ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 351E
MICROCOMPUTER LABORATORY EXPERIMENT REPORT

EXPERIMENT NO	:	3
EXPERIMENT DATE	:	04.12.2020
LAB SESSION	:	FRIDAY - 15.30
GROUP NO	:	G11

GROUP MEMBERS:

150170063 : BURAK ŞEN

150180080 : BAŞAR DEMİR

150190719 : OBEN ÖZGÜR

FALL 2020

Contents

1 INTRODUCTION

In this week's experiment, we have faced a milestone of our computer engineering knowledge in our Arduino experiment because for this week's experiment we have only used inline Assembly to manipulate our Arduino circuitry. Circuit given to us had 8 LEDs attached to its digital ports just as previous weeks, 4 buttons to control our circuitry. In addition, as opposed to previous weeks we had two 7 segment display attached to our circuit. We have continued to design patterns similar to previous weeks but by only using inline Assembly which was completely new to all of us.

2 MATERIALS AND METHODS

- Tinkercad
- Arduino Uno R3
- 8 Resistors
- 8 LEDs
- 2 Buttons
- 2 Switches

2.1 PART 1

In this part of experiment, we are expected to implement an assembly program that generates the sequence over LEDs using Arduino. We have a knowledge about programming structure and registers of Arduino.

We used D-Ports to manipulate LED transitions and we used all ports as an output. Therefore in start label, we have to define this property. Using register-16 (r16), we read the value that is written in DDRD and we have performed immediate OR operation with all 1 values. Then, we have written values that are in r16 to DDRD register. Address of the DDRD register is given in table that is placed below. While implementing Arduino programming, we cannot used directly DDRD keyword, instead of this we used address of DDRD register.

To manipulate LEDs, we used r17 register. This register initialized with 1 in start label. In every LOOP iteration, we have performed right circular shift to obtain required pattern. To observe our patter in LEDs, we have given value of r17 as a output to PORTD (0x0b) register.

To solve the problem of transition from the far left to the right, we have used ROL (Rotate left through carry) instruction. When it performs shift operation to 0b10000000, number becomes 0 and carry bit is occured. Using BREQ, we have checked Z flag and when number become 0, it branches to lefttoright label. In this label we perform another ROL operation and it writes carry bit to least sgnificant bit of register. Then we jump to beginning of the for loop.

Our implementation is given below.

Register Name	Address
PINB	0x03
DDRB	0x04
PORTB	0x05
PINC	0x06
DDRC	0x07
PORTC	0x08
PIND	0x09
DDRD	0x0A
PORTD	0x0B

Table 1: Addresses of the registers

```
void setup()
{
  asm(
"          JMP start                    \n"
"delay:    LDI     r23,   81            \n"      //Delay 1 sec
"w1:       LDI     r24,   255           \n"
"w2:       LDI  r25,   255              \n"
"w3:       DEC  r25                     \n"
"          BRNE w3                      \n"
"          DEC  r24                     \n"
"          BRNE w2                      \n"
"          DEC  r23                     \n"
"          BRNE w1                      \n"
```

```
"        RET                      \n"
";Your Functions Code Begin       \n" //Write your code
"    PRINTCARRY:                  \n" //  printcarry subroutine
"    CALL delay                   \n" //  call delay
"    ROL r17                      \n" //  right circular shift
"                                 \n"
";Your Functions Code End         \n"
"                                 \n"
"start:                           \n" //Write your code
"    IN    r16,  0x0A             \n"     // read DDRD
"    ORI   r16,  0b11111111       \n"// r16 <- define them as outputs
"    LDI   r17,  0b00000001       \n" //r17 <- 1
"    OUT   0x0A, r16              \n"  // assignes DDRD as output
  );

}

void loop()
{
  asm(
"LOOP:                            \n"//Write your code
";Your LOOP Code Begin            \n"
"    OUT   0x0b, r17              \n" //PORTD <- r17
"    ROL   r17                    \n"// right circular shift
"    BREQ PRINTCARRY              \n"
//if there is carry branch to printcarry
"    CALL delay                   \n"//Call delay function
";Your LOOP Code End              \n"
"        JMP   LOOP               \n"
  );

}
```

2.2 PART 2

In this part, we were expected to implement a two digit counter that counts up if one of the buttons is pressed and counts down if the other one of them pressed.

In the start subroutine we did get our pins to local registers and assign them if they are input or outputs and we get outputs from them. We defined five other registers for other operations.

In the loop subroutine when we take inputs in the register and show them with the same register we could not take more than one input, we realised if we increment our value we could not control this value again because it was changed when it is changed. We approached this with another way. When we take inputs we right shift them constant amount of time our input value becomes the 0'th value and we can read as we like. Then we controlled if our input is increment or decrement if it is increment we branch to increment lower subroutine if it is decrement we branch to decrementlower subroutine.

In the incrementlower subroutine we increment our lower digit, if it is become ten we branch to incrementupper subroutine and we load 0 to lower digit and increment the upper digit by one. In the incrementupper subroutine we also check if upper digit is nine or not as well, if it is nine we branch to tozero subroutine. And in the tozero subroutine we make zero both of the digits.

