ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 351E

MICROCOMPUTER LABORATORY EXPERIMENT REPORT

EXPERIMENT NO	:	3
EXPERIMENT DATE	:	30.11.2020
LAB SESSION	:	MONDAY - 13.30
GROUP NO	:	G10

GROUP MEMBERS:

150180103 : EGEMEN GÜLSERLİLER

150180028 : ABDÜLKADİR PAZAR

150180716 : AHMET SELÇUK TUNÇER

AUTUMN 2020

Contents

1 INTRODUCTION

In this experiment, a system will be designed using interrupts to detect a button press in assembly language. Interrupts are the conditions that temporarily suspend the main program, pass the control to the external sources and execute their task. By using interrupts rare events can be detected.

2 MATERIALS AND METHODS

This experiment is conducted via using Arduino Uno Board, 8 led and 2 seven segment display. This board is programmed using Tinkercad IDE according to desired tasks on the experiment handout.

2.1 Part 1

In the first part of the experiment, a left shift program that is displayed on LEDs repeatedly with 1-second delay at each increment is designed.
In order to implement the desired feature, the following setup code given in Figure 1 was written.

- Line 5-14: These lines was already given with template code and it is used for 1 second delay.
- Line 15: This line was used for Carry=1 condition that if occured loop part is branched EX subroutine.
- Line 16-21: In these lines we set DDRD register to r16 register and set r16 as output and also we assign r17 register to 1 and set it as PORTD

In order to implement the desired feature, the following loop code given in Figure 2 was written.

- Line 30-31: Calling for 1 second delay and setting PORTD to register r16.
- Line 32-33: Left shift operation and if carry equals to 1 branch to EX subroutine
- Line 33-34: Setting r16 register as PORTD and jump to loop in order to continue loop.

```
1  void setup()
2  {
3    asm(
4  "        JMP start                  \n"
5  "delay:  LDI  r23,  81              \n" //Delay 1 sec
6  "w1:     LDI  r24,  255             \n"
7  "w2:     LDI  r25,  255             \n"
8  "w3:     DEC  r25                   \n"
9  "        BRNE w3                    \n"
10 "        DEC  r24                   \n"
11 "        BRNE w2                    \n"
12 "        DEC  r23                   \n"
13 "        BRNE w1                    \n"
14 "        RET                        \n"
15 "EX:     ROL     r16                \n"//Extra shift if carry=1
16 "start:                             \n" //Write your code
17 "        IN   r16,  0x0A ;          \n" //r16 <- DDRD
18 "        LDI  r16,  0xFF            \n" //r16[0] <- 1
19 "        LDI  r17,  0x01            \n" // r17[0] <- 1
20 "        OUT  0x0A, r16             \n" //DDRD <- r16
21 "        OUT  0x0B, r17             \n" // PORTD <- r17
22   );
23
24 }
```

Figure 1: Code for part 1

```
26 void loop()
27 {
28   asm(
29 "LOOP:                             \n"
30 "        CALL delay                 \n" //Call delay function
31 "        IN      r16,  0x0B ;       \n" //r16 <- PORTD
32 "        ROL     r16                \n" //CSL r16
33 "        BRCS   EX                 \n" //Branch if C=1
34 "        OUT  0x0B, r16             \n" //PORTD <- r16
35 "        JMP  LOOP                  \n" //continue loop
36   );
37 }
```

Figure 2: Code for part 1

2.2 Part 2

In this part we are given 2 seven segment display and 2 button for interrup handling. When first button clicked counter will increase by one and if second button clicked counter will decrease by 1.
In order to implement the desired feature, the following setup code given in Figure 3 and 4 was written.

- Line 5-14: These lines was already given with template code and it is used for 12 millisecond delay.
- Line 15-23: This lines was used for incrementing by 1. We set r19 register as PINB input. We increment via line 18 and if button is not released we skipped incrementation thanks to line 17. Also if increment reach 9, we use lines 19 and 20 by jumping w5 to set 1's digit 0 and increment 10's digit by 1.
- Line 24-32: In these lines we implemented decrements by 1. We set r19 register as PINB input. We decremented via line 27 and if button is not released we skipped decrement thanks to line 26. Also if decrement is 0, we use lines 28 and 29 by jumping w8 to set 1's digit 9 and decrement 10's digit by 1.
- Line 33-36: This lines is used to jump ART-TEN or AZT-TEN subroutines. And HUND subroutine is implemented if counting reaches 99. When this happen 10' digit set 0 and 1' digit incremented by 1.
- Line 37-44: In these lines we implemented 1's digit cases. If 1's digit reaches 9 and we push first button, we jumped ART-TEN and increment 10' digit by 1. If counting reaches 100 we jumped HUND subroutine to turn back 00. If 1's digit reaches 0 and we push second button, we jumped AZT-TEN and decrement 10' digit by 1 and set 1' digit 9.
- Line 46-55: We set DDRB to r16 register and DDRC to r17. r18 register is designed to keep and display the least significant digit. r19 and r20 is used to get values from PINB and PINC accordingly in the loop.

