**Department of Computer Engineering**

BLG 351E  
Microcomputer Laboratory Experiment Report

Experiment No : 1

Experiment Date : 09.10.2017

Group Number : Monday - 17

Group Members :

|  |  |  |
| --- | --- | --- |
| **ID** | **Name** | **Surname** |
| 150150302 | İrem Nur | Demirtaş |
| 150160706 | Merve Elif | Demirtaş |
| 150140719 | Cemal | Türkoğlu |
| 150140722 | İbrahim | Dolapci |

Laboratory Assistant : Ahmet Arış

# Introduction

The objective of this experiment was to program the microcomputer to first, make the last LED blink in an infinite loop in place, then make all the LEDs blink in sequence one after another and lastly make all the LEDs blink back and forth in an infinite loop. For this objective, first, the given assembly code was written, uploaded to the microcomputer, debugged and run, then it was altered to achieve the other tasks. Throughout the project, assembly language was used.

# Experiment

Before moving onto the following parts, a project was created on Code Composer Studio which included an assembly file named main.asm. As instructed, the following codes were added in the area marked “Main loop here” through the experiment. Afterwards, the microcomputer was connected, the project was built, run and debugged when necessary.

## Part I

In this part of the experiment, the task was to transfer the given code to the microcomputer. What the below code does is activating the last LED at P1.0 output and making it blink in an infinite loop. To better understand how the microcomputer works and what parts of it should be altered to making blink in the way asked in the other two parts, the code was debugged line by line and the values at registers were observed. The label SetupP1 activates the last LED at P1; Mainloop label XORs the value at P1.0 with 0x01 to toggle it; Wait sets a counter to be decremented to observe the changes; L1 decrements R1 until it is 0, and when it is, it jumps back to Mainloop to continue on the infinite loop.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | SetupP1 | bis.b | #001h, &P1DIR | ; Activates the first LED at P1.0 output |
| 2 |  |  |  | ; |
| 3 | Mainloop | xor.b | #001h, &P1OUT | ; Toggles P1.0 |
| 4 | Wait | mov.w | #050000, R15 | ; Sets a counter to cause delay to observe the change (blinking) |
| 5 | L1 | dec.w | R15 | ; Decrement counter |
| 6 |  | jnz | L1 | ; Checks if delay is over |
| 7 |  | jmp | Mainloop | ; Returns to start of Mainloop |

Code for Part 1

## Part II

In this part of the experiment, the code written was altered in such a way that it made all the lights at P1.0 output blink one by one in order. It was obvious that the pattern is binary 1 being multiplied by 2 seven, which is the same as shifting it to left 7 times. So, the rough plan was to load 0x01 to the output, make the corresponding LED blink, shift it, load the resultant value, make it blink and so on.

In order to achieve this, another loop labeled bas, and three lines of code were added between the jump instructions in addition to changing Mainloop altogether. The bas loop resets R5 and sets R4. Later on, R5 is used as a counter to make sure the loop goes on 8 times and R4 is used to turn the LEDs on. Mainloop now turns all the LEDs at P1OUT off, and forwards the value at R4 to turn the led corresponding to its 1 bit on. R15 is still used as a counter to pass the time. L1 decrements R15, if it is not zero, goes back to the start of L1, else, it shifts the value of R4 to left. Then, it increments R5 counter, if it is 8, it jumps to bas label to start over, if not, it jumps back to Mainloop to continue until the counter is 8. The code can be seen below.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | SetupP1 | bis.b | #0ffh, &P1DIR | ; Activates all LEDs at P1.0 output |
| 2 | bas | clr.b | R5 | ; Clears the contents of R5 to use as counter |
| 3 |  | mov.b | #001h, R4 | ; Sets the contents of R4 as #001h |
| 4 |  |  |  | ; |
| 5 | Mainloop | clr.b | &P1OUT | ; Turns all LEDs at P1 off |
| 6 |  | bis.b | R4, &P1OUT | ; Turns the LED at P1 according to 1 bits of R4 |
| 7 | Wait | mov.w | #050000, R15 | ; Sets R15 as a counter |
| 8 | L1 | dec.w | R15 | ; Decrements R15 |
| 9 |  | jnz | L1 | ; Checks if R15 is zero, jumps to L1 if it is not |
| 10 |  | rla.b | R4 | ; Aritmetically shifts the content of R4 to left |
| 11 |  | inc.b | R5 | ; Increments R5 (counter) |
| 12 |  | cmp | #008h, R5 | ; Compares the value with R5 |
| 13 |  | jeq | bas | ; If R5 is 8 (in decimal), jumps to bas label |
| 14 |  | jmp | Mainloop | ; Returns to start of Mainloop |

Code for Part 2

What this code does is making the LEDs blink in the following pattern:

⚫⚫⚫⚫⚫⚫⚫⚪

⚫⚫⚫⚫⚫⚫⚪⚫

⚫⚪⚫⚫⚫⚫⚫⚫

⚪⚫⚫⚫⚫⚫⚫⚫

⚫⚫⚫⚫⚫⚫⚫⚪

⚫⚫⚫⚫⚫⚫⚪⚫

Figure 1

The part where this code differs from the first one is that the first one did not require a register to hold a value for making the LED blink as it only had to make one LED blink whereas the shifting operation made it necessary to use a register to keep the initial and current values.