In the decremtnlower subroutine we decrement our lower digit, if it is become zero we branch to decrementupper subroutine and we load nine to lower digit and decrement the upper digit by one. In the decrementupper subroutine we also check if upper digit is zero or not as well, if it is 0 we branch to tonintynine subroutine. And in the tonintynine subroutine we make nine both of the digits.

For the push button control we use another register for holding last input. If the last input is equal the present input we branch it to delay with calldelay subroutine but if it is not we continue our program.

```
void setup()
{
  asm(
"   JMP start                    \n"
"delay: LDI  r23,  40            \n" //Delay 1 sec
"w1:    LDI  r24,  120           \n"
"w2:    LDI  r25,  120           \n"
"w3:    DEC  r25                 \n"
"       BRNE w3                  \n"
"       DEC  r24                 \n"
"       BRNE w2                  \n"
"       DEC  r23                 \n"
"       BRNE w1                  \n"
"       RET                      \n"
";Your Functions Code Begin      \n" //Write your code
"incrementupper:                 \n"
//incrementupper subroutine for incrementing upper digit
"       CALL delay               \n" //call delay
"       CPI r20, 0b00001001      \n" //compare if upper digit is 9 or not
"       BREQ tozero              \n" //if it is 9 branch to to ninty nine
"       INC r20                  \n" //increment upper digit
"       LDI r19, 0b00000000      \n" //load lower digit with 0
"       JMP LOOP                 \n" //jump to loop
"                                \n"
"tozero:                         \n"
//tozero subroutine for making both digits 0
"       LDI r20, 0b00000000      \n" //load upper digit with 0
"       LDI r19, 0b00000000      \n" //load lower digit with 0
"       JMP LOOP                 \n" //jump to loop
"tonintynine:                    \n"
// tonintynine subroutine for making both digits 9
"       LDI r20, 0b00001001      \n" //load upper digit with 9
"       LDI r19, 0b00001001      \n" //load lower digit with 9
"       JMP LOOP                 \n" //jump to loop
```

```
"incrementlower:          \n"
//incrementlower subroutine for incrementing lower digit
"        CALL delay              \n" //cal delay
"        INC r19                 \n" //increment lower digit
"        CPI r19, 0b00001010     \n" //compare if lower digit is 10
"        BREQ incrementupper     \n" //if it is branch to increment upper
"        JMP LOOP                \n" //jump to loop
"                                \n"
"decrementupper:                 \n"
//decrementupper subroutine for decrementing upper digit
"        CALL delay              \n" //call delay
"        CPI r20, 0b00000000     \n" //compare if upper digit is 0 or not
"        BREQ tonintynine        \n"
// if it is 0 branch to subroutine tonintynine
"        DEC r20                 \n" //decrement upper digit
"        ORI r19, 0b00001001     \n" //or immediate lower digit with 9
"        JMP LOOP                \n" //jump to loop
"                                \n"
"decrementlower:                 \n"
//decrement lower subroutine for decrementing lower digit
"        CALL delay              \n" //  call delay
"        CPI r19, 0b00000000     \n"
//compare if lower digit is zero or not
"        BREQ decrementupper     \n"
//if it is 0 branch decrementupper subroutine
"        DEC r19                 \n" //decrement lower digit
"        JMP LOOP                \n" //jumo to loop
"                                \n"
"calldelay:                      \n"
//calldelay subroutine for controlling buttons
"        CALL delay              \n" //call delay
"        JMP LOOP                \n" //jump to loop

"start:                          \n" //Write your code
"        IN r17, 0x07            \n" // <---- DDRC analog
"        ORI r17, 0b001111       \n"     // input output
"        OUT 0x07, r17           \n" // assign etme
```

```
"        IN r18, 0x04            \n" //DDRB soldaki digital
"        ORI r18, 0b001111       \n" //input output
"        OUT 0x04, r18           \n" //assgin
"        LDI r19, 0b00000000     \n" // lowerdaki say
"        LDI r20, 0b00000000     \n" // upperdaki say
"        LDI r21, 0b000000       \n" //temp
"        LDI r22, 0b100000       \n" //temp_for_12
"        LDI r16, 0b100000       \n" //temp_for_13
  );
}

void loop()
{
  asm(
"LOOP:                           \n"//Write your code
";Your LOOP Code Begin                   \n"
"        OUT 0x05,r19            \n" //out lower digit
"        OUT 0x08,r20            \n" //out upper digit
"        IN r17, 0x03            \n" //take inputs from buttons
"        CP r21, r17             \n"
//compare if inputs are the same inputs from before
"        BREQ calldelay          \n"  //if it is branch to calldelay
"        LSR r21                 \n" //  right shift for taking inputs
"        LSR r21                 \n"
"        LSR r21                 \n"
"        LSR r21                 \n"
"        CPI r21, 0b00000001     \n" //compare if input is 1
"        BREQ incrementlower     \n" // if it is branch to incrementlower
"        LSR r21                 \n"  //right shift again
"        CPI r21, 0b00000001     \n" //compare if input is 1
"        BREQ decrementlower     \n" //if it is branch decrementlower
"        MOV r21, r17            \n"
//mov input value to previnput register
"        JMP  LOOP               \n" //jump loop
  );
}
```

2.3 PART 3

In this part, we are expected to implement a counter that's delay and increment number are determined by pushing buttons.

