



كلية الهندسة و التكنولوجيا

College of Engineering and Technology



Electrical and Control Engineering Department

---

# Microprocessor Based Control Laboratory Experiments Handbook

Name : \_\_\_\_\_

Reg. No. : \_\_\_\_\_

# **Table of Contents**

## Part 1- Op-Amp. Circuit Design :

- Experiment No.1 – Power supply
- Experiment No.2 – Inverting , Non - Inverting Amp. , Difference Amp.
- Experiment No.3 – Summing Amp. , Current to Voltage Converter , Buffer
- Experiment No.4 – Cases

## Part 2- Open Loop System :

### Introduction to AVR MicroController :

- Digital Input / Output
  - Experiment No.5 – Flasher Circuit
  - Experiment No.6 – Push button - LED Control
  - Experiment No.7 – 7 Segments Interface
  - Experiment No.8 – LCD Interface
  - Experiment No.9 – Keypad and LCD
- Analog Input / Output
  - Experiment No.10 – Analogue to Digital Conversion

## Part 3- Closed Loop Control:

- Experiment No. 11 -- AVR Microcontroller Interrupts
- Laboratory Experiment No. 12 -- Timer & Timer Interrupt

## Part 4- Arduino:









## **Experiment No.1 (Handling simulation and design)**

### **Power Supply**

#### **Objective:**

The objective of the experiment is to learn how to design an electrical circuit and to implement it practically on bread board and on a printed circuit board (PCB) using all the tools needed.






#### **PCB equipment:**

Equipment	Quantity	Picture
Soldering iron	1	
Solder	1	
Printed circuit board	2	
Aluminum solder sucker	1	
PCB's Acid	1	
PCB's Driller	1	
Glossy paper	2	
Plastic bucket	1	

Name : \_\_\_\_\_

Reg. No. : \_\_\_\_\_

**Power supply equipment:**

Equipment	Quantity	Picture
Transformer 220V/12V 0 12V 1A Or 220V/6V 0 6V 1A	1	
Regulator 7805 (5V) 1A	1	
Regulator 7812 (12V) 1A	1	
Capacitor 100 $\mu$ F 25V	2	
Capacitor 1000 $\mu$ F 50V	2	

**Simulation and PCB designing software (recommended):**

- 1- Proteus PCB Design.
- 2- CadSoft EAGLE PCB Design.

**Extra simulation software:**

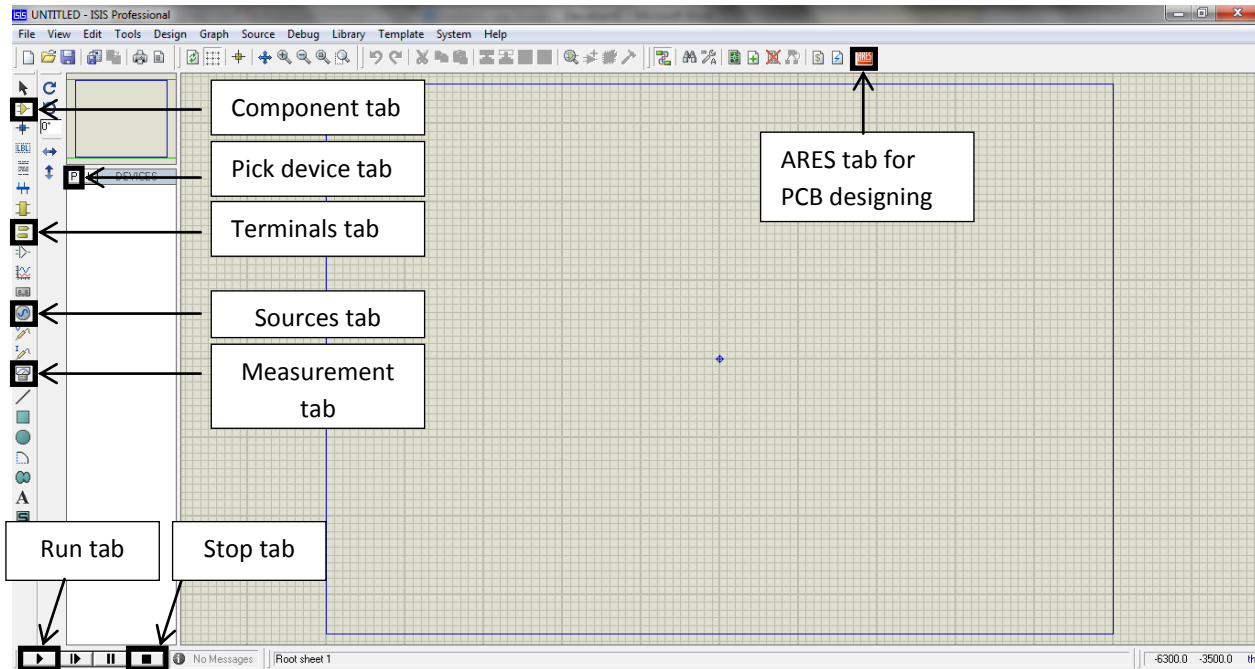
- 1- Matlab.
- 2- Pspice.
- 3- Multisim.

**Power supply implementation:**

This program is used for simulating electric circuits with components that are selected from the real life. The advantage of this program that any electrical circuit can be simulated and tested with no hardware cost and no hardware failure and to be sure that this design is ready to be practically implemented. This program can be also used for PCB design not only for simulation.

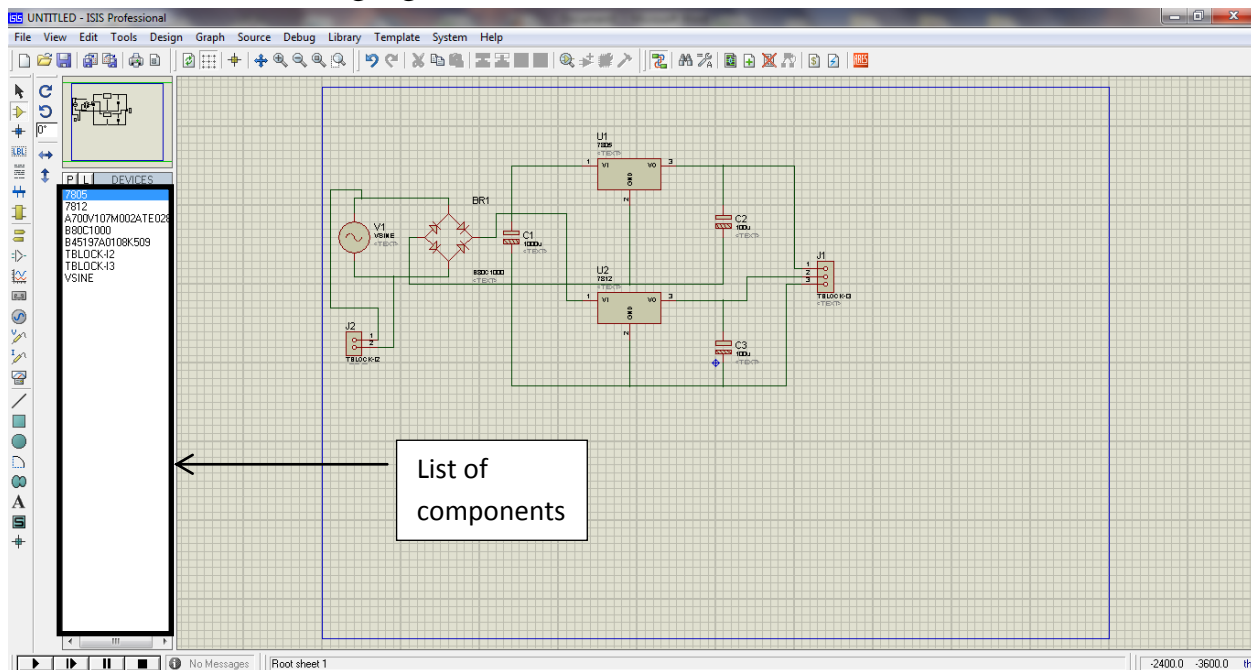
## Procedures:

1- Open the ISIS .exe file to open the simulator and this is the main page.



2- Use the pick device tab to choose the components relevant to the desired design.

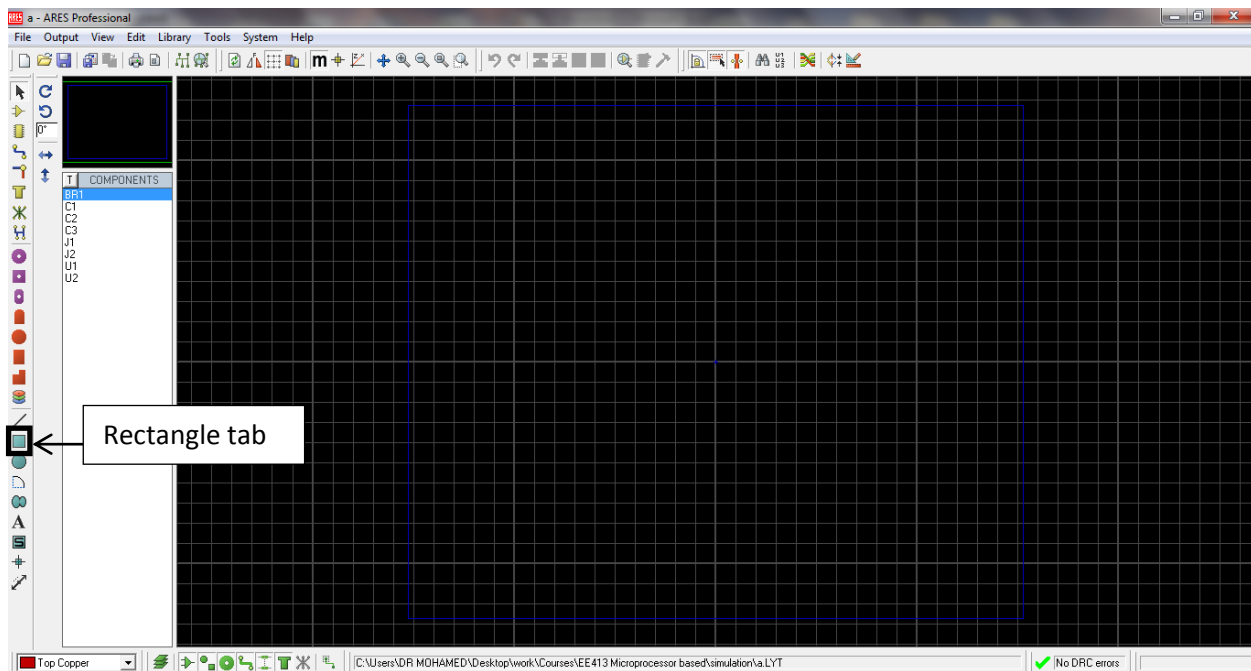
3- After finishing the design click on the run tab to start simulating the circuit it should be like the following figure.



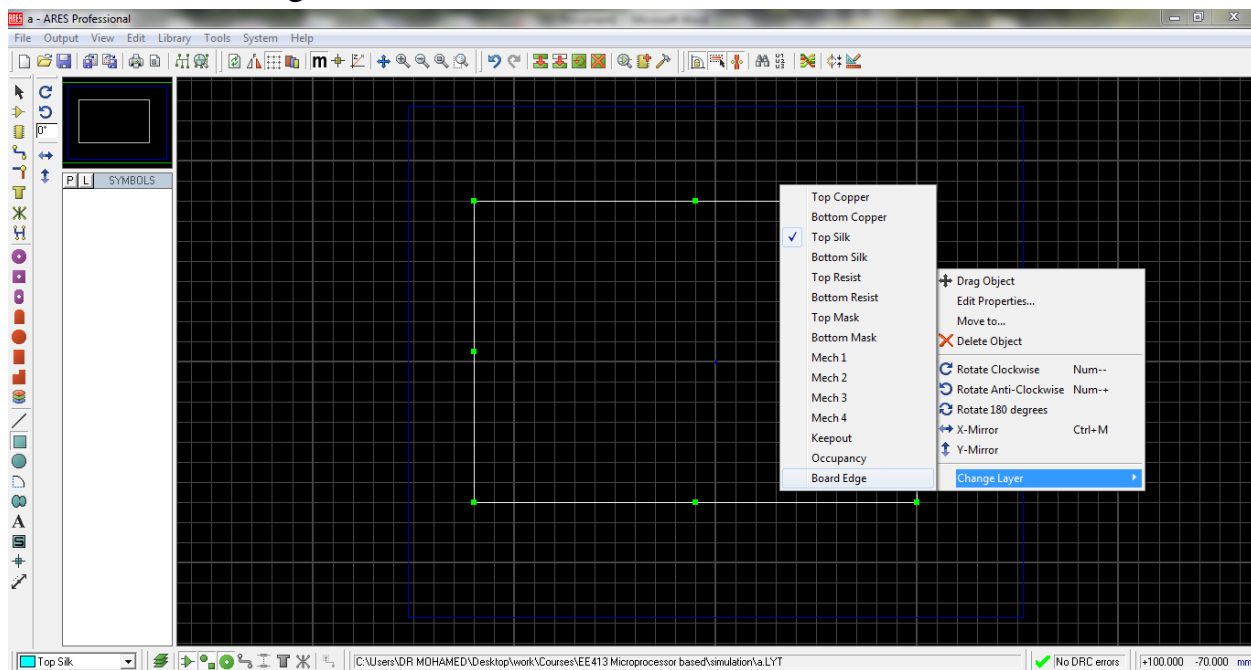
4- The ARES tab can be used to convert the components of the design to be printed on a board then the following window will open.

Name : \_\_\_\_\_

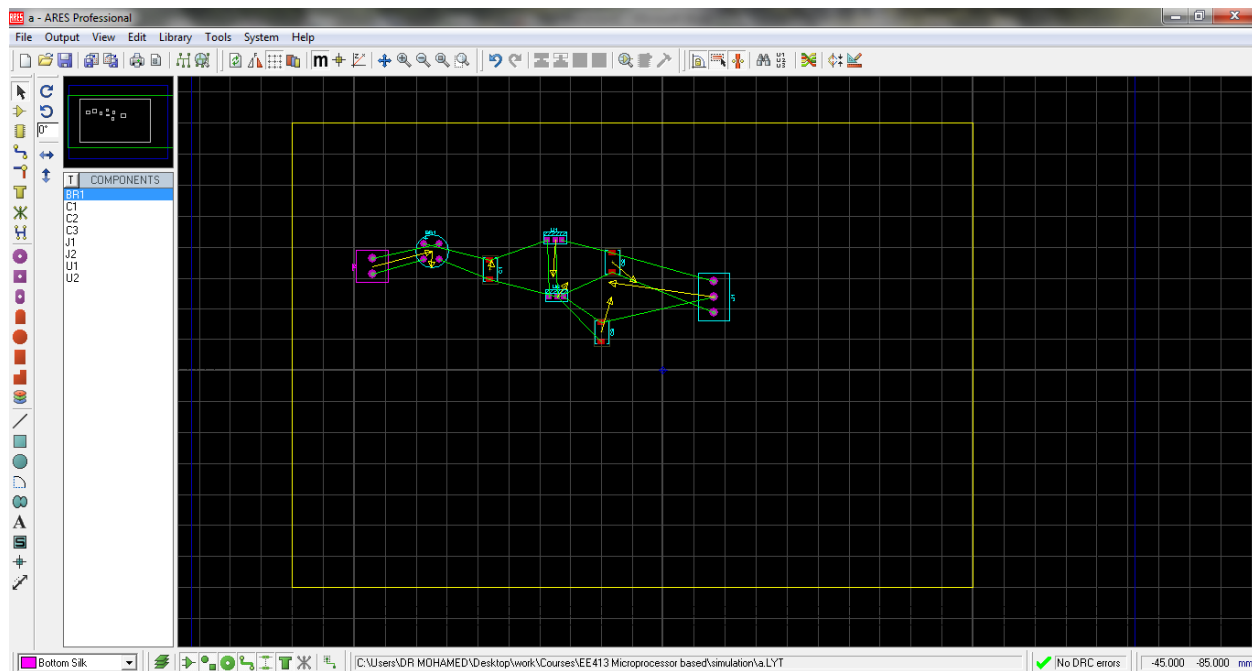
Reg. No. : \_\_\_\_\_



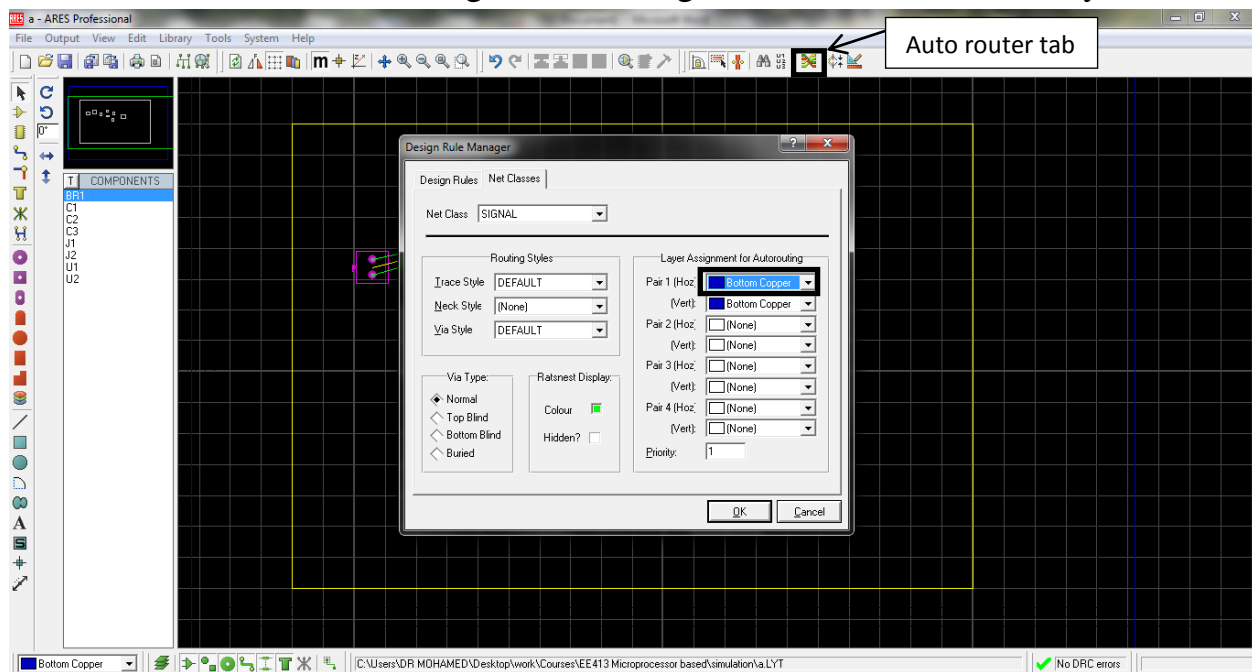
### 5- Draw rectangle to be the border of the PCB.