In order to implement the desired feature, the following loop code given in Figure 5 was written.

- Line 64-68: We ste PINB to r19 register. If first button clicked call ARTB, skip if not. If second button clicked call AZTB, skip if not.

- Line 69-72: This lines was used to set r18 register to PORTB and r20 to PORTC and we call delay.

```
1  void setup()
2  {
3    asm(
4  "         JMP start                \n"
5  "delay: LDI     r23,  1            \n"
6  "w1:    LDI     r24,  255          \n"
7  "w2:    LDI  r25,  255             \n"
8  "w3:    DEC  r25                   \n"
9  "       BRNE w3                    \n"
10 "       DEC  r24                   \n"
11 "       BRNE w2                    \n"
12 "       DEC  r23                   \n"
13 "       BRNE w1                    \n"
14 "         RET                      \n"
15 "ARTB:  IN      r19,  0x03;        \n"
16 "       CLZ                        \n"
17 "       SBRS r19, 4                \n"
18 "         INC r18                  \n"
19 "       CPI r18,10                 \n"
20 "       BREQ w5                    \n"
21 "         SBRS r19, 4              \n"
22 "w6:      RET                      \n"
23 "       JMP ARTB                   \n"
24 "AZTB:  IN      r19,  0x03;        \n"
25 "       CLZ                        \n"
26 "       SBRS r19, 5                \n"
27 "       DEC r18                    \n"
28 "       CPI r18, 0xFF              \n"
29 "       BREQ w8                    \n"
30 "       SBRS r19, 5                \n"
31 "w7:      RET                      \n"
32 "       JMP AZTB                   \n"
33 "w5:    JMP ART TEN                \n"
```

Figure 3: Code for part 2

```
"w5:    JMP ART_TEN                 \n"
"w8:    JMP AZT_TEN                 \n"
"HUND:  LDI r20,0x00                \n"
"       JMP w6                      \n"
"ART_TEN: CLR r18                   \n"
"           INC r20                 \n"
"       CPI r20,10                  \n"
"           BREQ HUND               \n"
"       JMP w6                      \n"
"AZT_TEN: LDI r18, 0x09             \n"
"       DEC r20                     \n"
"       JMP w7                      \n"
"                                   \n"
"start:                             \n"
"       IN      r16,  0x04 ;        \n"
"       LDI     r16,  0b11001111    \n"
"       OUT  0x04, r16              \n"
"       IN      r17,  0x07 ;        \n"
"       LDI     r17,  0b11001111    \n"
"       OUT  0x07, r17              \n"
"       LDI  r18, 0x00              \n"
"       LDI  r19, 0x00              \n"
"       LDI r20, 0x00               \n"
  );
```

Figure 4: Code for part 2

```
void loop()
{
  asm(
"LOOP:                              \n"
"        IN      r19,  0x03;        \n"
"        SBRC    r19, 4             \n"
"        CALL    ARTB               \n"
"        SBRC    r19, 5             \n"
"        CALL    AZTB               \n"
"        OUT     0x05, r18          \n"
"        OUT     0x08, r20          \n"
"        CALL    delay              \n"
"        JMP     LOOP               \n"
  );
}
```

Figure 5: Code for part 2

2.3 Part 3

In this part we are asked to develeop an up-counter. We are given 2 seven segment display, 8 leds and 4 buttons. First and second button were supposed to control increment value, and third and fourth button were supposed to control delay duration. Moreover count value should diplayed on both seven segments and leds. In order to implement the desired feature, the following setup code given in Figure 6 was written.