As we always perform, we determined our variables and input, output ports in start label. We load DDRC(0x07) with 0b001111 to used last two pins as input and similarly, we load DDRB(0x04) with 0b001111 to used last two pins as input. To manipulate LEDs, we have assigned DDRD(0x0a) all 1's.

In this part we used 5 variables that are used for lower digit representation(r19), upper digit representation(r20), delay multiplier counter(r21), increment counter(r22) and temporary usage(r16). We have initialized these values.

In this part we used 5 variables that are used for lower digit representation(r19), upper digit representation(r20), delay multiplier counter(r21), increment counter(r22) and temporary usage(r16). We have initialized these values.

We have written logic of the add operation in the second part of experiment. But in this part, we have implemented a recursive label that is used for increment operation. We assign increment number to our temp variable. Every iteration, we call mainincrement function. Every call of this function, we decrease our temp variable and check it if it is zero or not. If it is not equal to zero, we branch to increment lowerdigit label. It works similar with part-2 implementation. But in the end of label, it jumps mainincrement label instead of loop. By performing this operation we decrement our temp value, when it reaches to 0, it returns. This logic allows us to perform for operation similar to programming languages. We have implemented same structure for delay operation. We assign value of delay counter to temp and we call maindelay label that is branches to delay function delay counter times.

To use buttons that are separated to control increment and delay quantity, we have check our B and C pins. We know that our buttons are connected to 2 most significant bits of ports. To reach them, we used left shift instruction. We have shifted our inputs until we reach the button bits. Using compare immediate operation, we checked that if they are pushed or not. If one of them pushed, we branched to related label. For every control label we have checked our boundaries to prevent range crossing. If it is proper, we have incremented or decremented our delay or increment counter. Thus, we have completed button parts.

To manipulate our seven segment displays and LEDs, we have given our registers as outputs to their addresses. We have already keep our digits in separate registers. Therefore, we give these values to displays directly. But we did not keep whole number in a register. So, we used multiplication instruction. We multiply our upper digit with 10 and it writes answer to r0. Then, we add lower digit to this register. By this way, we obtain whole number and we give it as an output to LEDs.