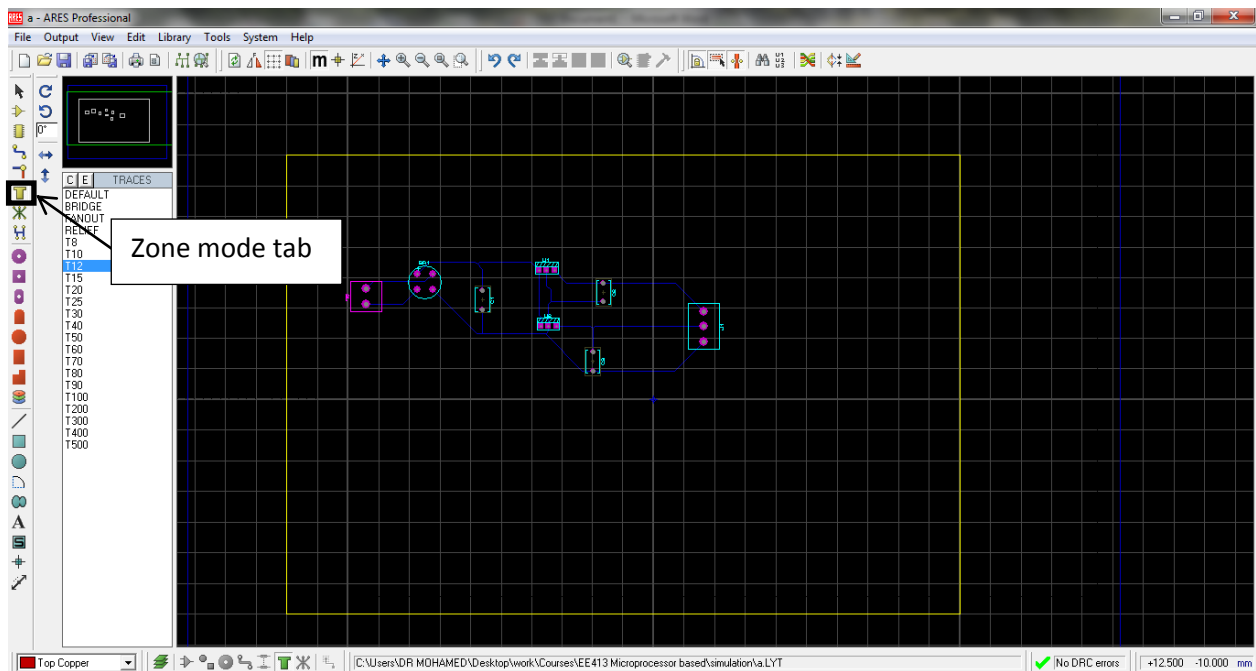
6- Choose this rectangle to be the board edge by a right click on it, then click on tools and choose Auto Placer to put the components inside the rectangle and after a modification of the components it should be as follows.



7- Click on tools then Design Rule Manager and choose to be one layer.



8- Click on the Auto-router and the paths will be as follows.



- 9- The zone mode tab is used at the end to draw a rectangle of copper around the area needed. Finally is to print it using a laser printer and start the PCB process.

### **Answer the following questions:**

- 1- What are the components needed to be changed in order to increase the ampere capacity of the whole circuit?  
.....  
.....  
.....
- 2- What happens by changing the values of the capacitors before and after the regulators?  
.....  
.....  
.....
- 3- Does the transformer used at the beginning can be substituted by 220V/6V and why?  
.....  
.....  
.....



- 4- What is the function of the zone mode tab at the end of the PCB design procedures and why?

.....  
.....  
.....

### **Bonus Task:**

- Make a presentation about the Cadsoft Eagle and how it can be used for PCB design.

## Experiment no. 2 (Handling simulation)

### Amplifiers

### 1 Inverting Amplifier :

#### 1.1 Objective:

The purpose of this experiment is to apply the Op-Amp circuit analysis & obtain the output voltage almost identical to the pre-proved transfer function .

#### 1.2 Procedures:

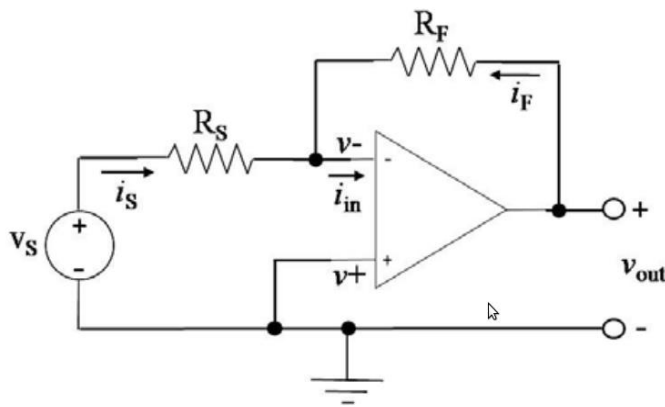


Figure 1

Component	Value
$R_f$	30 K $\Omega$
$R_s$	10 K $\Omega$
$V_s$	5 V

- 1- Connect the circuit as shown in figure ( with +ve & -ve Supply Voltage).
- 2- Add resistance value with a suitable gain ( Measure using OhmMeter).
- 3- Measure the output voltage .
- 4- Compare the practical results with those obtained from calculations .
- 5- Comment on the results.

.....

.....

.....

.....

### 1.3 Answer the following questions :

- 1- The output voltage should be \_\_\_\_\_ .
- 2- The theoretical Gain is \_\_\_\_\_ .
- 3- The actual Gain is \_\_\_\_\_ .

## 2 Non-Inverting Amplifier :

### 2.1 Objective:

The purpose of this experiment is to apply the Op-Amp circuit analysis & obtain the output voltage almost identical to the pre-proved transfer function .

### 2.2 Procedures:

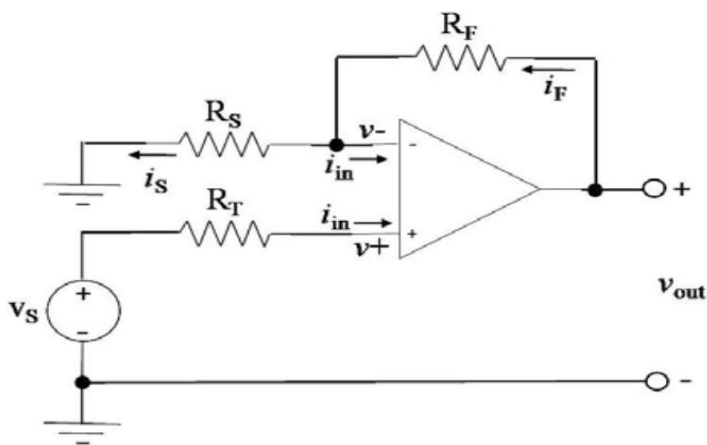


Figure 2

Component	Value
$R_F$	30 K $\Omega$
$R_S=R_T$	10 K $\Omega$
$V_s$	5 V

- 1- Connect the circuit as shown in figure (with +ve & -ve Supply Voltage).
- 2- Add resistance value with a suitable gain (Measure using Ohm Meter).
- 3- Measure the output voltage.
- 4- Compare the practical results with those obtained from calculations.
- 5- Comment on the results.

.....

.....

## 2.3 Answer the following questions :

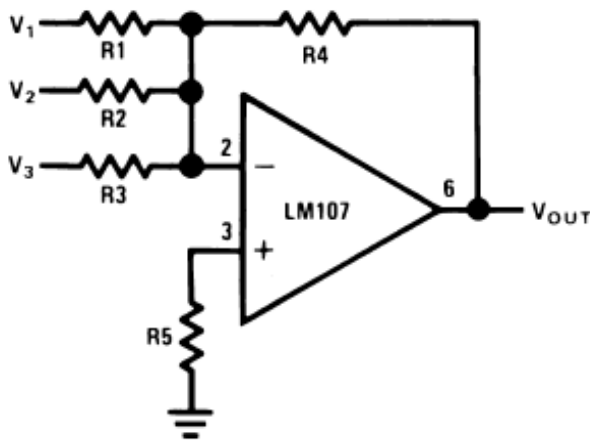
- 1- The output voltage should be \_\_\_\_\_.
- 2- The theoretical Gain is \_\_\_\_\_.
- 3- The actual Gain is \_\_\_\_\_.
- 4- Write down the Transfer Function \_\_\_\_\_.

## 3 Summing Amplifier:

### 3.1 Objective:

The purpose of this experiment is to apply the Op-Amp circuit analysis & obtain the output voltage almost identical to the pre-proved transfer function.

### 3.2 Procedures:



Component	Value
<b>R4(Gain)</b>	10 K $\Omega$
<b>R1=R2=R3</b>	10 K $\Omega$
<b>V1</b>	5 V
<b>V2</b>	7V

Figure 3

- 1- Connect the circuit as shown in figure (with +ve & -ve Supply Voltage).
- 2- Select resistance values so that the gain is unity (Measure using Ohm Meter).
- 3- Measure the output voltage.
- 4- Compare the practical results with those obtained from calculations.
- 5- Comment on the results.

.....

.....

.....

.....

### 3.3 Answer the following questions:

- 1- The output voltage should be \_\_\_\_\_.
- 2- The difference between actual& practical results is \_\_\_\_\_.
- 3- The actual Gain is \_\_\_\_\_.
- 4- Write down the Transfer Function \_\_\_\_\_.

### Boards layout:

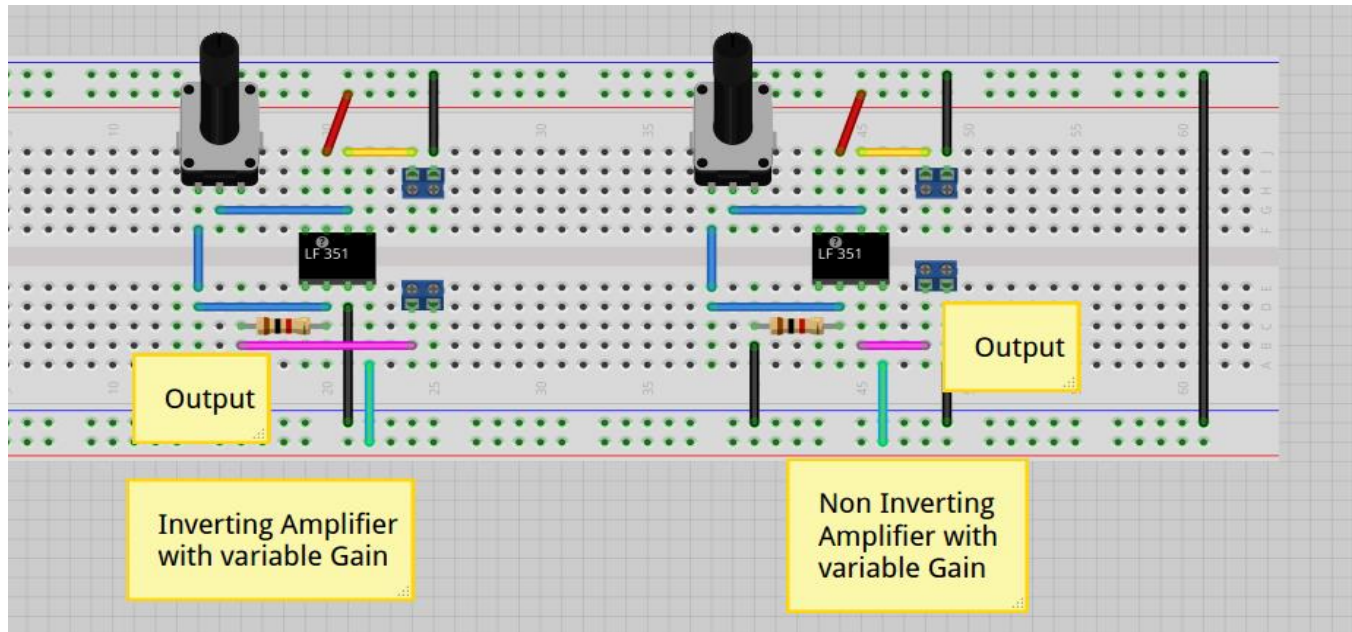


Figure 4

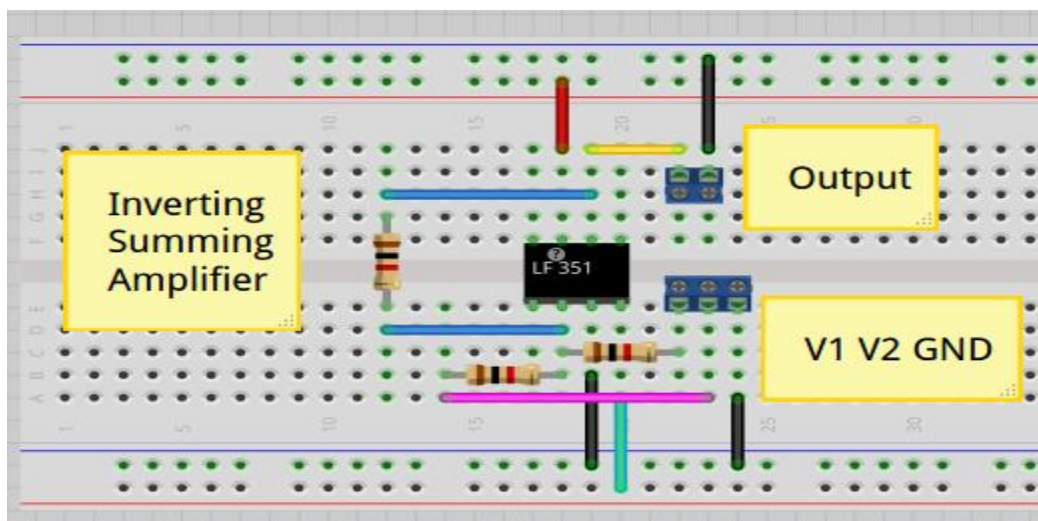


Figure 5

## Experiment no. 3 (Handling Simulation)

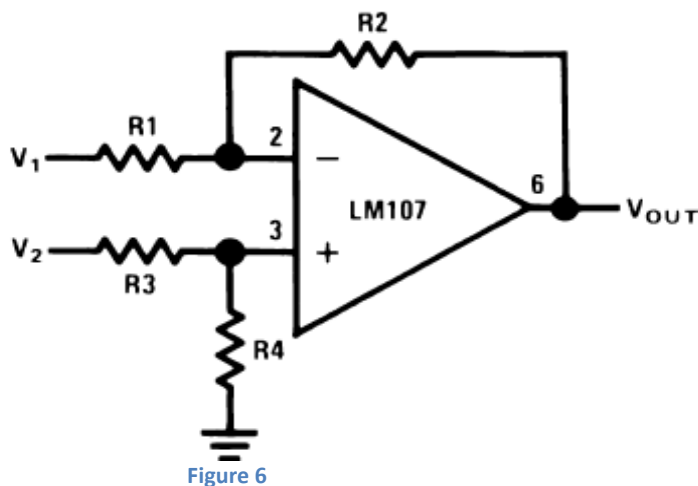
### Amplifiers

### 1 Difference Amplifier:

#### 1.1 Objective:

The purpose of this experiment is to apply the Op-Amp circuit analysis & obtain the output voltage almost identical to the pre-proved transfer function.