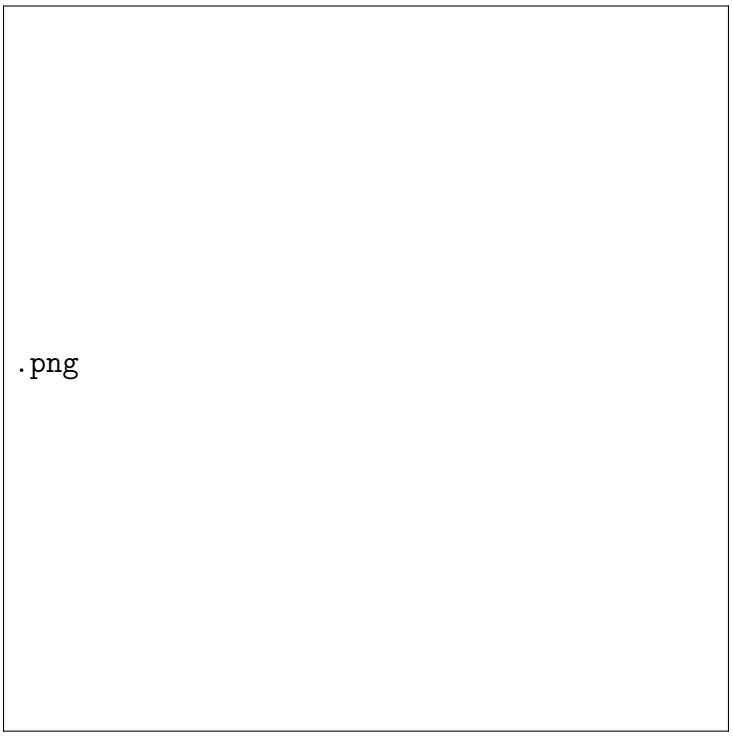

Figure 6:

XXX

3 RESULTS

In the first part of the experiment an assembly program created that generates the given sequence - left shift. Result of the program is observed on the leds and everything was smoothly and consistent with our theoretical knowledge.

In the second part of the experiment, simple two-digit decimal counter is programmed and result is displayed on 7 segment displays. When first button is clicked counter has incremented and when second button is clicked counter has decremented as desired. Also edge cases incrementation from x9's to x0's, and decrement from x0's to x9's is performed well. Moreover 99 to 100 case is also taken into account.

In the third part of the experiment, desired up-counter has programmed via assembly

language. The 7 segment displays showed the output number as decimal number and leds showed the output number as binary value. Increment number has controlled via first and second button. When first button clicked increment number has decreased by 1 and when second button clicked increment number has increased by 1. Moreover, third and fourth buttons programmed to control delay period. When third button clicked increment number has decreased by 100ms and when fourth button clicked increment number has increased by 100ms. And as requested, push-buttons did not work circular.

time	led pattern
0	0000000**1**
1	000000**1**0
2	00000**1**00
3	0000**1**000
4	000**1**0000
5	00**1**00000
6	0**1**000000
7	**1**0000000
8	0000000**1**
9	000000**1**0
10	00000**1**00
11	0000**1**000
12	000**1**0000
13	00**1**00000
14	0**1**000000

Figure 7: Desired output sequence for Part 1

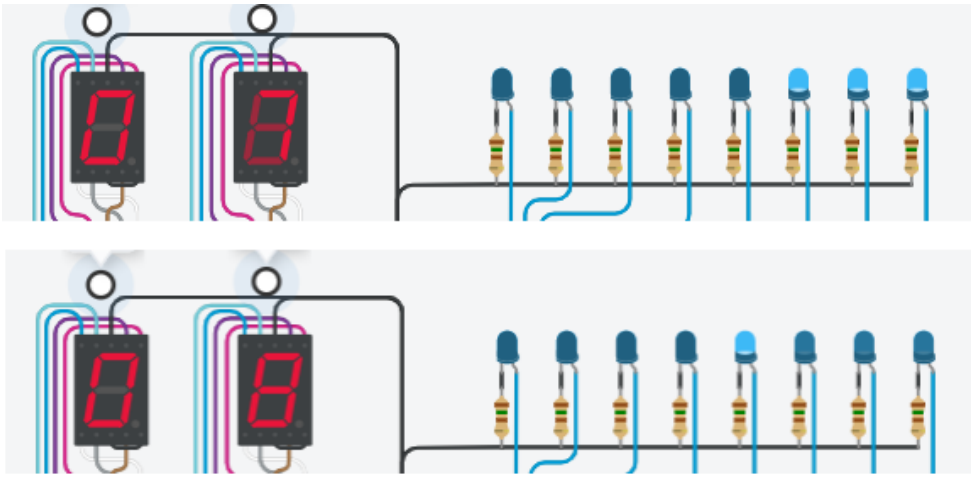

Figure 8: Output of the seven segment displays and leds for Part 3

4 DISCUSSION

At the end of the experiment, the importance and role of the interrupt handling were noticed by team members. Also how 7-segment display works and how can be manipulated are learned. How the registers are controlled and port manupulation via assembly language has learned.

5 CONCLUSION

With this experiment, team have gained more experience with assembly programming. In this experiment, the interrupt handlings are learned. This experiment was the hardest experiment compared to first 2 experiments but the one of the most important experiment. The team has spent much moretime to complete this experiment.