Our implementation is given below.

```
void setup()
{
  asm(
"         JMP start               \n"
"delay:   LDI      r23,  8        \n"//Delay 100 msec
"w1:      LDI      r24,  26       \n"
"w2:      LDI  r25,  26           \n"
"w3:      DEC  r25                \n"
"         BRNE w3                 \n"
"         DEC  r24                \n"
"         BRNE w2                 \n"
"         DEC  r23                \n"
"         BRNE w1                 \n"
"         JMP maindelay           \n" //jump to main
";Your Functions Code Begin       \n" //Write your code
"maindelay:                       \n"//main subroutine for delay
"         DEC r16                 \n" //decrement delay counter
"         CPI r16,0b00000000      \n" //compare if delay counter is 0
"         BRNE delay              \n" //if it is zero branch to delay subrout
"         RET                     \n" //return from maindelay
"incrementupper:                  \n"//increment subroutine for upper digit
"         CPI r20, 0b00001001     \n"//compare if it is 9
"         BREQ tozero             \n"//if is 9 branch to tozero subroutine
"         INC r20                 \n"//increment upper digit
"         LDI r19, 0b00000000     \n"//load 0 to lower digit
"         JMP mainincrement       \n"//jump to mainincrement
"                                 \n"
"tozero:                          \n"
//to zero subroutine for making zero
```

```
"        LDI r20, 0b00000000     \n"//load zero to upper digit
"        LDI r19, 0b00000000     \n"//load zero to lower digit
"        JMP mainincrement       \n"//jump to mainincrement
"mainincrement:                  \n"//mainincrement subroutine
//in this subroutine we increment our digits according to decrement
//counter
"        DEC r16                 \n"//decrement counter
"        CPI r16,0b00000000      \n" //compare if decrement counter is 0
"        BRNE incrementlower     \n" //if it is not 0 branch increment lower
"        RET                     \n" //return from mainincrement
"incrementlower:                 \n"//increment subrouite for lower digit
"        INC r19                 \n"//increment lower digit
"        CPI r19, 0b00001010     \n"//compare if lower digit is 9
"        BREQ incrementupper     \n"//if it is 9 branch increment upper
"        JMP mainincrement       \n"//jump to mainincrement
"                                          \n"
"plusincrement:                  \n"
//for increment counter plusincrement subroutine
"        CPI r22, 0b00001010     \n"//compare if counter is 10
"        BREQ LOOP               \n"//if it is branch to loop
"        INC r22                 \n" //increase counter
"        JMP LOOP                \n" //jump to loop
"negativeincrement:              \n"
//for increment counter negativeincrement subroutine
"        CPI r22, 0b00000001     \n"//compare if counter is 1
"        BREQ LOOP               \n"//if it is branch to loop
"        DEC r22                 \n"//decrement counter
"        JMP LOOP                \n" //jump to loop

"plusdelay:                      \n"//for delay counter plusdelay
"        CPI r21, 0b00001010     \n"//compare if counter is 10 it means 1000
"        BREQ LOOP               \n"//branch to loop
"        INC r21                 \n" //increase counter
"        JMP LOOP                \n" //jump to loop
"negativedelay:                  \n"//for delay counter negativedelay
"        CPI r21, 0b00000001     \n"//compare if counter is 1 it means 100ms
"        BREQ LOOP               \n"//if it is branch to loop
```

```
"       DEC r21                  \n"//decrement counter
"       JMP LOOP                 \n" //jump loop

"start:                          \n" //Write your code
"       IN r17, 0x07             \n" //DDRC analog
"       ORI r17, 0b001111        \n"//input output
"       OUT 0x07, r17            \n" //assign
"       IN r18, 0x04             \n" //DDRB left digital
"       ORI r18, 0b001111        \n" //input output
"       OUT 0x04, r18            \n" //assign
"       IN      r16,  0x0A ;     \n"//Get values of DDRD to r16
"       ORI     r16,  0b11111111\n" //Assign ones to define as an output
"  OUT  0x0A, r16               \n" //Give r16 values to DDRD
"       LDI r19, 0b00000000      \n" // lower digit
"       LDI r20, 0b00000000      \n" // upper digit
"       LDI r21, 0b00000001      \n" // delay counter
"       ORI r22, 0b00000001      \n" // increment counter
"       LDI r16, 0b10000000      \n" // temp-for-loops
  );
}

void loop()
{
  asm(
"LOOP:                           \n"//Write your code
";Your LOOP Code Begin           \n"//LOOP function
"       MOV r16,r22              \n" //move counter to temporarly register
"       INC r16                  \n"//increment temp register by one
"       CALL mainincrement       \n"//call mainincrement
"       MOV r16, r21             \n"//move counter to temporarly register
"       INC r21                  \n"//increment temp register by one
"       CALL maindelay           \n"//call maindelay
"       OUT 0x05,r19             \n"//out lower digit
"       OUT 0x08,r20             \n"//out upper digit
"       LDI r16, 0b00001010      \n"//r16 <- 10
"       MUL r16,r20              \n"//r0<-upperdigit*10
"       ADD r0,r19               \n"//r0 += lower digit
```

```
"        OUT 0x0b  ,r0               \n"//out number to LEDs
"        IN r18, 0x03                \n" //get input from pinb
"        LSR r18                     \n"//right shift input register
"        LSR r18                     \n"
"        LSR r18                     \n"
"        LSR r18                     \n"
"        CPI r18, 0b00000001         \n" //compare if imput register is 1
"        BREQ plusincrement          \n"//if it is branch plusincrement
"        LSR r18                     \n"//right shift input register
"        CPI r18, 0b00000001         \n" //compare if input register is 1
"        BREQ negativeincrement      \n"//if it is branch negativeincrement
"        IN r17, 0x06                \n" //get input from pinc
"        LSR r17                     \n"//right shift input register
"        LSR r17                     \n"
"        LSR r17                     \n"
"        LSR r17                     \n"
"        CPI r17, 0b00000001         \n" //compare if input register is 1
"        BREQ plusdelay              \n"//if it is branch plusdelay
"        LSR r17                     \n"//right shift input register
"        CPI r17, 0b00000001         \n" //compare if input reigster is 1
"        BREQ negativedelay          \n"//if it is branch negativedelay
"        JMP  LOOP                   \n"
  );
}
```

2.4 PART 4

In this part, we are expected to implement a system that's delay and pattern is changed by buttons.

We have started with implementing start label. We have perform proper initialization of DDR registers to identify which pin is input or output. We have used DDRB and DDRC for both operations, because we have read our inputs from there. Also, we have determined our needed variables that are step counter (r19), sequence counter(r20), delay multiplier counter(r21), flag (r22) and temp (r16).

In the loop label, firstly we give outputs to seven segment display to show our delay multiplier and pattern number. Then, we take our delay multiplier to temp variable then we call for maindelay label. That label operates similar with the mechanism that is implemented in part-3. That is recursive operation and it calls delay label for delay multiplier times.

We have to check buttons to catch push activity that is handled by the input operations. We have read inputs from registers and by shifting registers, we got the proper bits of our inputs. If we push the first or second buttons, it branches to sequence number labels that increment or decrement our sequence counter. Thus, we can change our patterns. In these labels, we check the boundaries (1-4) of pattern numbers. If there is a breach, it branches to loop without any operation. We have perform same operations for delay multiplier.