#### 1.2 Procedures:



Component	Value
<b>R1=R3</b>	10 K $\Omega$
<b>R2=R4</b>	10 K $\Omega$
<b>V1</b>	5 V
<b>V2</b>	10V

- 1- Connect the circuit as shown in figure (with +ve & -ve Supply Voltage).
- 2- Select the resistance value (Measure using Ohm Meter to get accurate results).
- 3- Measure the output voltage.
- 4- Compare the practical results with those obtained from calculations.
- 5- Comment on the results.

.....

.....

.....

.....

### **1.3 Answer the following questions :**

- 1- The output voltage should be \_\_\_\_\_.
- 2- The theoretical Calculation is \_\_\_\_\_.
- 3- The Transfer Function is \_\_\_\_\_.

## **2 Current to voltage converter:**

### **2.1 Objective:**

The purpose of this experiment is to apply the Op-Amp circuit analysis & obtain the output voltage almost identical to the pre-proved transfer function.

### **2.2 Procedures:**

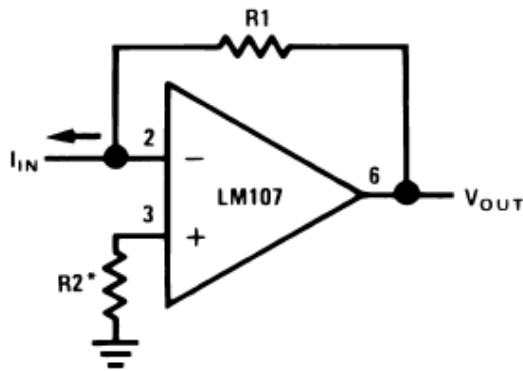


Figure 7

- 1- Connect the circuit as shown in figure (with +ve & -ve Supply Voltage).
- 2- Add resistance value for a suitable output voltage (Measure using Ohm Meter).
- 3- Measure the output voltage.
- 4- Compare the practical results with those obtained from calculations.
- 5- Comment on the results.

.....

.....

.....

### **2.3 Answer the following questions:**

- 1- The output voltage is \_\_\_\_\_.
- 2- The input current is \_\_\_\_\_.
- 3- Write down the Transfer Function \_\_\_\_\_.

### **3 Buffer (Voltage follower) Amplifier:**

#### **3.1 Objective:**

The purpose of this experiment is to apply the Op-Amp circuit analysis & obtain the output voltage almost identical to the pre-proved transfer function.

#### **3.2 Procedures:**

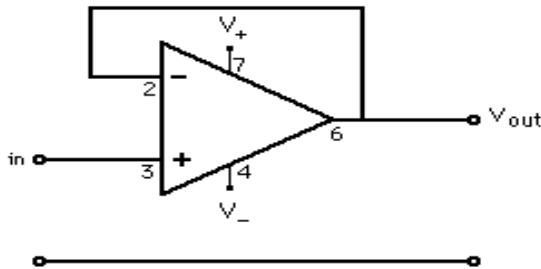


Figure 8

- 1- Connect the circuit as shown in figure (with +ve & -ve Supply Voltage).
- 2- Measure the output voltage.
- 3- Compare the practical results with those obtained from calculations.
- 4- Comment on the results.

.....

.....

.....

#### **3.3 Answer the following questions :**

- 1- The output voltage should be \_\_\_\_\_.
- 2- The difference between actual& practical results is \_\_\_\_\_.
- 3- The actual Gain is \_\_\_\_\_.
- 4- Write down the Transfer Function \_\_\_\_\_.



## Board layout :

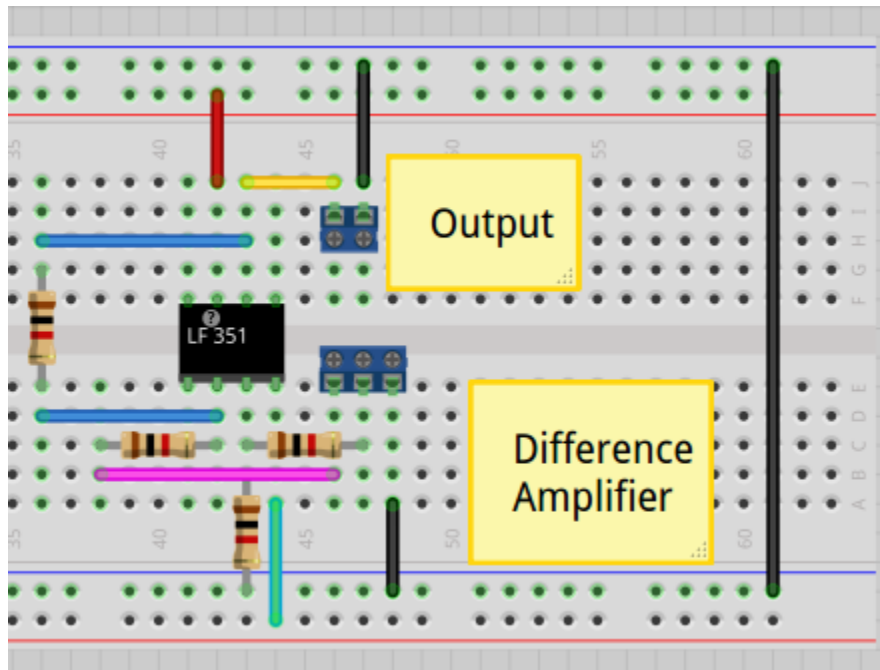


Figure 9

## Experiment no.4

### Four cases

A linear op amp transfer function is limited to the equation of a straight line

Equation:  $y = \pm mx \pm b$ .

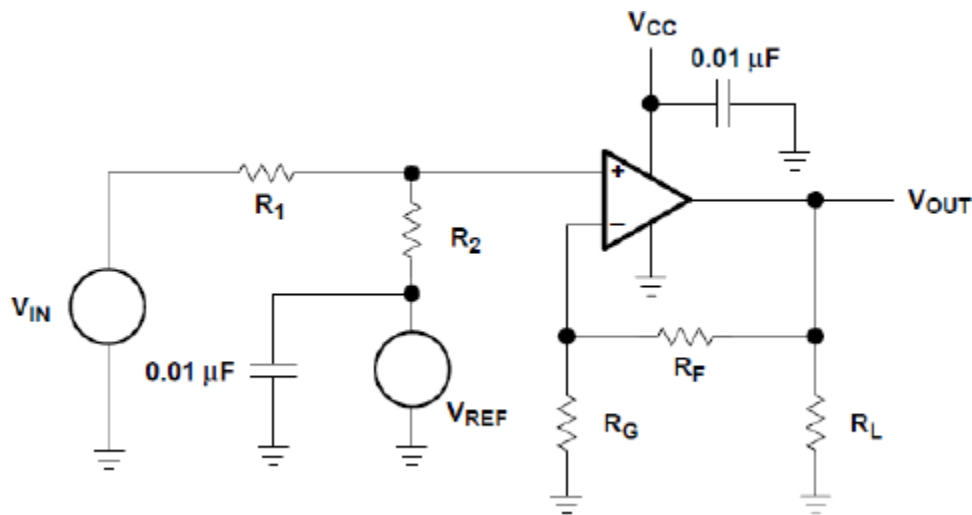
The equation of a straight line has four possible solutions depending upon the sign of m, the slope, and b, the intercept; thus simultaneous equations yield solutions in four forms.

Four circuits must be developed; one for each form of the equation of a straight line.

#### First case:

**Main equation:**  $V_{out} = + mV_{in} + b$

**Circuit diagram:**

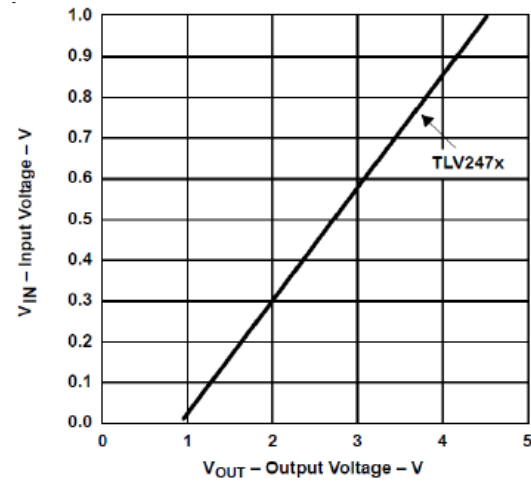
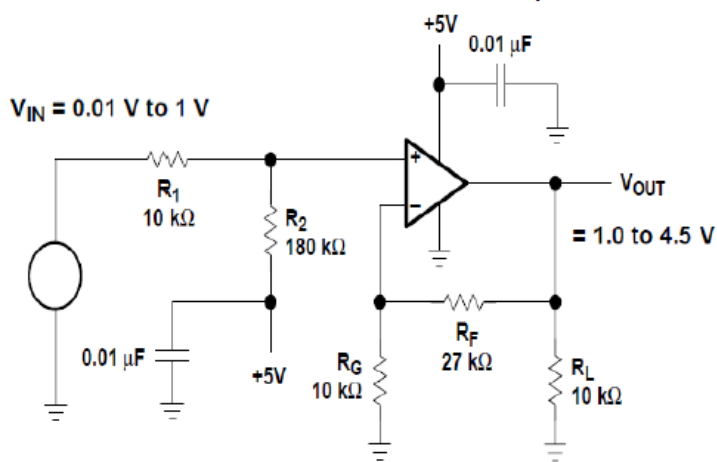


**Output Function:**  $V_{OUT} = V_{IN} \left( \frac{R_2}{R_1 + R_2} \right) \left( \frac{R_F + R_G}{R_G} \right) + V_{REF} \left( \frac{R_1}{R_1 + R_2} \right) \left( \frac{R_F + R_G}{R_G} \right)$

**Design:**

For  $V_{out} = 1\text{ V}$  at  $V_{in} = 0.01\text{ V}$ ,

$V_{out} = 4.5\text{ V}$  at  $V_{in} = 1\text{ V}$

**Comment:**

.....

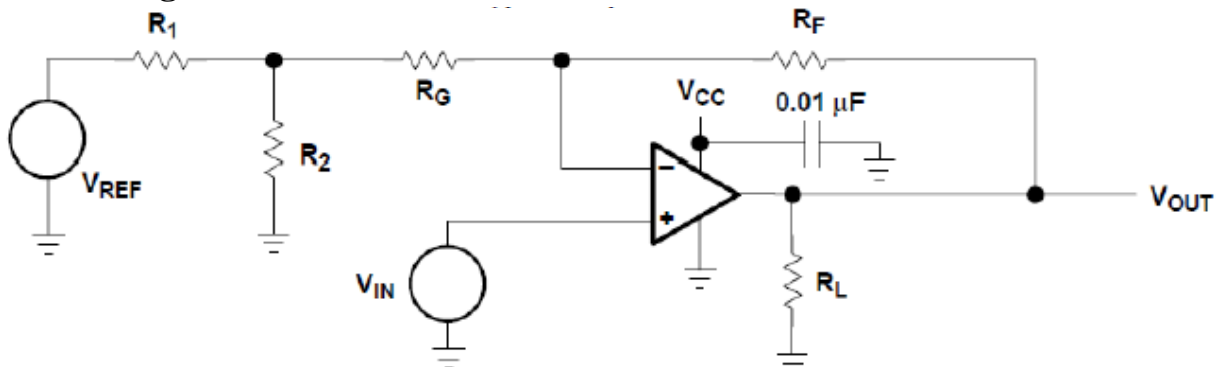
.....

.....

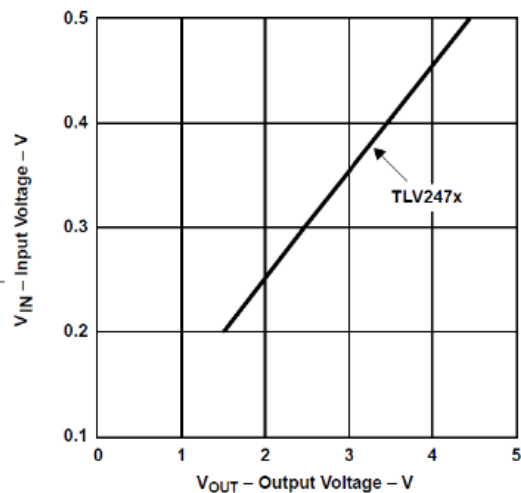
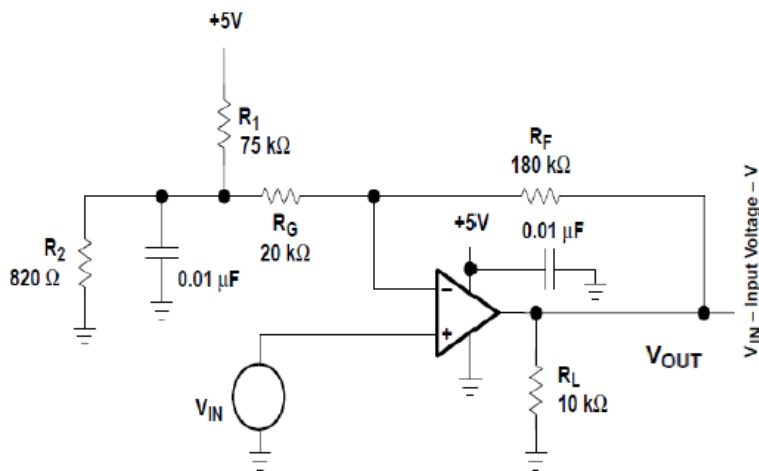
.....

Name : \_\_\_\_\_

Reg. No. : \_\_\_\_\_

**Second case:****Main Equation:**  $V_{out} = + mV_{in} - b$ **Circuit diagram:**

**Output Function:**  $V_{OUT} = V_{IN} \left( \frac{R_F + R_G + R_1 \parallel R_2}{R_G + R_1 \parallel R_2} \right) - V_{REF} \left( \frac{R_2}{R_1 + R_2} \right) \left( \frac{R_F}{R_G + R_1 \parallel R_2} \right)$

**Design:**For  $V_{out} = 1.5 \text{ V}$  at  $V_{in} = 0.2 \text{ V}$ , $V_{out} = 4.5 \text{ V}$  at  $V_{in} = 0.5 \text{ V}$ **Comment:**

.....

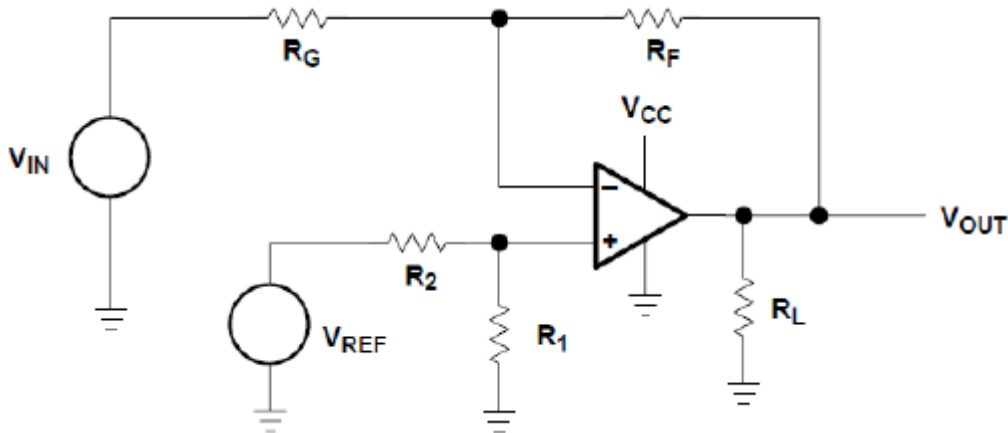
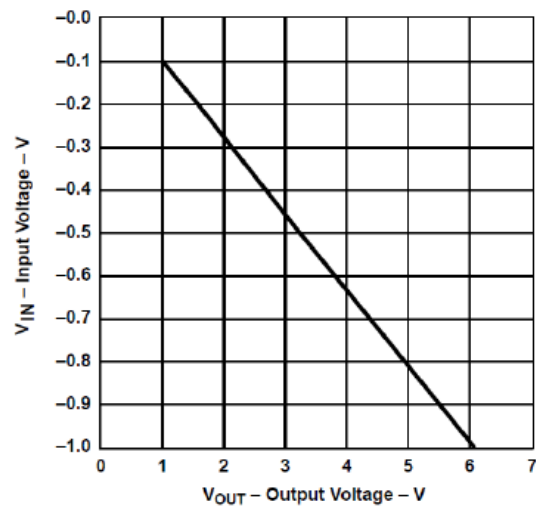
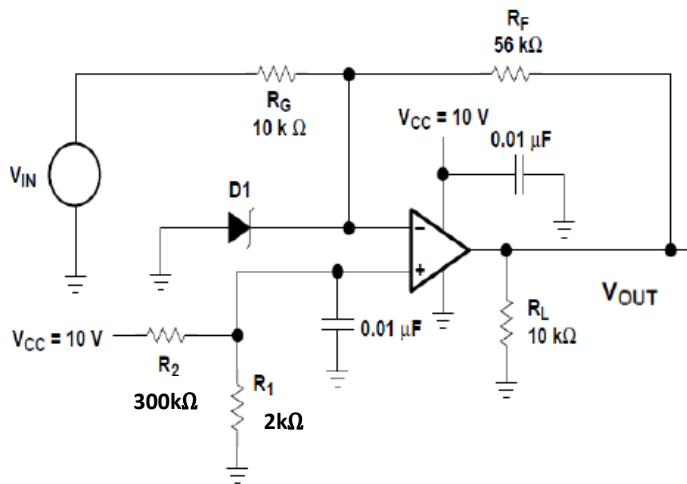
.....

