ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 351E

MICROCOMPUTER LABORATORY EXPERIMENT REPORT

EXPERIMENT NO	:	1
EXPERIMENT DATE	:	06.11.2024
LAB SESSION	:	WEDNESDAY - 12.30
GROUP NO	:	G8

GROUP MEMBERS:

150200009 : Vedat Akgöz

150200097 : Mustafa Can Çalışkan

150200016 : Yusuf Şahin

SPRING 2024

Contents

1 INTRODUCTION

The objective of this experiment is to work with assembly language and gain knowledge about the MSP430G2553 microcontroller. The experiment involved using Texas Instruments' Code Composer Studio IDE and the MSP430 Education Board to write code and observe its output.

2 MATERIALS AND METHODS

2.1 Part 1

In this part, we developed a module that sequentially illuminates LEDs. The sequence starts from the middle of one side and extends outward in the opposite direction. Once all LEDs on one side are lit, the process moves to the other side, lighting the LEDs in reverse order. The system resets when half of the P1 LEDs are turned on. The implementation steps are as follows:

Setup

- Configure P1 and P2 as output by setting all their bits to 1.
- Initialize both ports by setting all their bits to 0.
- Set register `R6` to `00001000b` as the initial step for P1.
- Set register `R7` to `00010000b` as the initial step for P2.
- Initialize register `R8` to `0d` to serve as the step counter.

Loop1Start

- Use bitwise OR (with all 1's dominant) to combine the current state of P2 with a left rotation of `R7`. This keeps the currently lit LEDs and turns on the next one.
- Increment `R8` and left rotate `R7` to prepare for the next LED activation in the next bitwise OR operation.
- Introduce a delay by assigning a large value to the Program Counter (`R15`).

Wait1

- Use the Program Counter (`R15`) to implement a delay.
- After the delay, check if the value of `R8` is 4.
  - If true, jump to `Loop2Start` to start lighting LEDs on the other side.
  - Otherwise, return to `Loop1Start` to continue lighting LEDs on the current side.

Loop2Start

- Once P2 (one side) is completed, reset its value.
- Use bitwise OR (with all 1's dominant) to combine the current state of P1 with a right rotation of `R6`. This keeps the currently lit LEDs and turns on the next one.
- Decrement `R8` and right rotate `R6` to prepare for the next LED activation in the next bitwise OR operation.
- Introduce a delay by assigning a large value to the Program Counter (`R15`).

Wait2

- Use the Program Counter (`R15`) to implement a delay.
- After the delay, check if the value of `R8` is 0.
  - If true, jump to `Setup` to light LEDs on the other side.
  - Otherwise, continue with `Loop2Start` to light the LEDs on the current side.

You can find code below.

```
Setup
    mov.b #11111111b,&P1DIR
    mov.b #11111111b,&P2DIR
    mov.b #00000000b,&P1OUT
    mov.b #00000000b,&P2OUT
    mov.b #00001000b , R6
    mov.b #00010000b , R7
    mov.b #0d , R8
Loop1Start
    bis.b R7 , &P2OUT
```

```
    inc R8
    rla R7
    mov.w #00500000 , R15
Wait1
    dec.w R15
    jnz Wait1
    cmp #4d , R8
    jeq Loop2Start
    jmp Loop1Start
Loop2Start
    mov.b #00000000b,&P2OUT
    bis.b R6 , &P1OUT
    dec.w R8
    rra R6
    mov.w #00500000 , R15
Wait2
    dec.w R15
    jnz Wait2
    cmp #0d , R8
    jeq Setup
    jmp Loop2Start
```

2.2 Part 2

In this section, we developed a module where LEDs light up sequentially, starting from the bottom right and switching sides with each step. The sequence resets after the top-left LED is illuminated. The implementation details are as follows:

Setup

- Configure all pins of port P1 as outputs (`P1DIR`) and set their values to 0.
- Configure all pins of port P2 as outputs (`P2DIR`) and set their values to 0.
- Initialize the first bit of register `R6` with the value 1, representing the initial step for P1.
- Initialize the first bit of register `R7` with the value 1, representing the initial step for P2.

- Set register `R8` to 0 to serve as a general loop counter.
- Set register `R9` to 0 to control when the side changes.

LoopStart

- Compare the value in register `R9` with 0.
  - If equal, jump to the `GoLeft` label, indicating the LEDs on the left side will be activated.
- If not, assign the value of `R7` to the output of port P2 and clear the output of port P1.
- Reset `R9` to 0 since this block is only executed when `R9` is non-zero.
- Jump to the `GoRight` label to skip the `GoLeft` label.