To manipulate LEDs as we are expected, we wrote some instructions. We have check our pattern with our register that is used for keeping pattern number. Using branch operation, we can branch to related label. For our case it is sequenceone. In sequenceone, our LED transition return from the edges. To implement this mechanism, we check our LED's current position and we keep the transition direction in the flag register. If LED reaches to edges, we branch to a label that changes our direction flag and jumps to related label (right or left transition label). In these labels we are basically shift the registers that are responsible for manipulating LEDs then it gives them as an output and they jumps to loop label. Also we wrote a label that initialize the pattern respect to pattern number.

Our implementation is given below.

```
void setup()
{
  asm(
"       JMP  start          \n"
"delay: LDI     r23,  8     \n"//Delay 100 msec
"w1:    LDI     r24,  26    \n"
"w2:    LDI  r25,  26       \n"
"w3:    DEC  r25            \n"
"       BRNE w3             \n"
"       DEC  r24            \n"
"       BRNE w2             \n"
"       DEC  r23            \n"
"   BRNE w1                 \n"
```

```
"         JMP maindelay             \n" //jump to maindelay
";Your Functions Code Begin         \n" //Write your code
"maindelay:                         \n"//main subroutine for delay
"         DEC r16                   \n" //decrement delay counter
"         CPI r16,0b00000000        \n" //compare if delay counter is 0
"         BRNE delay                \n" //if it is zero branch to delay subrout
"         RET                       \n" //return from maindelay
"plusdelay:                         \n"//for delay counter plusdelay
"         CPI r21, 0b00001001       \n"//compare if counter is 9 it means 1000m
"         BREQ LOOP                 \n"//branch to loop
"         INC r21                   \n" //increase counter
"         JMP LOOP                  \n" //jump to loop
"negativedelay:                     \n"//for delay counter negativedelay
"         CPI r21, 0b00000001       \n"//compare if counter is 1 it means 100ms
"         BREQ LOOP                 \n"//if it is branch to loop
"         DEC r21                   \n"//decrement counter
"         JMP LOOP                  \n" //jump loop
"plussequence:                      \n"//Makes transition to upper patterns
"         CPI r21, 0b00000100       \n"//if pattern number is 4
"         BREQ LOOP                 \n"//branch to loop
"         INC r20                   \n" //increase pattern number
"         LDI r19, 0b00000000       \n"//sets step number to 0
"         LDI r22, 0b00000000       \n"//sets flag to 0
"         JMP LOOP                  \n" //jump to loop
"negativesequence:                  \n"//Makes transition to lower patterns
"         CPI r21, 0b00000001       \n"//if pattern number is 1
"         BREQ LOOP                 \n"// branch to loop
"         DEC r20                   \n"//decrease pattern number
"         LDI r19, 0b00000000       \n"//sets step number to 0
"         LDI r22, 0b00000000       \n"//sets flag number to 0
"         JMP LOOP                  \n" //jump loop
"returnright:                       \n"//it returns right from left edge
"         LDI r22, 0b00000001       \n" //set flag to 1
"         JMP sequenceoneright      \n" //jump to sequenceoneright
"returnleft:                        \n"//it returns left from right edge
"         LDI r22, 0b00000000       \n" //set flag to 0
"         JMP sequenceoneleft       \n" //jump to sequenceoneleft
```

```
"sequenceoneleft:               \n"//left transition of sequence one
"       CPI r20, 0b10000000     \n"//if LED is comes to left edge
"       BREQ returnright        \n"//branch to return right
"       INC r19                 \n"//increment step number
"       LSL r20                 \n" //left shift to pattern
"       OUT 0x0b, r20           \n"//gives output to LEDs
"       JMP LOOP                \n" //jump to loop
"sequenceoneright:              \n" //right transition of sequence one
"       CPI r20, 0b00000001     \n"//if LED is comes to right edge
"       BREQ returnleft         \n"//branch to return left
"       INC r19                 \n"//increment step number
"       LSR r20                 \n" //right shift to pattern
"       OUT 0x0b, r20           \n"//gives output to LEDs
"       JMP LOOP                \n" //jump to loop
"sequenceone:                   \n"//main label for sequence one
"       CPI r19, 0b00000000     \n"//if is is first step
"       BREQ sequenceonefirst   \n"//branch to sequenceonefirst
"       CPI r22, 0b00000000     \n"//flag check
"       BREQ sequenceoneleft    \n" //if it is 0, branch to sequenceoneleft
"       CPI r22, 0b00000001     \n"//flag check
"       BREQ sequenceoneright   \n"//if it is 1, branch to sequenceoneright
"sequenceonefirst:              \n"//label for initialize first pattern
"       INC r19                 \n"//increment step number
"       LDI r20, 0b00000001     \n" //initilaize
"       OUT 0x0b, r20           \n"//give output to LEDs
"       JMP LOOP                \n" //jump to loop
"start:                         \n" //Write your code
"       IN r17, 0x07            \n" //DDRC analog
"       ORI r17, 0b001111       \n"     //input output
"       OUT 0x07, r17           \n" //assign
"       IN r18, 0x04            \n" //DDRB left digital
"       ORI r18, 0b001111       \n" //input output
"       OUT 0x04, r18           \n" //assign
"       IN      r16,  0x0A ;    \n"//Get values of DDRD to r16
"       ORI     r16,  0b11111111\n" //Assign ones to define as an output
"       OUT  0x0A, r16          \n" //Give r16 values to DDRD
"       LDI r19, 0b00000000     \n" // pattern
```

```
"        LDI r20, 0b00000001      \n" // sequence counter
"        LDI r21, 0b00000101      \n" // delay counter initialized with 5
"        LDI r22, 0b00000000      \n" // flag
"        LDI r16, 0b10000000      \n" // temp
  );
}

void loop()
{
  asm(
"LOOP:                            \n"//LOOP function
";Your LOOP Code Begin            \n"
"        OUT 0x05,r21             \n"//out delay multiplier
"        OUT 0x08,r20             \n"//out pattern number
"        MOV r16, r21             \n"//move counter to temporarly register
"        INC r21                  \n"//increment temp register by one
"        CALL maindelay           \n"//call maindelay
"        IN r18, 0x03             \n" //get input from pinb
"        LSR r18                  \n"//right shift input register
"        LSR r18                  \n"
"        LSR r18                  \n"
"        LSR r18                  \n"
"        CPI r18, 0b00000001      \n" //compare if imput register is 1
"        BREQ negativesequence    \n"//if it is branch negativesequence
"        LSR r18                  \n"//right shift input register
"        CPI r18, 0b00000001      \n" //compare if input register is 1
"        BREQ plussequence        \n"//if it is branch plussequence
"        IN r17, 0x06             \n" //get input from pinc
"        LSR r17                  \n"//right shift input register
"        LSR r17                  \n"
"        LSR r17                  \n"
"        LSR r17                  \n"
"        CPI r17, 0b00000001      \n" //compare if input register is 1
"        BREQ plusdelay           \n"//if it is branch plusdelay
"        LSR r17                  \n"//right shift input register
"        CPI r17, 0b00000001      \n" //compare if input reigster is 1
"        BREQ negativedelay       \n"//if it is branch negativedelay
```

```
"        CPI r20, 0b00000001        \n" //if pattern is 1
"        BREQ sequenceone           \n" //branch to sequence one
"        JMP  LOOP                  \n"
  );
}
```