.....

.....

Name : \_\_\_\_\_

Reg. No. : \_\_\_\_\_

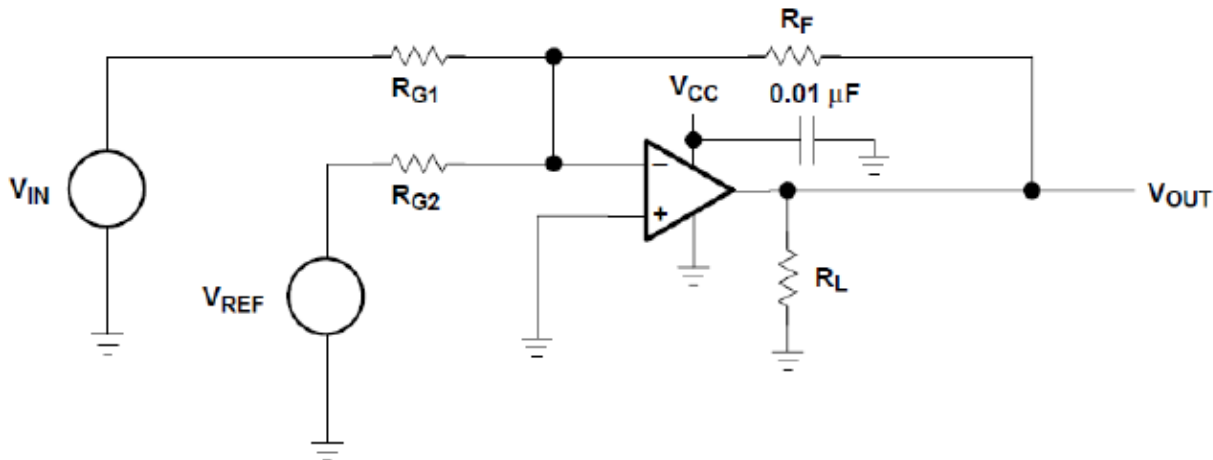
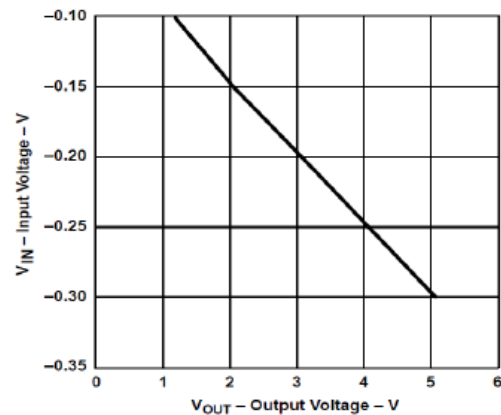
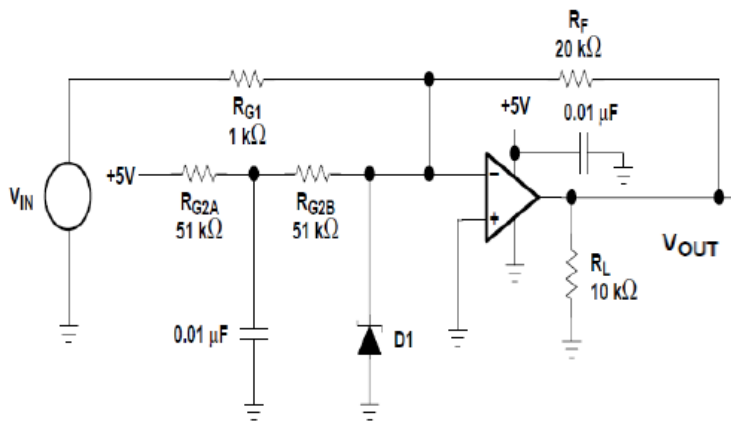
**Third case:****Main equation:**  $V_{OUT} = -mV_{IN} + b$ **Circuit diagram:****Output Function:**  $V_{OUT} = -V_{IN} \left( \frac{R_F}{R_G} \right) + V_{REF} \left( \frac{R_1}{R_1 + R_2} \right) \left( \frac{R_F + R_G}{R_G} \right)$ **Design:**For  $V_{OUT} = 1 \text{ V}$  at  $V_{IN} = -0.1 \text{ V}$ , $V_{OUT} = 6 \text{ V}$  at  $V_{IN} = -1 \text{ V}$ **Comment:**

.....

.....

.....

.....

**Fourth case:****Main Equation:**  $V_{out} = -mV_{in} - b$ **Circuit diagram:****Output Function:**  $V_{OUT} = -V_{IN} \frac{R_F}{R_{G1}} - V_{REF} \frac{R_F}{R_{G2}}$ **Design:**For  $V_{out} = 1 \text{ V}$  at  $V_{in} = -0.1 \text{ V}$ , $V_{out} = 5 \text{ V}$  at  $V_{in} = -0.3 \text{ V}$ **Comment:**

.....

.....

.....

.....

## Laboratory Experiment No. 5 (Handling outputs)

### Flasher Circuit

#### Objective:

To make a flasher circuit using the ATmega32 microcontroller

#### Instrument and equipment:

Equipment	Quantity
ATmega32 or ATmega16	1
Led	1
Resistor 330 $\Omega$	1

#### Connection:

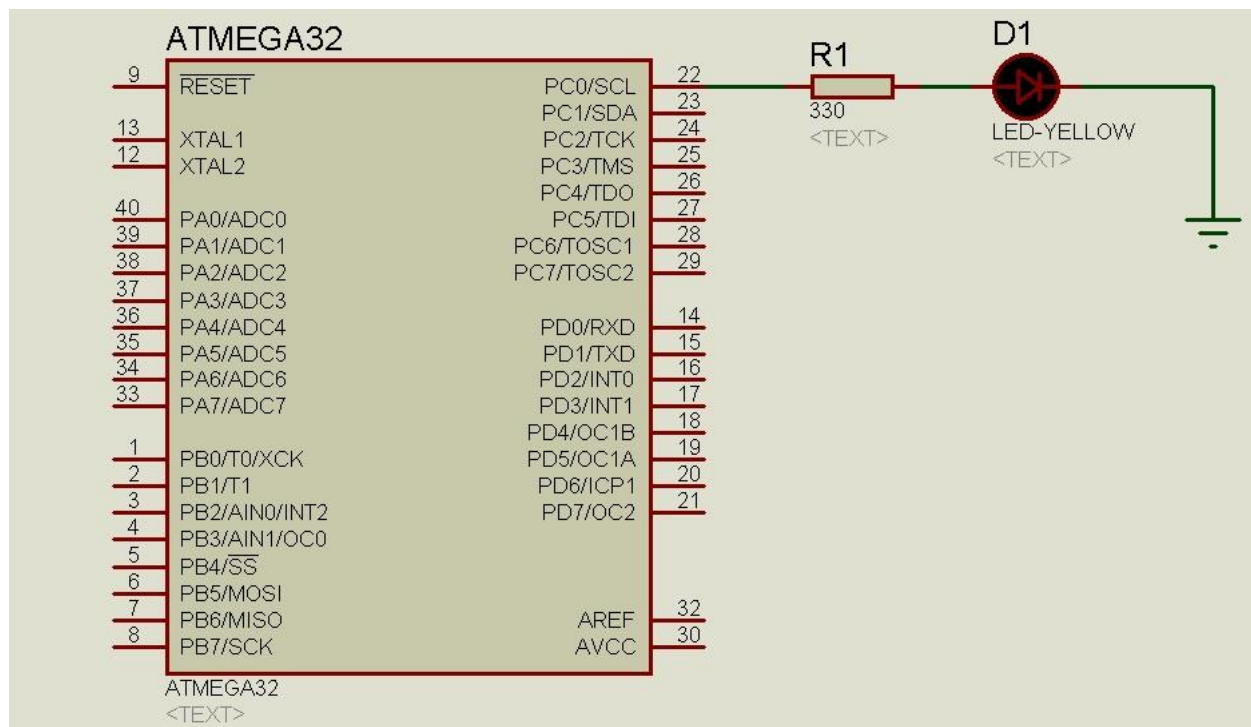
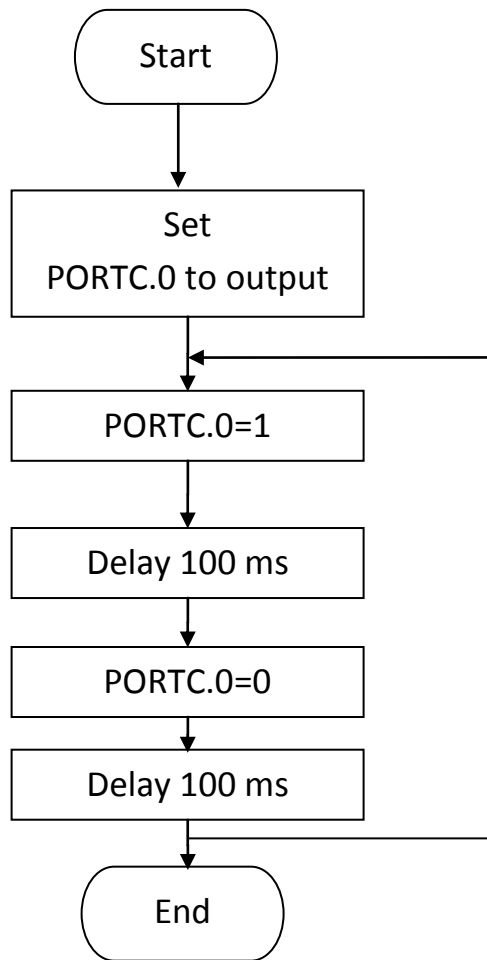


Figure 10

#### Procedures:

- 1- Connect the ATmega32 to a source of power 5v.
- 2- Connect the Led cathode pin to any port in the microcontroller and the anode to resistor 330  $\Omega$  to ground.
- 3- Make the code
- 4- Download the code on the microcontroller



```
#include < atmega16.h >
#include < delay.h >
Void main ( )
{
    DDRC=0x01;
    while (1)
    {
        PORTC.0 = 1;
        delay_ms(100);
        PORTC.0 = 0;
        delay_ms(100);
    };
}
```

## Conclusion

In this exercise you have learned how the ATMEGA16 can be used as a digital output device.

## Lab assignments

Change your circuit connection and add another 7 LEDs to make a flasher series code which make the LEDs on and off with sequence.



## Laboratory Experiment No. 6 (Handling inputs)

### Micro-Controller as Input Device

#### Objective

After completing this exercise you will be able to Toggle the output (LEDs) using push buttons.

#### Instrument and equipment:

Equipment	Quantity
ATmega32 or ATmega16	1
Led	8
Resistor 330 $\Omega$	8
Push buttons	8
Resistors 10 k $\Omega$	8

#### Procedure

##### Phase-1 building your circuit

1- Use the circuit you built in exercise1 and modify it as shown in figure

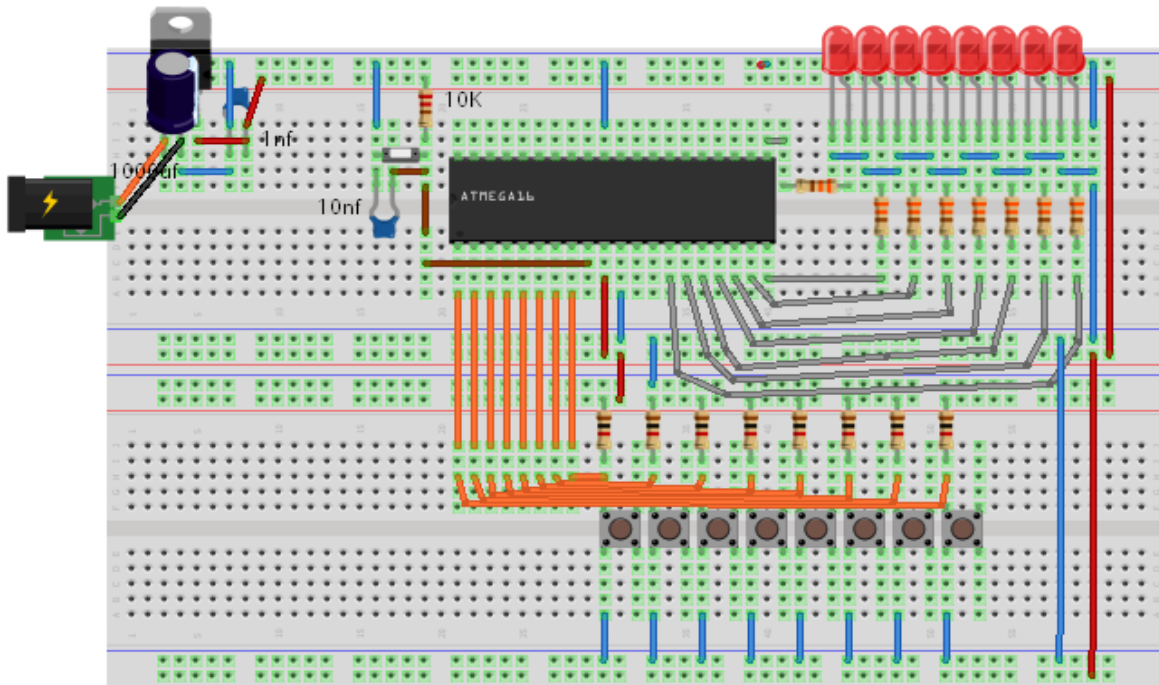
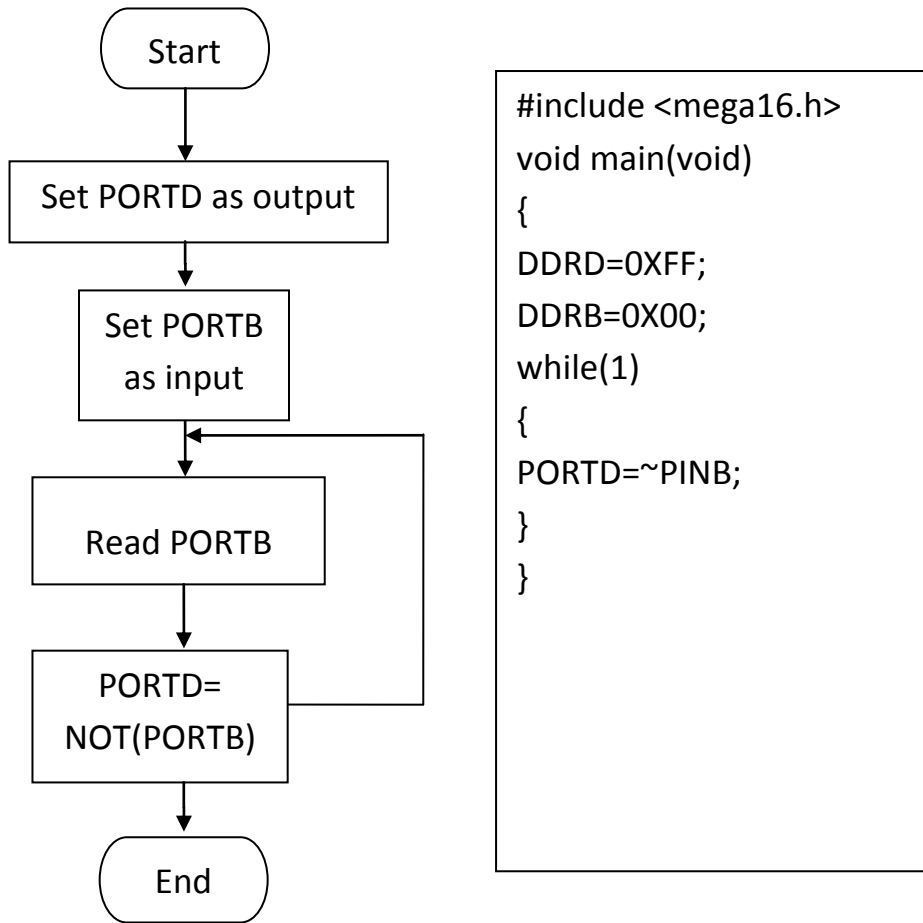


Figure 11

1- Use CodeVision software to write the following program



2- From your observation of the circuit what happen when you press any pushbutton?

