible.

The result of this setup is a rotating LED pattern that resets after four iterations.

2.2 Part 2

In Part 2, we modified the assembly code from Part 1 to create a new LED sequence on the MSP430 microcontroller. The objective was to achieve a sequential activation of LEDs across both Port 1 and Port 2, creating an alternating pattern as shown in the figure. This section describes key code modifications and logic for each part.

SetupP2 Section:

- `mov.b #11111111b, &P1DIR` and `mov.b #11111111b, &P2DIR`: Sets P1 and P2 as output ports.
- `mov.b #00000000b, &P1OUT` and `mov.b #00000000b, &P2OUT`: Clears outputs on both ports (LEDs off).
- `mov.b #10000000b, R6`: Loads initial LED pattern into `R6`.
- `mov.b #0d, R8`: Initializes loop counter `R8`.

Mainloop1 (P1 LED Control):

- `mov.b #00000000b, &P2OUT`: Clears P2 to ensure only P1 LEDs are active.
- `bis.b R6, &P1OUT`: Activates specific LEDs on P1 according to `R6`.
- `rra R6`: Rotates `R6` right, changing the LED pattern for the next cycle.
- `mov.w #00500000, R15`: Sets delay for P1 LEDs.
- `jmp Mainloop2`: Proceeds to delay loop.

Mainloop2 (P1 Delay):

- Delay loop using `R15` to keep P1 LEDs on for a visible period.
- `mov.b #00000000b, &P1OUT`: Clears P1 after delay.

Delay (P2 Delay):

- Additional delay to keep LEDs off briefly before P2 activation.

Mainloop3 (P2 LED Control):

- `bis.b R6, &P2OUT`: Activates specific LEDs on P2 according to `R6`.
- `rra R6`: Rotates `R6` right, changing the LED pattern for the next cycle.
- `mov.w #00500000, R15`: Sets delay for P2 LEDs.
- `jmp L1`: Proceeds to delay loop.

L1 (P2 Delay Loop and Sequence Reset):

- Delay loop for P2 LEDs.
- `cmp #8d, R8`: Checks if the loop has completed 8 cycles.
- `jeq SetupP2`: Resets the sequence if 8 cycles are complete.

Final code looks like this:

Listing 2: Assembly Code for Part 2

```
; Setup direction registers for Port 1 and Port 2
SetupP2
    mov.b   #11111111b, &P1DIR        ; Set all bits of P1 direction regi
    mov.b   #11111111b, &P2DIR        ; Set all bits of P2 direction regi
    mov.b   #00000000b, &P1OUT        ; Set all bits of P1 output registe
    mov.b   #00000000b, &P2OUT        ; Set all bits of P2 output registe

    mov.b   #10000000b, R6            ; Load initial LED pattern for P1 i
    mov.b   #0d, R8                   ; Initialize counter R8 to 0
    jmp Mainloop1                     ; Jump to Mainloop1

Mainloop1
    mov.b   #00000000b, &P2OUT        ; Set P2 output to zero
    bis.b   R6, &P1OUT                ; Bitwise OR of R6 with P1OUT (turn
    inc     R8                        ; Increment counter R8
    rra     R6                        ; Rotate R6 right (change LED patte

    mov.w   #00500000, R15            ; Load a large value into R15 for d
    jmp Mainloop2                     ; Jump to Mainloop2 for delay loop

Mainloop2
```

```
    dec.w   R15                     ; Decrement R15 (delay loop)
    jnz     Mainloop2               ; Jump back to Mainloop2 if R15 is

    mov.b   #00000000b, &P1OUT      ; Set P1 output to zero

    mov.w   #00500000, R15          ; Load a large value into R15 for d
    jmp delay                       ; Jump to delay loop for Port 2

delay
    dec.w   R15                     ; Decrement R15 (delay loop for P2
    jnz     delay                   ; Jump back to delay if R15 is not

Mainloop3
    bis.b   R6, &P2OUT              ; Bitwise OR of R6 with P2OUT (turn
    rra     R6                      ; Rotate R6 right (change LED patte
    inc     R8                      ; Increment counter R8
    mov.w   #00500000, R15          ; Load a large value into R15 for d
    jmp L1                          ; Jump to L1 for delay loop

L1
    dec.w   R15                     ; Decrement R15 (delay loop)
    jnz     L1                      ; Jump back to L1 if R15 is not zer

    cmp     #8d, R8                 ; Compare R8 with 8 (decimal)
    jeq     SetupP2                 ; If R8 == 8, restart from SetupP2

    jmp     Mainloop1               ; Repeat Mainloop1
```

2.3 Part 3

In Part 3, we modified the code to create a new LED pattern and added functionality to stop the sequence using a button on Port 2. The LED sequence now shifts downward in a moving pattern as shown in the figure. Additionally, the button on Port 2 is configured to stop the LED sequence when pressed.

- **SetupP3 Section:**
  - `mov.b #11111111b, &P1DIR`: Sets P1 as output.