3 RESULTS

3.1 PART 1

After our code implementation, we have simulated the circuit and examined left shift of the LED.

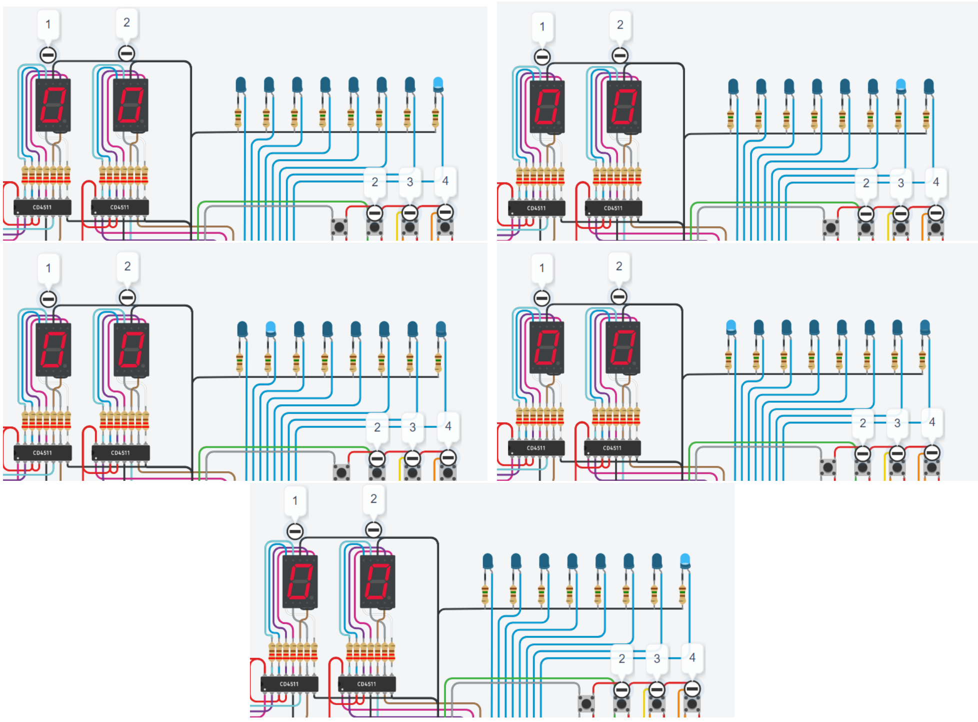

Figure 1: Left shift behaviour

3.2 PART 2

After our code implementation, we have simulated the circuit and examined our increment and decrement operation.