.....

3- According to you connection of the pushbuttons what is the logic input to the microcontroller when a button is pressed?

.....

4- Why is  $PORTD = \text{NOT}(PORTB)$ ?

.....

## **Conclusion**

In this exercise you have learned how to connect input devices To the ATMEGA16 and configure the microcontroller as a simple output device.

## **Lab assignments**

Change your circuit connection and program so that the logic input is high when a button is pressed.

## **Laboratory Experiment No. 7 (Handling outputs)**

### **7 segment display**

#### **Objective**

When completing this unit you will be able to connect a 7-segment and LCD to your AVR microcontroller.

#### **Lab Equipment**

Equipment	Quantity
ATmega32 or ATmega16	1
7 Segment common anode	1
IC 7447	1

#### **Discussion**

A seven-segment display (SSD), or seven-segment indicator, is a form of electronic display device for displaying decimal numerals. Seven-segment displays are widely used in digital clocks, electronic meters, and other electronic devices for displaying numerical information.

Figure 12 shows a typical seven segment display, as its name indicates, is composed of seven elements. Individually on or off, they can be combined to produce simplified representations of the Arabic numerals.

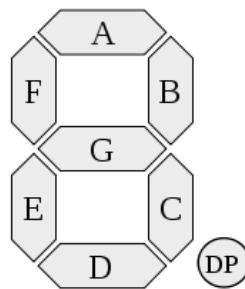


Figure 12

There are two main types of 7-segment displays, the Common anode display and the common cathode display. As the name indicates a common anode consists of 8 LEDs each corresponding to a given segment with their positive terminal connected as common.

The common cathode implies the same idea with the exception that the negative terminals are connected as common. Figure 13 shows the basic structure of a common anode and cathode 7-segments.

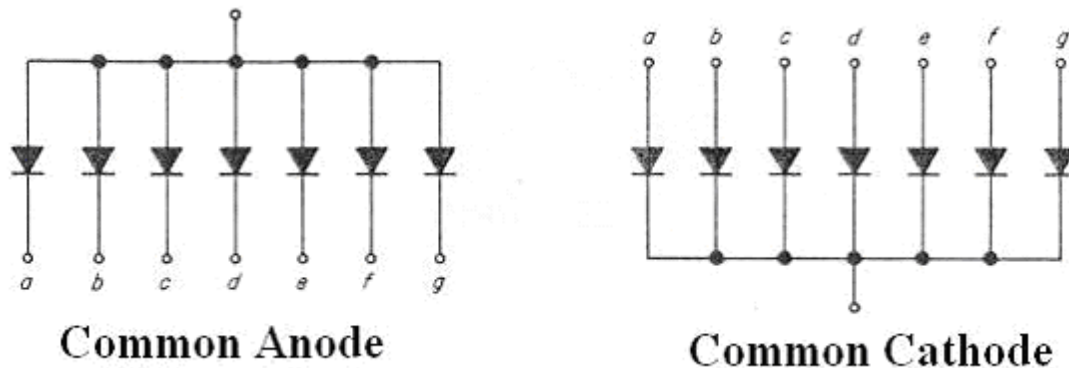


Figure 13

It is obvious from Figure 2-2 that the common Anode requires an input of 0 to switch any segment to on and an input 1 to switch it off. The common cathode is the opposite with 0 to switch off any segment and 1 to switch it on.

## **Procedure**

- 1- To test the 7-segment use your Digital multi-meter and set it in the ohmmeter, connect the red probe to pin 3 or 8 of the 7-segment and run the black probe on pins 1,2,4,5,6,7,9,10, as shown in Figure 14.

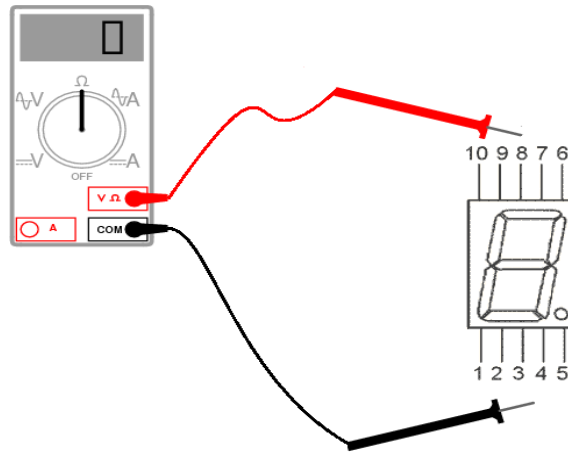


Figure 14

2- What happen when you connect the Black Probe to pins other than 3,8?

.....

3- Use the above method to Fill in the following table which describe the pins of each segment refer to Figure 12 and Figure 14.

Segment	Pin
A	
B	
C	
D	
E	
F	
G	

4- Follow the schematic in Figure 15 to connect 7-segments to the circuit you built .What is the function of the 7447 ?

.....  
 .....

5- Following the Next Flow Chart write a C++ program to switch all the segments of the 7-Segments on

```

While (1)
{
    While (i<=9)
    {
        PORTD.0 = 1;
        PORTC = i;
        delay_ms(250);
        i++;
    }
    i=0;
};

```

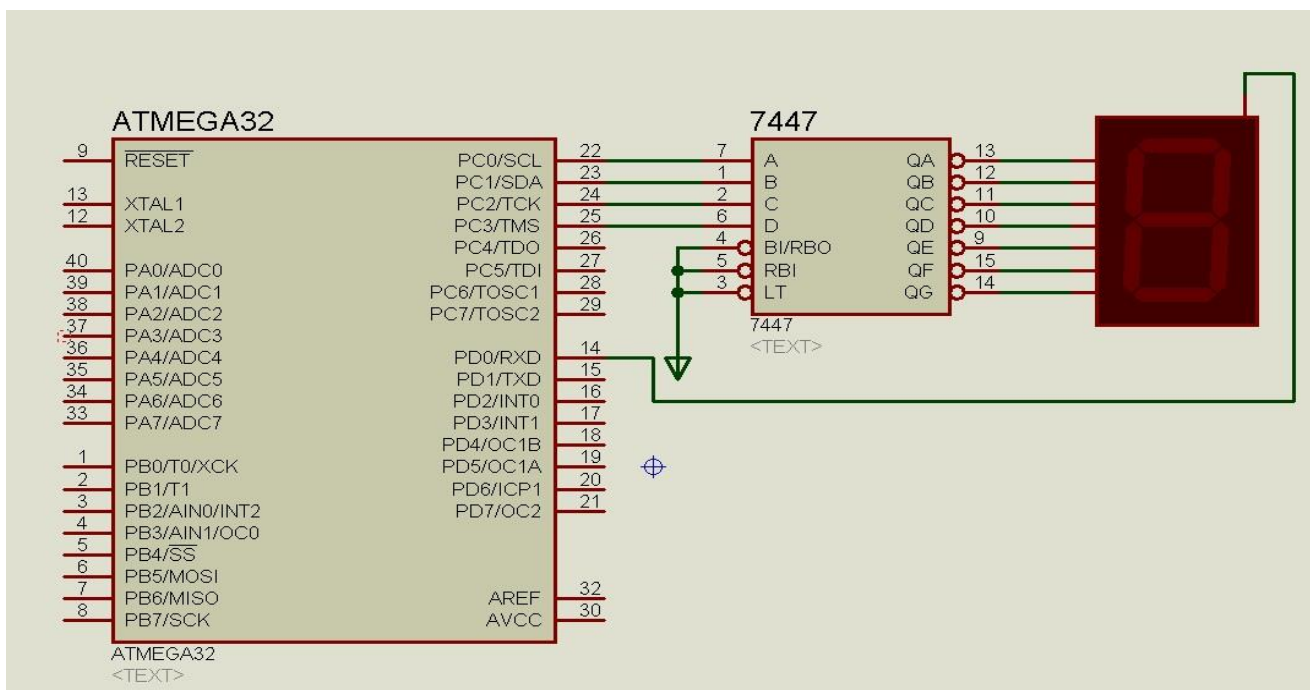
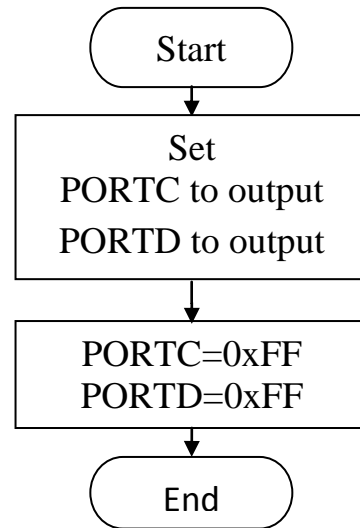


Figure 15

6- From your observation to the circuit which segments are activated, what is the output displayed on it?

.....

.....

.....

Name : \_\_\_\_\_

Reg. No. : \_\_\_\_\_

## **Conclusion**

In this exercise you have learned how to connect the 7-Segment to the ATMEGA16 and configure the microcontroller as a simple BCD counter.

## **Lab assignments**

Change your connection and code to make the same BCD counter without the use of 7447 IC



## **Laboratory Experiment No. 8 (Handling outputs)**

### **LCD display**

#### **Objective**

Upon completing this exercise you will be able to use LCD in 4-bit mode

#### **Discussion**

Liquid crystal displays (LCD) come in two main types that are of interest Character LCD displays and pixel (graphic) LCD displays. This exercise will cover the more popular and less expensive character LCD displays based on the very common Hitachi HD44780 controller.

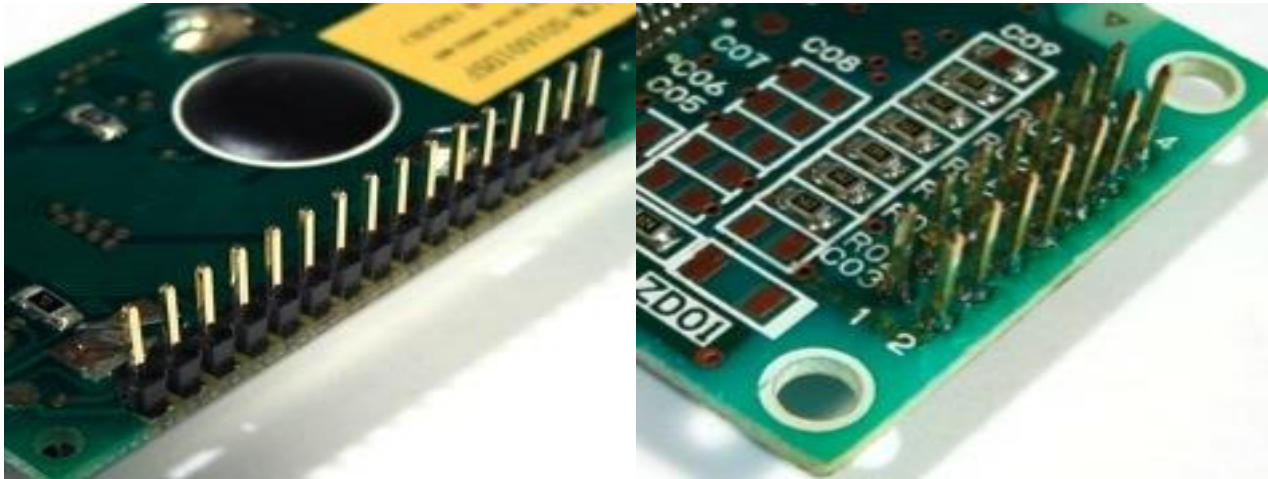
LCD displays come in many sizes most often named by the number of rows and then the length of the display line. For example as shown in Figure 2-6 a 1x16 LCD display will have one row of sixteen characters similarly a 4x20 LCD display will have four rows with twenty characters in each.



LCDs can be have backlighting or be reflective (think calculator). In either case the programming that goes into working these displays is the same. LCDs with backlight normally use two pins to provide power to the backlighting.

Most LCDs that are being made now come with one row of sixteen pins as shown in Figure 2-7. The first fourteen pins are used to control the display and the last two are for the backlighting (if the display has backlighting).

Older LCDs sometimes came with two rows of seven making a fourteen pin connector. These fourteen pins, most often, have the same signals on them that the 1x16 pin displays do. For example, pin #1 on the 2x7 connector is the same signal as pin #1 on the 1x16 connector, they are just arranged differently. If you have a 2x7 pin display but need to connect it to a 1x14 (1x16) backpack or device, the basic Message Pump PCB makes a great converter.

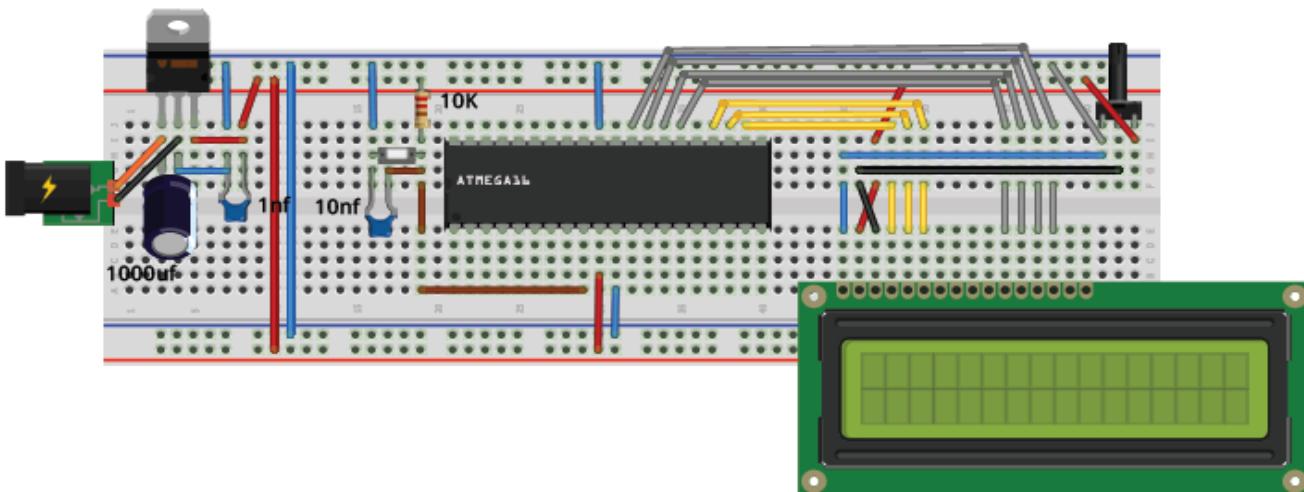


Pin	Function
1	Ground
2	Positive 5 volt
3	VEE (Connect to +5 and GND through 10k Pot for display contrast)
4	RS (Low for command - high for characters)
5	RW (Low for write - high for read)
6	EN Enable Pin (Toggle to load data and command)
7-14	Data pins
15	Backlight +
16	Backlight Ground

### Procedure

- 1- Connect the Circuit shown in Figure 16 use Figure 17 to verify your connections

Figure 16



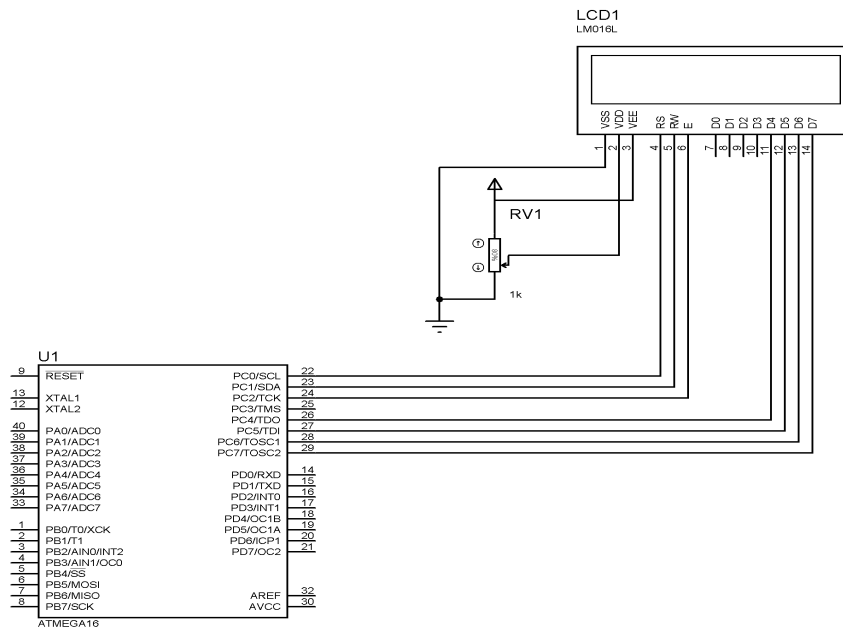


Figure 17

2-The LCD module must be connected to the port bits:

[LCD Pins]	[AVR Port]
RS (pin4)	
RD (pin 5)	
EN (pin 6)	
DB4 (pin 11)	
DB5 (pin 12)	
DB6 (pin 13)	
DB7 (pin 14)	

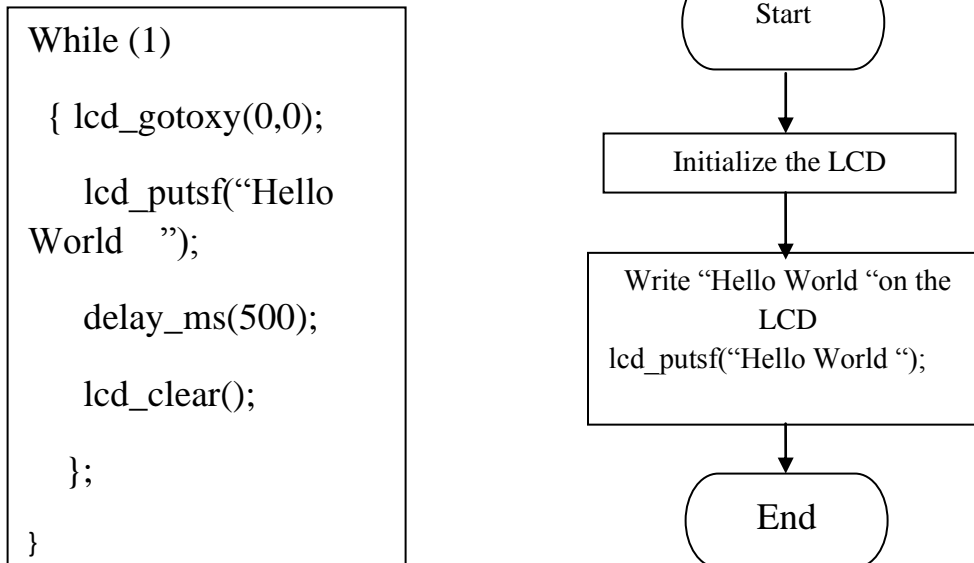
3-CodeVision LCD Commands

```

lcd_gotoxy(x,y);
//Where: x = is the position of the character (0 to 15)
//y = is the line number (0 to 1) Assuming LCD16*2
lcd_putsf("AVR ");
lcd_clear();
//For clearing the LCD
Char text[16];
Sprintf(text,"X = %d",x);
Lcd_puts(text);

```

4-Now we will make the code as Shown in the figure



5- From your observation of the circuit what happen when you down load the code?