  - `mov.b #11111100b, &P2DIR`: Sets P2.0 and P2.1 as inputs for button functionality.
  - `mov.b #00000000b, &P1OUT`: Clears outputs on P1 (LEDs off).
  - `mov.b #10000000b, R6`: Loads initial LED pattern into `R6`.
  - `mov.b #0d, R8`: Initializes loop counter `R8`.

- **Mainloop1 (LED Control):**
  - `bis.b R6, &P1OUT`: Activates specific LEDs on P1 according to `R6`.
  - `inc R8`: Increments the loop counter `R8`.
  - `rra R6`: Rotates `R6` right, changing the LED pattern.
  - `mov.w #00500000, R15`: Sets delay for LED transitions.
  - `jmp L1`: Proceeds to delay loop.

- **Stop (Check Button Input):**
  - `cmp #2d, &P2IN`: Checks the continue bit; if on, it allows the loop to proceed.
  - `jeq L1`: If continue bit is on, jumps to `L1` to proceed with delay.
  - `jmp Stop`: Otherwise, it remains in the stop loop.

- **L1 (Delay Loop):**
  - `dec.w R15`: Decrements `R15` to create a delay loop.
  - `jnz L1`: Loops back to `L1` if `R15` is not zero, maintaining the delay.
  - `cmp #1d, &P2IN`: Checks if button on P2.0 is pressed.
  - `jeq Stop`: If button is pressed, jumps to the `Stop` loop to halt the system.

- **Loop Control and Reset:**
  - `cmp #8d, R8`: Compares `R8` to 8 (decimal) to check if the sequence has completed.
  - `jeq SetupP3`: If `R8` equals 8, it resets by jumping to `SetupP3`.
  - `jmp Mainloop1`: If `R8` is not 8, it continues to `Mainloop1` to repeat the sequence.

Listing 3: Assembly Code for Part 3

```
; Setup direction registers for Port 1 and Port 2
SetupP3
    mov.b #11111111b, &P1DIR        ; Set P1 to output
    mov.b #11111100b, &P2DIR        ; Set P2.0 and P2.1 as input
    mov.b #00000000b, &P1OUT        ; Clear P1
    mov.b #10000000b, R6            ; Load initial LED pattern
    mov.b #0d, R8                   ; Initialize counter
    jmp Mainloop1

Mainloop1
    bis.b R6, &P1OUT                ; Activate P1 LEDs
    inc R8                          ; Increment counter
    rra R6                          ; Rotate R6
    mov.w #00500000, R15            ; Delay for LED transition
    jmp L1

Stop
    cmp #2d, &P2IN                  ; Check continue bit on P2
    jeq L1                          ; If continue bit is on, proceed
    jmp Stop                        ; Else, stay in stop loop

L1
    dec.w R15                       ; Delay loop
    jnz L1
    cmp #1d, &P2IN                  ; Check if button is pressed
    jeq Stop                        ; If button pressed, jump to stop

    cmp #8d, R8                     ; Check loop completion
    jeq SetupP3                     ; If complete, reset sequence
    jmp Mainloop1                   ; Otherwise, continue loop
```

3 RESULTS [15 points]

In this section, we summarize the results obtained during the experiment. The final LED patterns on the MSP430 microcontroller matched the expected outputs as specified in each part:

- **Part 1:** The LEDs on Port 1 and Port 2 were observed to toggle in a rotating sequence, with visible delays that allowed us to clearly follow the LED pattern.
- **Part 2:** The modified code successfully generated a sequential LED pattern, with alternating activations between Port 1 and Port 2, creating a smooth moving effect

across both ports.

- **Part 3:** The LED pattern shifted downward in sequence as per the requirements. Additionally, the button on Port 2 was able to stop the sequence effectively when pressed, adding interactivity to the system.

4 DISCUSSION [25 points]

In this experiment, we practiced low-level assembly programming on the MSP430 microcontroller, focusing on controlling hardware output and adding interactivity through button inputs. The main challenges encountered included:

- **Understanding Bitwise Operations:** Working with bitwise operations such as OR and rotate instructions required precision to achieve the desired LED patterns. Gaining familiarity with these operations helped us understand how to manipulate specific bits to control individual LEDs.
- **Implementing Delays:** Creating visible delays using busy-wait loops allowed us to slow down the transitions, making the LED sequences observable. The selection of appropriate delay values was crucial to balancing visibility and execution time.
- **Button Control Logic:** Adding button control to stop the LED sequence required configuring input pins correctly and implementing conditional branching based on button states. This part reinforced our understanding of input handling on the MSP430.

Overall, this experiment improved our understanding of assembly programming concepts and hardware control on the MSP430 microcontroller.

5 CONCLUSION [10 points]

This experiment provided valuable experience with assembly-level programming on the MSP430, specifically in controlling LED patterns and incorporating input from external buttons. Key learnings included:

- Using bitwise operations to manipulate individual bits for hardware control.
- Implementing delay loops to control the speed of visual output.
- Setting up and handling input conditions to add interactivity to the system.

We encountered some difficulties initially in configuring the button for stopping the LED sequence, but troubleshooting this issue enhanced our problem-solving skills. This experiment deepened our understanding of assembly language and gave us practical insights into low-level programming on microcontrollers.