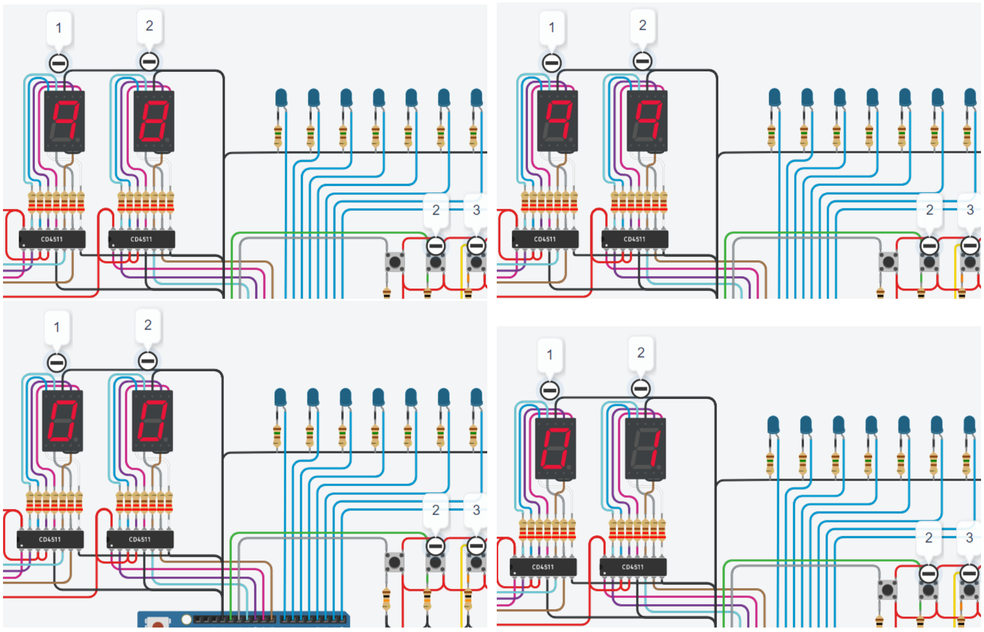

Figure 2: Up-counting

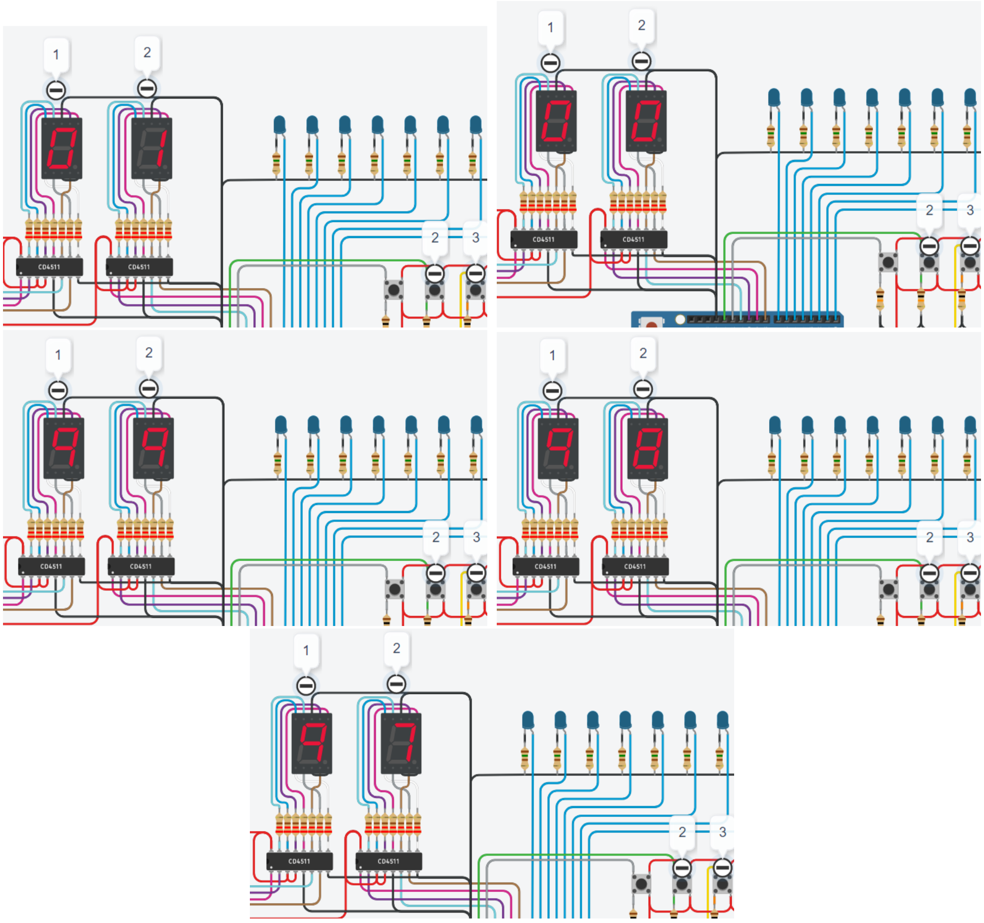

Figure 3: Down-counting

3.3 PART 3

After our code implementation, we have simulated the circuit. We can watch the working mechanism and implementation in the given YouTube link here.