.....  
.....

### **Conclusion**

In this exercise you have learned how to connect LCD to the ATMEGA16 and configure the microcontroller to show variables and characters.

### **Lab assignments**

Write a code that show a counter count from 0 to 100 in the LCD

## Experiment No. 9 (Handling input and output)

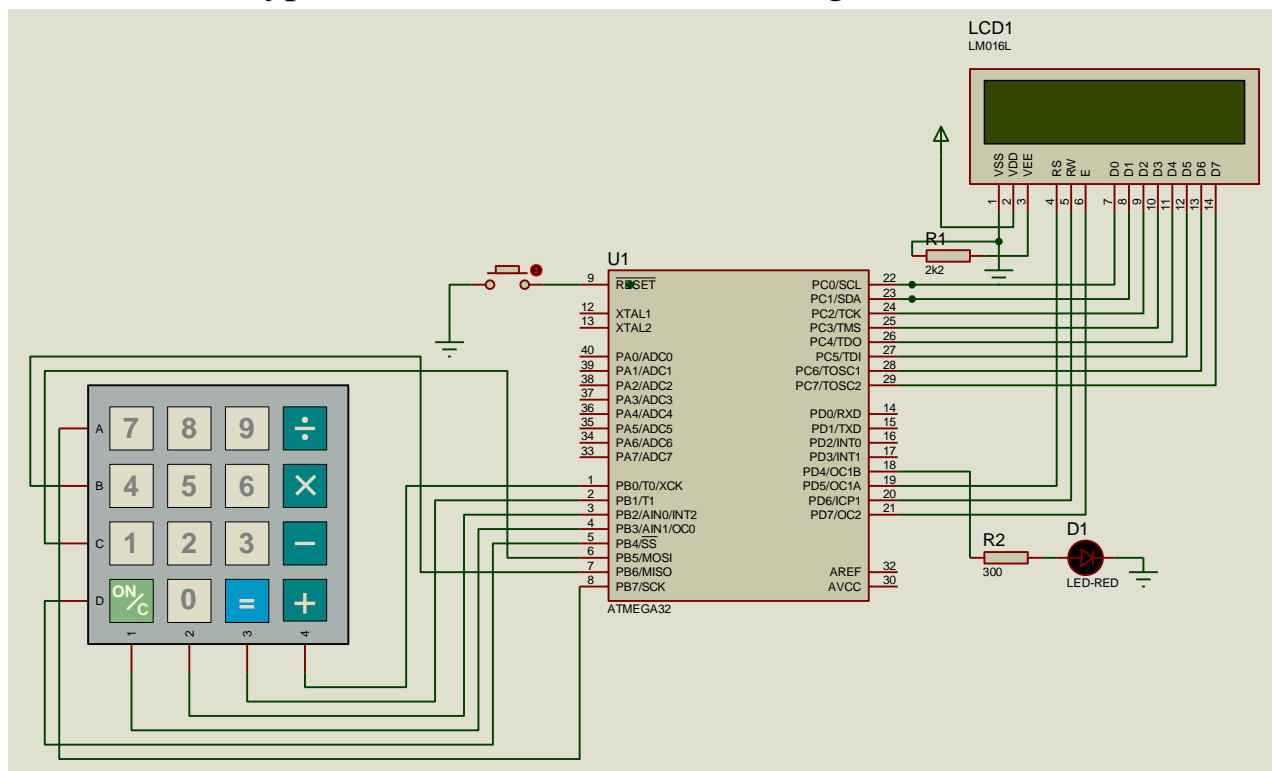
### Keypad and LCD

#### Objective:

This experiment targets the interfacing between a keypad and LCD through a microcontroller.

#### Wiring diagram:

Connect the keypad and the LCD as shown in figure:



#### Code:

```
#define xtal 16000000L
#include <mega32.h>
#include <delay.h>
//***** LCD Functions declaration ***** //
void LCD_init(void);
void LCD_WriteCommand (unsigned char Command);
void LCD_WriteData (unsigned char Data);
void LCD_DisplayString_F(char row, char column, flash unsigned char *string);
void LCD_Cursor(char row, char column);
```

Name : \_\_\_\_\_

Reg. No. : \_\_\_\_\_

```
#define ENABLE_LCD          PORTD |= 0x80
#define DISABLE_LCD        PORTD &= ~0x80
#define SET_LCD_DATA       PORTD |= 0x20
#define SET_LCD_CMD        PORTD &= ~0x20
#define KB_PORT_OUT        PORTB
#define KB_PORT_IN         PINB

void port_init(void)
{
    DDRA  = 0x00;
    PORTA = 0x00;
    DDRB  = 0x0f;      //Key-board port, higer nibble - input, lower nibble - output
    PORTB = 0xff;      //pull-up enabled for higher nibble
    DDRC  = 0xff;
    PORTC = 0x00;
    DDRD  = 0xf0;
    PORTD = 0x00;
}
//call this routine to initialize all peripherals
void init_devices(void)
{
    //stop errant interrupts until set up
    #asm("cli"); //disable all interrupts
    port_init();
    LCD_init();

    MCUCR = 0x00;
    TIMSK = 0x00; //timer interrupt sources
}

void main(void)
{
    unsigned char upperNibble, keyCode, keyPressed, i;
    init_devices();
    LCD_DisplayString_F(1,1,"  WELCOME  ");
    LCD_WriteCommand(0xc0);      //moving LCD cursor to second row
    while(1)
    {
        upperNibble = 0xff;

        for(i=0; i<4; i++)
        {
            delay_ms(1);
            KB_PORT_OUT = ~(0x01 << i);
```

```
delay_ms(1); //delay for port o/p settling
upperNibble = KB_PORT_IN | 0x0f;

if (upperNibble != 0xff)
{
    delay_ms(20); //key debouncing delay
    upperNibble = KB_PORT_IN | 0x0f;
    if(upperNibble == 0xff) goto OUT;

    keyCode = (upperNibble & 0xf0) | (0x0f & ~(0x01 << i));

    while (upperNibble != 0xff)
        upperNibble = KB_PORT_IN | 0x0f;

    delay_ms(20); //key debouncing delay

    switch (keyCode) //generating key characetr to display on LCD
    {
        case (0xee): keyPressed = '+';
            break;
        case (0xed): keyPressed = '=';
            break;
        case (0xeb): keyPressed = '0';
            break;
        case (0xe7): keyPressed = 'C';
            break;
        case (0xde): keyPressed = '-';
            break;
        case (0xdd): keyPressed = '3';
            break;
        case (0xdb): keyPressed = '2';
            break;
        case (0xd7): keyPressed = '1';
            break;
        case (0xbe): keyPressed = 'X';
            break;
        case (0xbd): keyPressed = '6';
            break;
        case (0xbb): keyPressed = '5';
            break;
        case (0xb7): keyPressed = '4';
            break;
        case (0x7e): keyPressed = '/';
            break;
```

```
        case (0x7d): keyPressed = '9';
                break;
        case (0x7b): keyPressed = '8';
                break;
        case (0x77): keyPressed = '7';
                break;
        default      : keyPressed = 'X';
    } //end of switch
    LCD_WriteData(keyPressed);
    OUT;;
} //end of if
} //end of for
} //end of while(1)
} //end of main()

void LCD_init(void)
{
    delay_ms(100);                // wait for 100ms

    //SET_LCD_WRITE ;                // Set LCD in write mode

    LCD_WriteCommand (0x38);        // 8 data lines
    LCD_WriteCommand (0x08);        // display off
    LCD_WriteCommand (0x01);        // clear LCD memory
    delay_ms (10);                  // 10ms delay after clearing LCD
    LCD_WriteCommand (0x06);        // cursor setting
    LCD_WriteCommand (0x0f);        // display ON
}

void LCD_WriteCommand (unsigned char Command)
{
    SET_LCD_CMD;                    // Set LCD in command mode
    PORTC = Command;                // Load data to port
    ENABLE_LCD;                      // Write data to LCD
    #asm("nop");
    #asm("nop");
    DISABLE_LCD;                    // Disable LCD
    delay_ms(1);                    // wait for 1ms
}

void LCD_WriteData (unsigned char Data)
```



```
{
    SET_LCD_DATA;                // Set LCD in data mode
    PORTC = Data;                // Load data to port
    ENABLE_LCD;                  // Write data to LCD
    #asm("nop");
    #asm("nop");
    DISABLE_LCD;                // Disable LCD
    delay_ms(1);                // wait for 1ms
}

void LCD_DisplayString_F (char row, char column, flash unsigned char *string)
{
    LCD_Cursor (row, column);
    while (*string)
        LCD_WriteData(*string++);
}

void LCD_Cursor (char row, char column)
{
    switch (row)
    {
        case 1: LCD_WriteCommand (0x80 + column - 1); break;
        case 2: LCD_WriteCommand (0xc0 + column - 1); break;
        default: break;
    }
}
```

## **Task:**

Modify the code to design a calculator

Name : \_\_\_\_\_

Reg. No. : \_\_\_\_\_

## Laboratory Experiment No. 10 (Handling A/D)

### Analogue to Digital Conversion

#### Objective

We want to use to read the analog signals from sensors or potentiometer using the ADC in microcontroller ATmega16.

#### Lab Equipment

Equipment	Quantity
ATmega32 or ATmega16	1
Potentiometer 10k $\Omega$	1
LCD 16x2	1

#### Discussion

An ADC samples an analogue signal at discrete times, and converts the sampled signal to digital form.

Used with transducers, ADCs allow us to monitor real world inputs and perform control operations based on these inputs.

Many dedicated ICs are made for ADC like **ADC0804: 8-bit, successive approximation.**  
**Maxim104: 8-bit, flash type.**

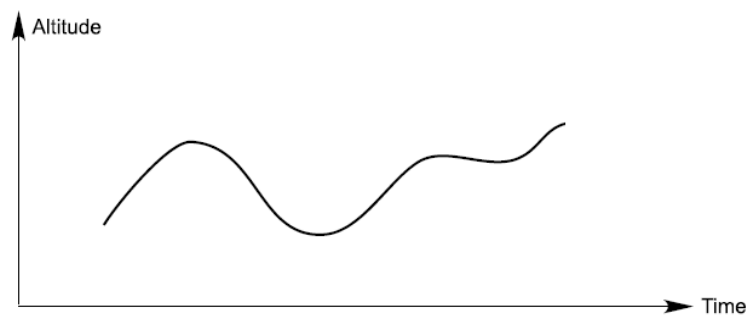


Figure 18

There are two related steps in A-to-D conversion:

1. Sampling
2. Quantization

**Sampling:**

The analogue signal is extracted, usually at regularly spaced time instants. The samples have real values.

**Quantization:**

The samples are quantized to discrete levels. Each sample is represented as a digital value.

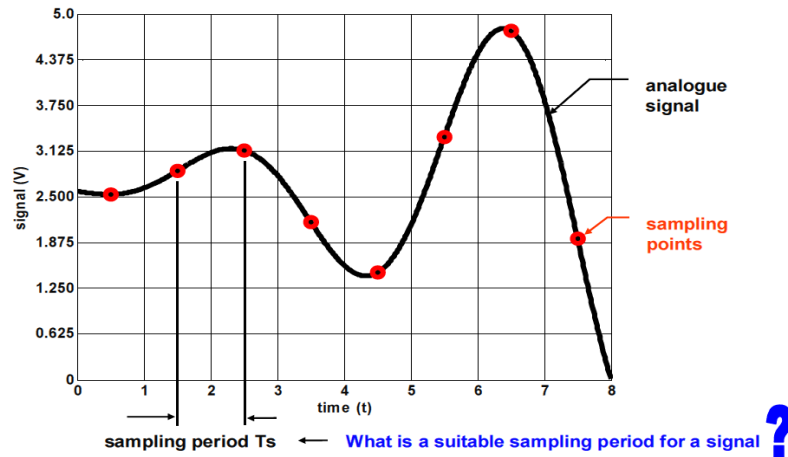


Figure 19

An analogue signal  $x(t)$  with frequencies of no more than  $F_{\max}$  can be reconstructed exactly from its samples if the sampling rate satisfies:

$$F_s \geq 2F_{\max}$$

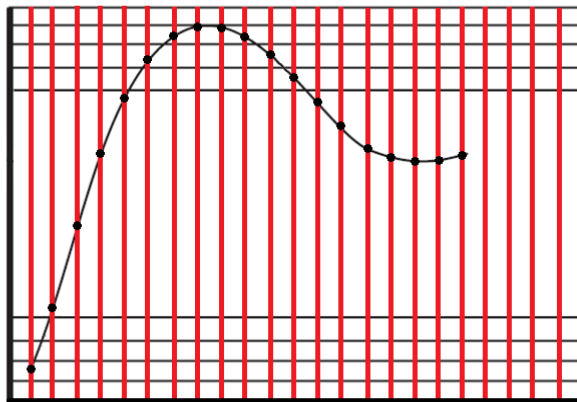
**Significance**

If maximum frequency of the signal is known to be  $F_{\max}$ , the sampling rate we use should be at least:

$$\text{Nyquist rate} = 2 \times F_{\max}$$

1111 1111 Level 255  
1111 1110 Level 254  
1111 1101 Level 253

0000 0100 Level 4  
0000 0011 Level 3  
0000 0010 Level 2  
0000 0001 Level 1  
0000 0000 Level 0



Resolution =  $R = 8$  bits

Number of Levels =  $L = 2^8 = 256$  Level

Max Analog input =  $V_{\text{ref}} = 5$  V

Step Level =  $\frac{V_{\text{ref}}}{L} = 20$  mv

Max Error = 50% of Step Level = 10 mv

Figure 20

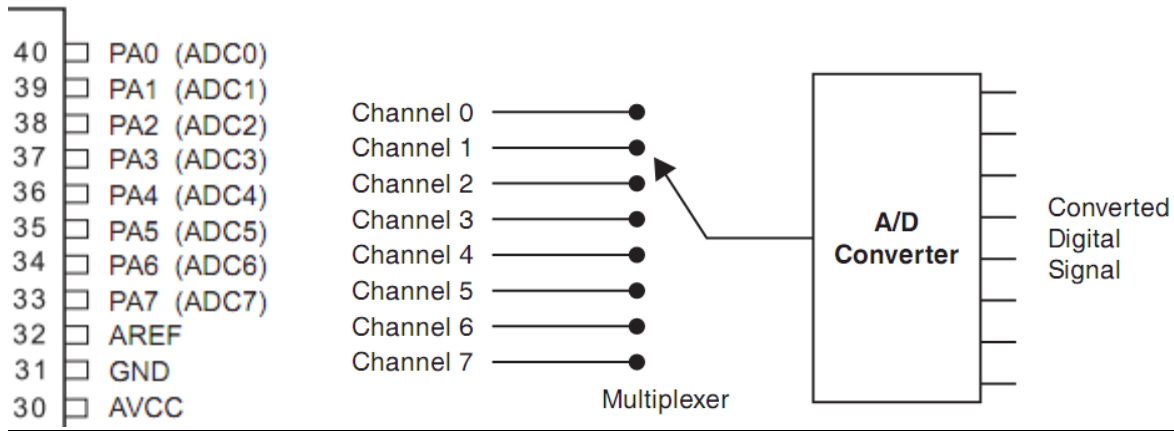


Figure 21

The AVR Microcontroller have ADC module that contains 8 channels that can be used to convert analog data (0 - Vref) to digital data (8-bits or 10 bits).