## Part III

In this part of the experiment, the code in the second part was altered further to change the direction of the order in which the LEDs blink once the leftmost and rightmost LEDs blink so that rather than resetting they would blink back and forth. The plan was to construct an additional loop that shifts the value to the right and decrement the counter until it is zero. Once it is zero it would jump to the loop that increments the counter and shifts the value to the left, which is Mainloop. When the program reaches the end of Mainloop, rather than jumping back to either bus or Mainloop loops, it would jump back to bas or decrement loops. So, shortly, the idea was to connect two loops one of which increments the counter and shifts right and the other decrements the counter and shifts left.

It was implemented as the code seen below. The additions are azalt label, which sets the starting value of decrement loop as 0x40 at R6; Wait2 label, which is basically the same loop as Wait1; and L2 loop, which decrements R15 counter, which should be between 8 and 0, forwards the value at R6 to the output, shifts the value at R6, compares the value at R5 counter with 0, if it is 0 it jumps to bas label, if not, it jumps to flag label.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | SetupP1 | bis.b | #0ffh, &P1DIR | ; Activates all LEDs at P1.0 output |
| 2 | bas | clr.b | R5 | ; Clears the contents of R5 to use as counter |
| 3 |  | mov.b | #001h, R4 | ; Sets the contents of R4 as #001h |
| 4 |  |  |  | ; |
| 5 | Mainloop | clr.b | &P1OUT | ; Turns all LEDs at P1 off |
| 6 |  | bis.b | R4, &P1OUT | ; Turns the LED at P1 according to 1 bits of R4 |
| 7 | Wait | mov.w | #050000, R15 | ; Sets R15 as a counter |
| 8 | L1 | dec.w | R15 | ; Decrements R15 counter |
| 9 |  | jnz | L1 | ; Checks if R15 is zero, jumps to L1 if it is not |
| 10 |  | rla.b | R4 | ; Aritmetically shifts the content of R4 to left |
| 11 |  | inc.b | R5 | ; Increments R5 (counter) |
| 12 |  | cmp | #008h, R5 | ; Compares the value with R5 |
| 13 |  | jeq | bas | ; If R5 is 8 (in decimal), jumps to bas label |
| 14 |  | jmp | Mainloop | ; Returns to start of Mainloop |
| 15 |  |  |  | ; |
| 16 | azalt | mov.b | #040h, R6 | ; Sets the starting value of decrement loop |
| 17 | flag | dec.b | R5 | ; Decrements R5 |
| 18 |  |  |  | ; |
| 19 | Wait2 | mov.w | #050000, R15 | ; Restarts the wait loop |
| 20 | L2 | dec.w | R15 | ; Decrements R15 counter |
| 21 |  |  |  | ; |
| 22 |  | mov.b | R6, &P1OUT | ; Forwards the value at R6 to output |
| 23 |  | rra.b | R6 | ; Shifts the value at R6 right arithmetically |
| 24 |  | cmp.b | #000h, R5 | ; Compares the counter R5 with zero |
| 25 |  | jeq | bas | ; If R5 is 0, goes to start of bas label |
| 26 |  | jmp | flag | ; Else, goes back to start of flag label |
|  |  |  |  |  |

Code for Part 3

What this code does is making the LEDs blink in the following pattern:

⚫⚫⚫⚫⚫⚫⚫⚪

⚫⚫⚫⚫⚫⚫⚪⚫

⚫⚪⚫⚫⚫⚫⚫⚫

⚪⚫⚫⚫⚫⚫⚫⚫

⚫⚪⚫⚫⚫⚫⚫⚫

⚫⚫⚪⚫⚫⚫⚫⚫

Figure 2

# Conclusion

The only difficulty we faced was an occasional initialization error, the cause of which could not be determined and which could easily be fixed by disconnecting and connecting the microcomputer. One of the most crucial things we got from this session was the difference between bis and mov instructions. For the first two parts, we used these instructions interchangeably and did not notice they operated differently until Part III where we used bis instead of mov on line 16. Now we understand that bis sets the bits at the destination at the positions corresponding to 1 bits of source while mov copies the value on source to destination. For example, if the value on R4 is 1001 1001 in binary, bis #022h would make R4’s content 1011 1011 while mov would make it 0010 0010. This experiment helped us get a firmer grasp of the assembly language and learn how to use this particular microcomputer.