3.4 PART 4

We have faced with errors, we have failed and we cannot run our code.

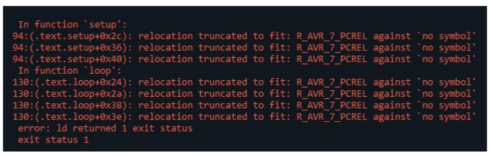

Figure 4: The error message that we get

4 DISCUSSION

4.1 PART 1

In this part of the experiment, we have implemented a predetermined pattern using only the LEDs. The pattern we are required to implement was implemented by us using C-like language of Arduino in previous weeks but now we had to use inline Assembly to achieve our goal. After checking some tutorials and documentation to grasp inline Assembly features of Arduino the algorithm became a little easier to implement. In fact, we have only used a right circular shift subroutine. Since this experiment was our first encounter with Assembly language we have only faced some syntactic and logical issues at first tries.

4.2 PART 2

In this part of the experiment, we have implemented an up-down counter by only using two seven segment displays and two up-down counting buttons. The circuit uses button 1 to increment and button 2 to decrement with the restriction of only chancing it's value when the is the clicked.

The circuit must not change its value when buttons are continuously pressed or not pressed. The first difficulty we have faced was to provide counter its circular counting effect. In order to provide this effect we have created two subroutines called "tozero" and "tonintynine" when the counter reaches its end values.

After bringing in the circular counting ability to our circuitry second challenge we have faced was to make circuit change its value only when the button is clicked. To achieve that we have spend hours with trying different method but finally we have come up with a solution. We checked whether the input on buttons changes or not. Then, we created our own delay routine called "calldelay" but opposing to other delay function given to us our function goes back at the starting of the loop after delaying. This way we have made circuit count only when button is clicked. This part was quite challenging since we had no background of Assembly but finally we have made our circuit perform the required operation after many hours of discussion and work.

4.3 PART 3

In this part of the experiment, we have implemented an up-counter with the ability of dynamic counting and timing. We first two buttons are used to control the increment number. The right-end two buttons are used to control the delay time between numbers. In addition to displaying number using seven segment displays, we were also required display it in binary form using LEDs.

Since we were required to use all the elements in our circuitry in this part of the experiment, we had hard times with this part in general. First, we have realized we can build this circuitry by editing the Assembly code of part 2. We have edited and cropped some of our code from part 2 and designed some subroutines to iterate the increment function that we have designed in part 2. After many hours of work and complicated Assembly code implementation we have achieved our goal. Our circuit had the ability to change its increment value and delay value by clicking the buttons.

Final difficulty we have faced was to display the number in terms of binary. We have realized we had the tens and ones digit in our registers so all we had to was to change these digits into a full integer form to represent as binary. For decimal conversion tens digit must be multiplied by 10. We have checked the commands we can use in Assembly then we have realized that there exists a command to perform multiplication operation between registers. We have added our code a few lines accordingly and finally we have made our circuit represent the number in binary. Our searches made us achieve our goal but this part was the hardest part for us to implement in terms of time and complexity even though we have tried to do our best.

4.4 PART 4

In this part, we are expected to implement a system that can be controlled by buttons to change pattern and period of the LED transition. When we think like a software programming, we know that it is easy to implement. But in hardware programming, it is too difficult to handle all operations. We tried to implement it but we have faced with lots of problems. Arduino allows us to use just r16-r25 and most of the registers must be used for input and output operation. Delay label also used 3 of the registers. Therefore, we have to implement all system with limited storage capacity that is 5 registers. It was hard to handle all operations with these registers.

This part also needs to many comparison operations to determine our step and pattern number. When we used many branches and jumps into our assembly code, assembler gives errors that states "relocation truncated to fit". We search this message in Internet but we cannot find a way to solve this problem. We found that it occurs because of jump usage.

This part is hard to implement for people who did not know assembly. It would be more beneficial, if we learn how to write assembly code and handle Arduino registers by a teacher. We think that we have not enough knowledge about assembly and this part of experiment further reduces our self-confidence. In essence, we tried our best, gave lots of time but we could not be successful. We hope that this will be the last experiment part that we cannot run.

5 CONCLUSION

Achievement of this week's experiment is to teach fundamentals of the Arduino circuitry with Arduino Avr Assembly language. Reading inputs from buttons was a hard task to do at first and arranging delay timings was hard. At first we did not know anything about Assembly language we had to improvise and learn some new things on our own. We come up some different solutions for each part of the experiment, but some of them work some of them did not. We think we did our best to finish this experiment with in given time. Even we did not complete all of the parts we try to complete as best as we can.

References

[1] Kadir Ozlem. *BLG 351E – Microcomputer Laboratory Slides*. 2020.

[2] Arduino. Arduino documentation. https://www.arduino.cc/reference/en/, 2018.

[3] Autodesk. Tinkercad. https://www.tinkercad.com/, 2011.