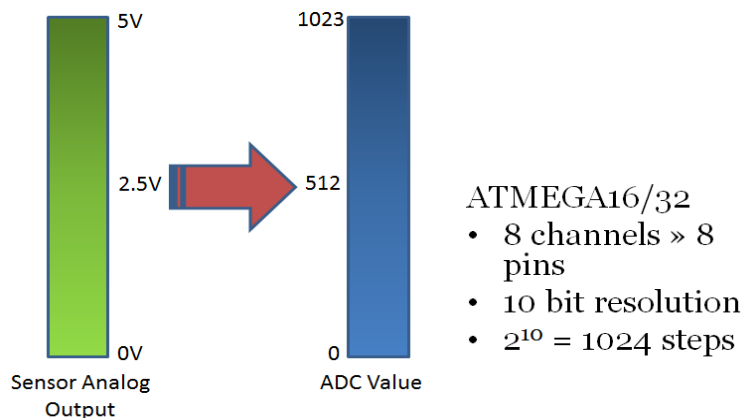


Figure 22

## ADC Operation:

The ADC converts an analog input voltage to a 10-bit digital value through successive approximation. The minimum value represents GND and the maximum value represents the voltage on the AREF pin minus 1 LSB. Optionally, AVCC or an internal 2.56V reference voltage may be connected to the AREF pin.

The internal voltage reference may thus be decoupled by an external capacitor at the AREF pin to improve noise immunity.

The ADC generates a 10-bit result which is presented in the ADC Data Registers, ADCH and ADCL. By default, the result is presented right adjusted, but can optionally be presented left adjusted by setting the ADLAR bit in ADMUX.

The ADC has its own interrupt which can be triggered when a conversion completes.

## Procedure

1-Connect the potentiometer to PORTA.0 and the LCD to PORTC

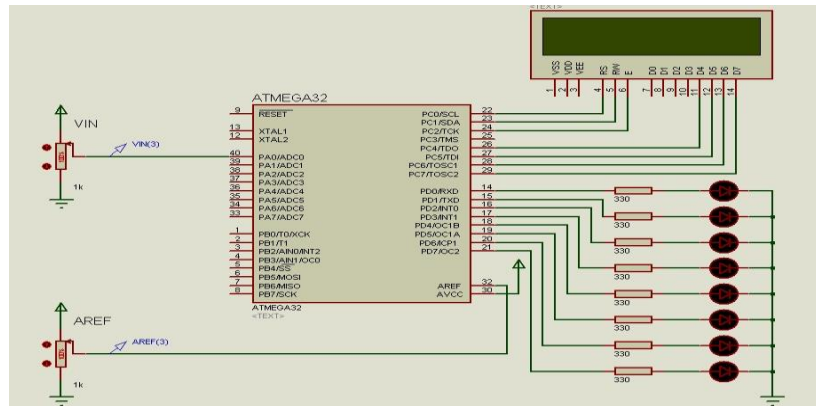


Figure 23

2-Write the ADC function

```
void adc_init()
{
    // AREF = AVcc
    ADMUX = (1<<REFS0);

    // ADC Enable and prescaler of 128
    // 16000000/128 = 125000
    ADCSRA = (1<<ADEN)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0);
}

// read adc value
uint16_t adc_read(uint8_t ch)
{
    // select the corresponding channel 0~7
    // ANDing with '7' will always keep the value
    // of 'ch' between 0 and 7
    ch &= 0b00000111; // AND operation with 7
    ADMUX = (ADMUX & 0xF8)|ch; // clears the bottom 3 bits before ORing

    ADCSRA |= (1<<ADSC); // start single conversion
                        // write '1' to ADSC

    while(ADCSRA & (1<<ADSC)); // wait for conversion to complete
                        // ADSC becomes '0' again
                        // till then, run loop continuously

    return (ADC);
}
```

3-Write the main code

```
lcd_init(16);  
  
while (1)  
{  
    x = read_adc(0);  
  
    PORTD = x;  
  
    lcd_gotoxy(0,0);  
  
    sprintf(text,"ADC = %d",x);  
  
    lcd_puts(text);  
  
    delay_ms(100);  
  
    lcd_clear();  
  
};  
}
```

4- From your observation of the circuit what happen when you download the code?

.....  
.....

## **Conclusion**

In this exercise you have learned how operate the ADC in the microcontroller.

## **Lab assignments**

A temperature sensor used in a process control system gives 5mv/°C. There are three indication lamps used for operation indication:

LED 1 ON: When temperature is greater than 50°C

LED 2 ON: When temperature is greater than 100°C

LED 3 ON: When temperature is greater than 250°C

## **Laboratory Experiment No. 11 (Handling interrupts)**

### **AVR Microcontroller Interrupts**

#### **Objective**

We want to use to read the analog signals from sensors or potentiometer using the ADC in microcontroller ATmega16.

#### **Lab Equipment**

Equipment	Quantity
ATmega32 or ATmega16	1
Resistors 10k $\Omega$	3
Push buttons	3
Leds	3
Resistors 330 $\Omega$	3

#### **Discussion**

Interrupts are basically events that require immediate attention by the microcontroller. When an interrupt event occurs the microcontroller pause its current task and attend to the interrupt by executing an Interrupt Service Routine (ISR) at the end of the ISR the microcontroller returns to the task it had pause and continue its normal operations.

In order for the microcontroller to respond to an interrupt event the interrupt feature of the microcontroller must be enabled along with the specific interrupt. This is done by setting the Global Interrupt Enabled bit and the Interrupt Enable bit of the specific interrupt.

#### **Interrupt Service Routine or Interrupt Handler**

An **Interrupt Service Routine (ISR)** or **Interrupt Handler** is a piece of code that should be execute when an interrupt is triggered. Usually each enabled interrupt has its own ISR

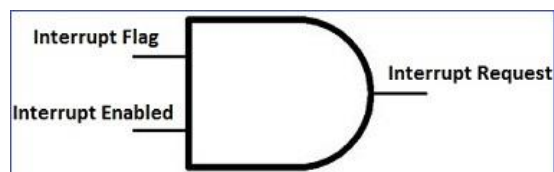
Vector No.	Program Address	Interrupt vector name	Description
1	\$000	RESET_vect	Reset
2	\$002	INT0_vect	External Interrupt Request 0
3	\$004	INT1_vect	External Interrupt Request 1
4	\$006	TIMER2_COMP_vect	Timer/Counter2 Compare Match
5	\$008	TIMER2_OVF_vect	Timer/Counter2 Overflow
6	\$00A	TIMER1_CAPT_vect	Timer/Counter1 Capture Event
7	\$00C	TIMER1_COMPA_vect	Timer/Counter1 Compare Match A
8	\$00E	TIMER1_COMPB_vect	Timer/Counter1 Compare Match B
9	\$010	TIMER1_OVF_vect	Timer/Counter1 Overflow
10	\$012	TIMER0_OVF_vect	Timer/Counter0 Overflow
11	\$014	SPI_STC_vect	Serial Transfer Complete
12	\$016	USART_RXC_vect	USART, Rx Complete
13	\$018	USART_UDRE_vect	USART Data Register Empty
14	\$01A	USART_TXC_vect	USART, Tx Complete
15	\$01C	ADC_vect	ADC Conversion Complete
16	\$01E	EE_RDY_vect	EEPROM Ready
17	\$020	ANA_COMP_vect	Analog Comparator
18	\$022	TWI_vect	2-wire Serial Interface
19	\$024	INT2_vect	External Interrupt Request 2
20	\$026	TIMER0_COMP_vect	Timer/Counter0 Compare Match
21	\$028	SPM_RDY_vect	Store Program Memory Ready

### Interrupt Flags and Enabled bits

Each interrupt is associated with two (2) bits, an **Interrupt Flag Bit** and an **Interrupt Enabled Bit**. These bits are located in the I/O registers associated with the specific interrupt:

- The **interrupt flag** bit is set whenever the interrupt event occur, whether or not the interrupt is enabled.
- The **interrupt enabled** bit is used to enable or disable a specific interrupt. Basically it tells the microcontroller whether or not it should respond to the interrupt if it is triggered.

In summary basically both the **Interrupt Flag** and the **Interrupt Enabled** are required for an interrupt request to be generated as shown in the figure below.





## Global Interrupt Enabled Bit

Apart from the enabled bits for the specific interrupts the global interrupt enabled bit **MUST** be enabled for interrupts to be activated in the microcontroller.

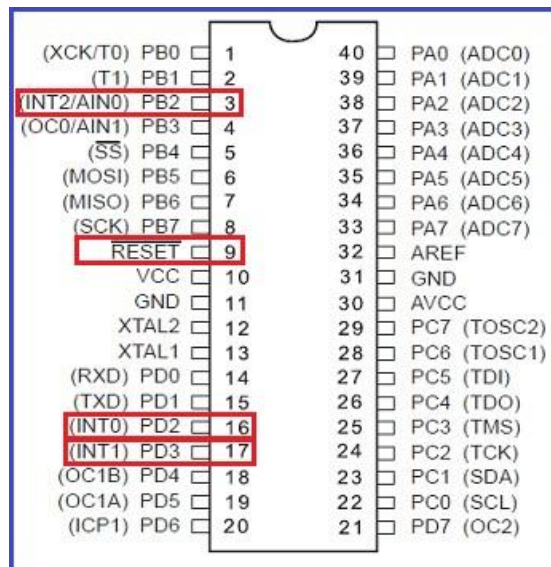
For the AVR 8-bits microcontroller this bit is located in the **Status I/O Register (SREG)**. The Global Interrupt Enabled is bit 7, the **I** bit, in the SREG.



## Interrupt sources provided with the AVR microcontroller

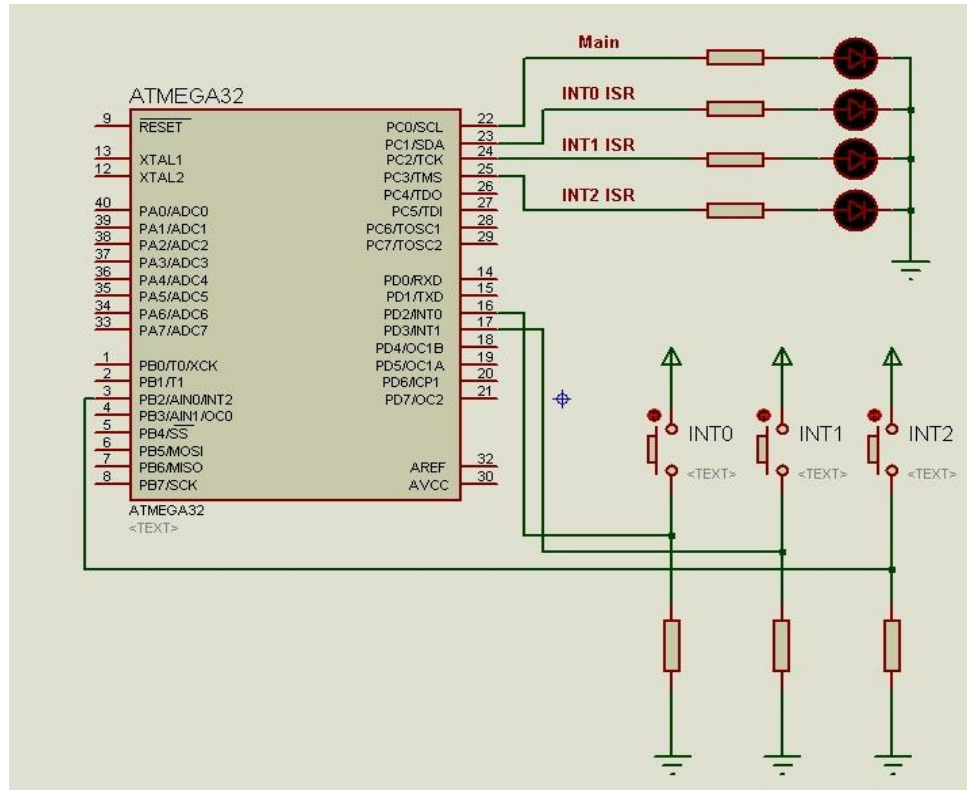
The AVR 8-bits microcontroller provides both internal and external interrupt sources. The internal interrupts are associated with the microcontroller's peripherals. That is the **Timer/Counter, Analog Comparator, etc.** The external interrupts are triggered via external pins. The figure below shows the pins, on which the external interrupts can be triggered, for an AVR 8-bit microcontroller. On this microcontroller there are four (4) external interrupts:

1. The **RESET** interrupt - Triggered from pin 9.
2. **External Interrupt 0 (INT0)** - Triggered from pin 16.
3. **External Interrupt 1 (INT1)** - Triggered from pin 17.
4. **External Interrupt 2 (INT2)** - Triggered from pin 3.



## Procedures

1-Connect the push buttons to the interrupt pins and the output leds to any PORT



2-Write the interrupt function using AVR studio program

```
#define F_CPU 8000000UL
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
ISR(INT0_vect)
{ uint8_t i=0;
  for(i=0;i<=5;i++)
  {
    PORTC=0b00000010;
    _delay_ms(500);
    PORTC=0b00000000;
    _delay_ms(500);
  }
}
```

## **Laboratory Experiment No. 12 (Handling Timer)**

### **Timer & Timer Interrupt**

#### **Objective:**

The objective of this experiment is to calculate the real time of an action

#### **Procedures:**

- 1- Set the timer zero of the microcontroller to normal top=0xFF.
- 2- Enable timer interrupt.
- 3- Count the number of interrupts occurred.
- 4- Calculate the real time.

#### **Write software program**

```
#include <mega16.h>
#define F_CPU 8000000
// Timer 0 overflow interrupt service routine
int count ;
float real_time;
interrupt [TIM0_OVF] void timer0_ovf_isr(void)
{ count ++; }
void main(void)
{
// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: 8000.000 kHz
// Mode: Normal top=0xFF
// OC0 output: Disconnected
TCCR0=0x01;
TCNT0=0x00;
OCR0=0x00;
// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=0x01;
// Global enable interrupts
#asm("sei")
while (1)
{ real_time=count*(256/F_CPU)+(TCNT0/F_CPU); }}
```

#### **Questions:**

Name : \_\_\_\_\_

Reg. No. : \_\_\_\_\_

1- State the reason for crystal oscillator selection instead of internal 8 MHZ

.....  
.....

2- If the CPU frequency changed to 12 MHZ, state the new transfer function

.....  
.....

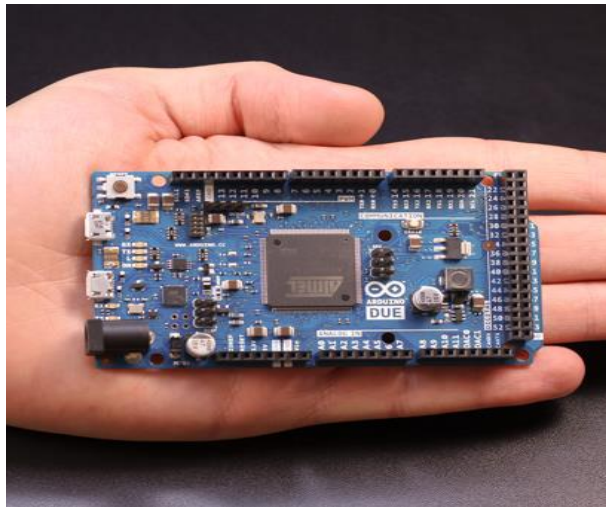
3- Implement a real time Clock that displays the output to a LCD

.....  
.....