GoLeft

- Assign the value in register `R6` to the output of port P1 and clear the output of port P2.
- Light the LED corresponding to `R6`.
- Increment the value of `R9`.

GoRight

- Increment the value of `R8` to advance the loop cycle.
- Perform a left rotation on both `R6` and `R7`, shifting the lit LED positions upwards.
- Set register `R15` to 50,000 to introduce a delay.
- Decrement `R15` until it reaches 0.
- Compare the value in `R8` with 8.
  - If equal, jump to the `Setup` label to restart the LED pattern.
  - If not, return to `LoopStart` to continue the loop.

You can find code below.

```
Setup
    mov.b #11111111b, &P1DIR
    mov.b #11111111b, &P2DIR
    mov.b #00000000b, &P1OUT
    mov.b #00000000b, &P2OUT
    mov.b #00000001b , R6
    mov.b #00000001b , R7
    mov.b #0d , R8
    mov.b #0d , R9
LoopStart
    cmp #0d , R9
    jeq GoLeft
    mov.b R7, &P2OUT
    mov.b #00000000b, &P1OUT
    mov.b #0d, R9
    jmp GoRight
GoLeft
    mov.b R6, &P1OUT
    mov.b #00000000b, &P2OUT
    inc R9
GoRight
    inc R8
    rla R6
    rla R7
    mov.w #00500000 , R15
DelayLoop
    dec.w R15
    jnz DelayLoop
    cmp #8d , R8
    jeq Setup
    jmp LoopStart
```

2.3 Part 3

In this section, we used Port 1 to observe the LED lights and a button from Port 2 to control them. In this setup, the LEDs on the left side light up sequentially from the bottom to the top. The loop can be paused at any point by pressing the button. The detailed implementation is as follows:

Setup

- Configure `P2DIR` as an output by initializing it to `11111111`.
- Configure `P1DIR` as an input by initializing it to `00000000`.
- Set the output of Port 2 (`P2OUT`) to 0.
- Initialize register `R6` with the value `00000001` to start the sequence from the bottom-left LED.
- Set register `R8` to 0 to count the loop iterations.

MainLoop

- Assign the value of `R6` to `P2OUT`.
- Increment `R8` in each iteration to keep track of the active LED.
- Perform an arithmetic left shift on `R6` to turn on the next LED in the sequence.
- Set `R15` to `00500000` to introduce a delay.

Delay

- Decrement `R15` until it reaches 0.
- If the zero flag (Z) is not set, jump back to `Delay` to continue the delay loop.

CheckBtn

- Set the value of `P1IN` to `10000000` to enable the button.
- If the zero flag (Z) is not set, jump to the `TurnOff` label, which stops the sequence permanently when the button is pressed.
- Check if all 8 LEDs are lit.
  - If not, jump back to `MainLoop` to continue the sequence.
  - Otherwise, jump to `Setup` to restart the loop.

You can find code below.

```
Setup
    mov.b #11111111b, &P2DIR
    mov.b #00000000b, &P1DIR
    mov.b #00000000b, &P2OUT
    mov.b #00000001b, R6
    mov.b #0d, R8
MainLoop
    bis.b R6, &P2OUT
    inc R8
    rla R6
    mov.w #00500000, R15
Delay
    dec.w R15
    jnz Delay
CheckBtn
    bit.b #10000000, &P1IN
    jnz TurnOff
    cmp #8d, R8
    jne MainLoop
    jmp Setup
```

3 RESULTS

All programs executed successfully. We observed that in scenarios like these, it is possible to adjust the delay, break the programs into smaller steps, and efficiently manage the desired registers.

4 DISCUSSION

We discovered that by simply using registers and loops, we can control LEDs in any desired pattern. Through this process, we realized that computers operate using numerous simple instructions like these. During the experiment, we reflected on how many lines of code would be necessary to build a functioning computer.

The board had some issues, such as the two bottom-right LEDs not working and the sides being reversed. However, our code functioned as expected.

Additionally, we explored the difference between the bis and mov operations. The bis operation performs a bitwise OR between the source and destination, while the mov operation transfers the source value to the destination (effectively setting the destination to the source). The bis operation was particularly helpful for sequentially activating LEDs without turning off the ones that were already lit. Meanwhile, the mov operation proved useful in various parts of our modules.

5 CONCLUSION

This experiment enhanced our understanding of assembly language and the MSP430G2553 microcontroller. We learned how to use bitwise operations, such as bis and mov, to efficiently control hardware like LEDs. Although we faced challenges, such as non-functional LEDs and reversed sides, the code functioned as intended. This experience demonstrated how simple instructions can build complex systems and improved our troubleshooting skills. Overall, the experiment was a valuable step in developing our knowledge of embedded systems.

REFERENCES