## Arduino Experiments

### Description

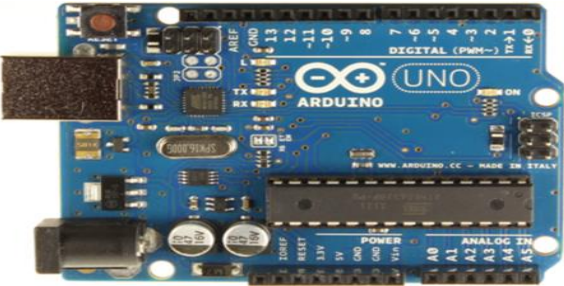
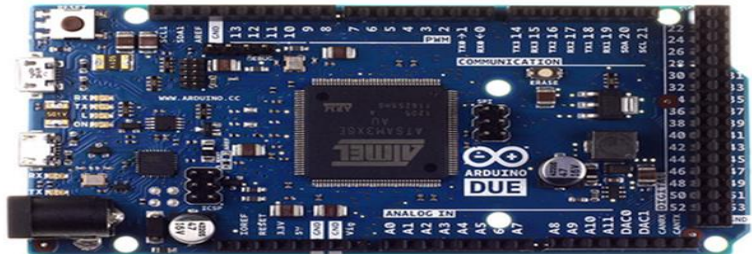
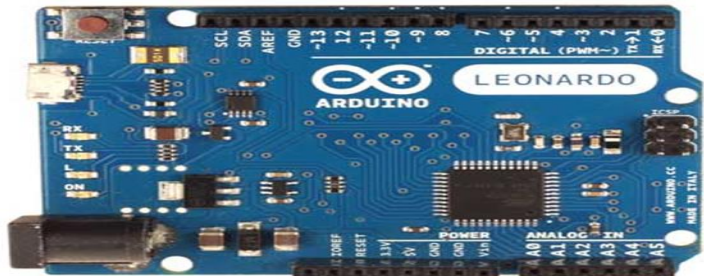
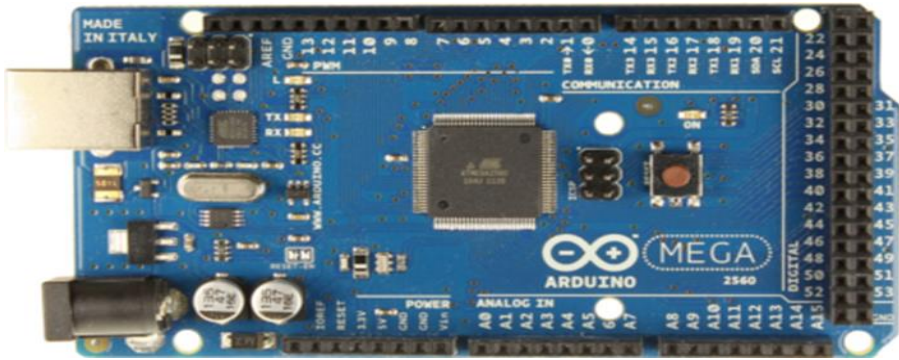
Arduino is an open-source electronics prototyping platform based on flexible, easy-to-use hardware and software. It's intended for artists, designers, hobbyists and anyone interested in creating interactive objects or environments.



Arduino can sense the environment by receiving input from a variety of sensors and can affect its surroundings by controlling lights, motors, and other actuators.

The boards can be built by hand or purchased preassembled; the software can be downloaded for free. The hardware reference designs are available under an open-source license; you are free to adapt them to your needs.

## Comparison between popular Arduino Boards

Name	Processor	Operating Voltage/Input Voltage	CPU Speed	Analog In/Out	Digital IO/PWM	EEPROM [KB]	SRAM [KB]	Flash [KB]	USB	UART
Uno	ATmega328	5 V/7-12 V	16 Mhz	6/0	14/6	1	2	32	Regular	1
										
Due	AT91SAM3X8E	3.3 V/7-12 V	84 Mhz	12/2	54/12	-	96	512	2 Micro	4
										
Leonardo	ATmega32u4	5 V/7-12 V	16 Mhz	12/0	20/7	1	2.5	32	Micro	1
										
Mega 2560	ATmega2560	5 V/7-12 V	16 Mhz	16/0	54/15	4	8	256	Regular	4
										

Name : \_\_\_\_\_

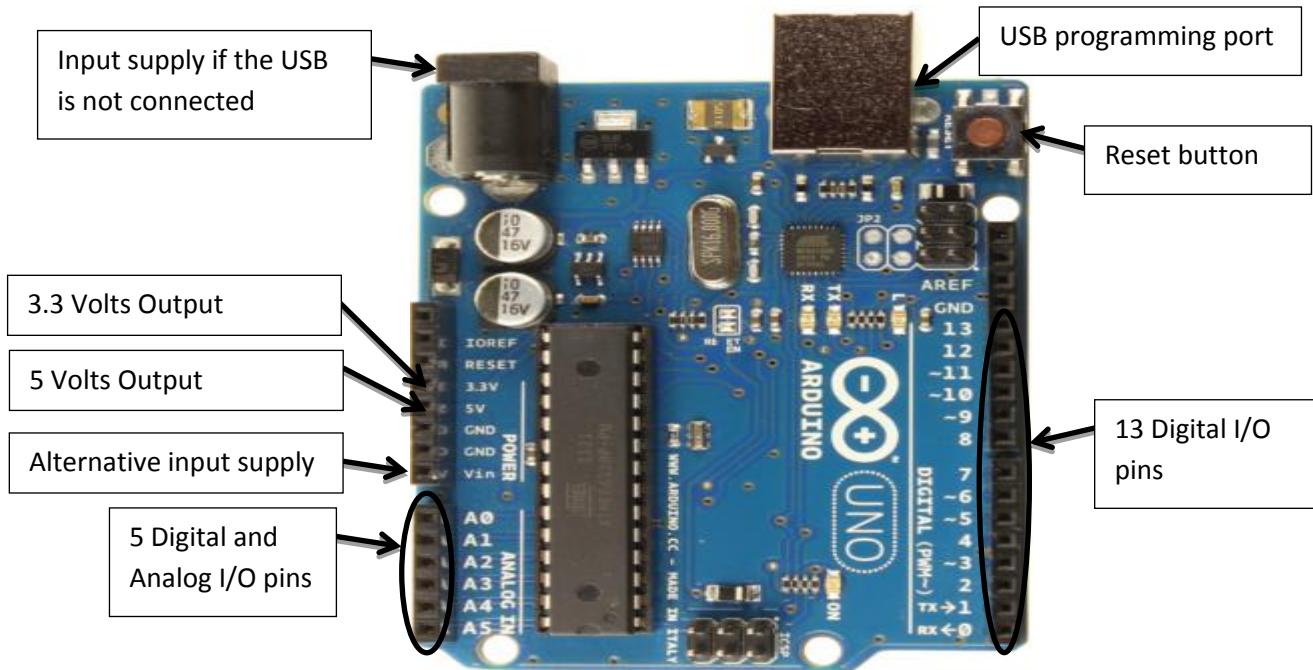
Reg. No. : \_\_\_\_\_



## Important to know:

### First: Arduino UNO Input and output description

We need to be familiar with the Arduino pins, so these pins are discussed as follows.



Each of the 14 digital pins on the Uno can be used as an input or output, using **pinMode()**, **digitalWrite()**, and **digitalRead()** functions. They operate at 5 volts. Each pin can provide or receive a maximum of 40 mA and has an internal pull-up resistor (disconnected by default) of 20-50 Kohms. In addition, some pins have specialized functions:

- **Serial: 0 (RX) and 1 (TX).** Used to receive (RX) and transmit (TX) TTL serial data. These pins are connected to the corresponding pins of the ATmega8U2 USB-to-TTL Serial chip.
- **External Interrupts: 2 and 3.** These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value. See the **attachInterrupt()** function for details.
- **PWM: 3, 5, 6, 9, 10, and 11.** Provide 8-bit PWM output with the **analogWrite()** function.
- **SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK).** These pins support SPI communication using the SPI library.
- **LED: 13.** There is a built-in LED connected to digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.

The Uno has 6 analog inputs, labeled A0 through A5, each of which provide 10 bits of resolution (i.e. 1024 different values). By default they measure from ground to 5 volts, though it is possible to change the upper end of their range using the AREF pin and the **analogReference()** function. Additionally, some pins have specialized functionality:

- **TWI: A4 or SDA pin and A5 or SCL pin.** Support TWI communication using the Wire library.

There are a couple of other pins on the board:

- **AREF.** Reference voltage for the analog inputs. Used with **analogReference()**.
- **Reset.** Bring this line LOW to reset the microcontroller. Typically used to add a reset button to shields which block the one on the board.

## Second: Arduino UNO Programming

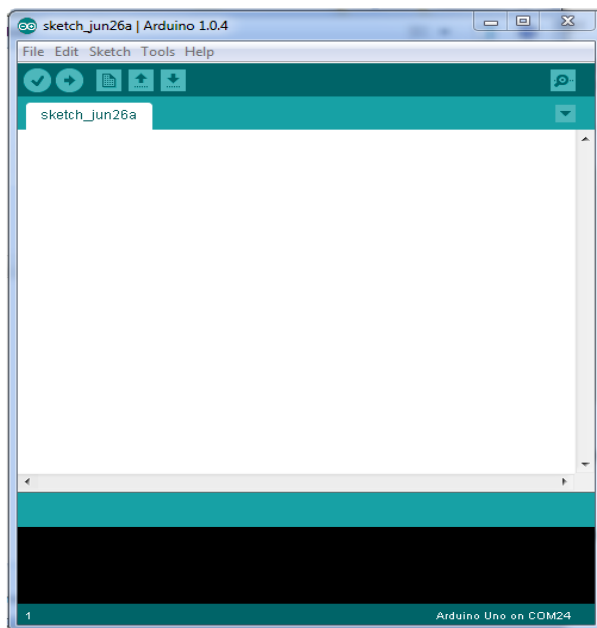
The open-source Arduino environment makes it easy to write code and upload it to the i/o board.

This link you can download the software needed from: <http://arduino.cc/en/Main/Software>

The software doesn't need to be setup just run the program normally.

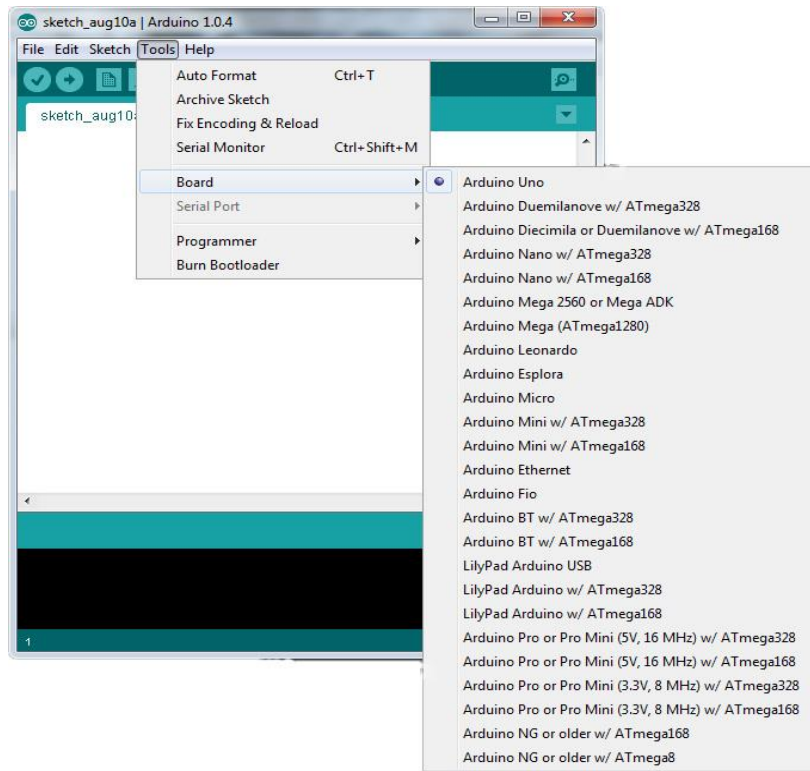
The following shows how to use Arduino software.

- 1- First start Arduino program then the following figure should appear.

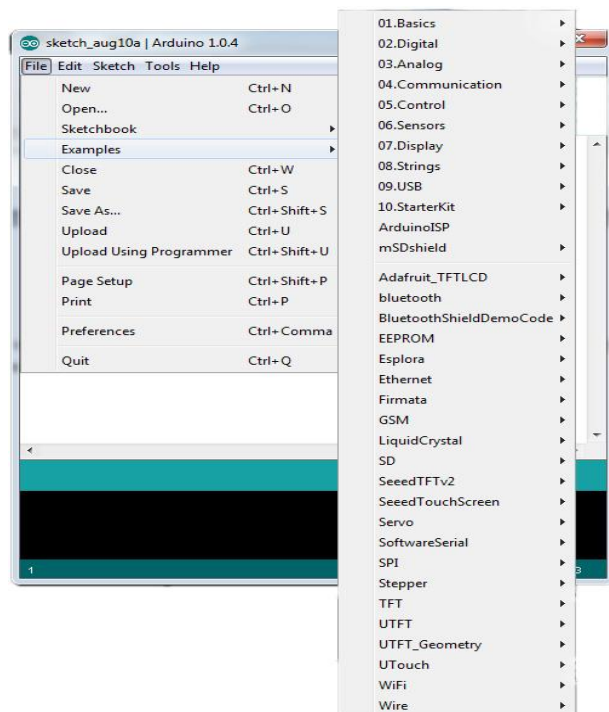




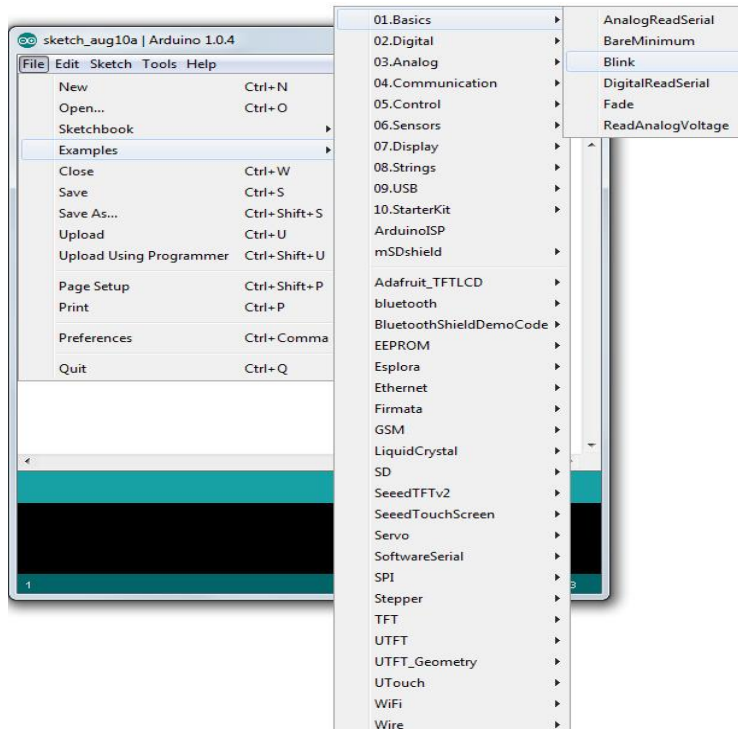
- 2- Click tools → Board (check that Arduino Uno is chosen “This option can be changed according to the Arduino’s type needed to be programmed”).



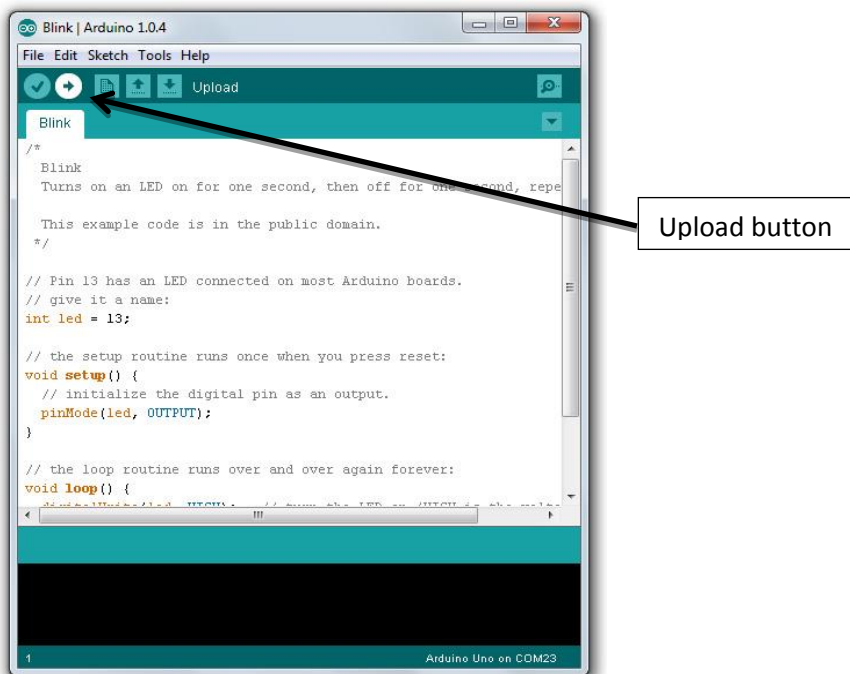
- 3- Start writing or choosing the program needed to be transferred to the Arduino. Choose the program from the examples built in the software as follows:



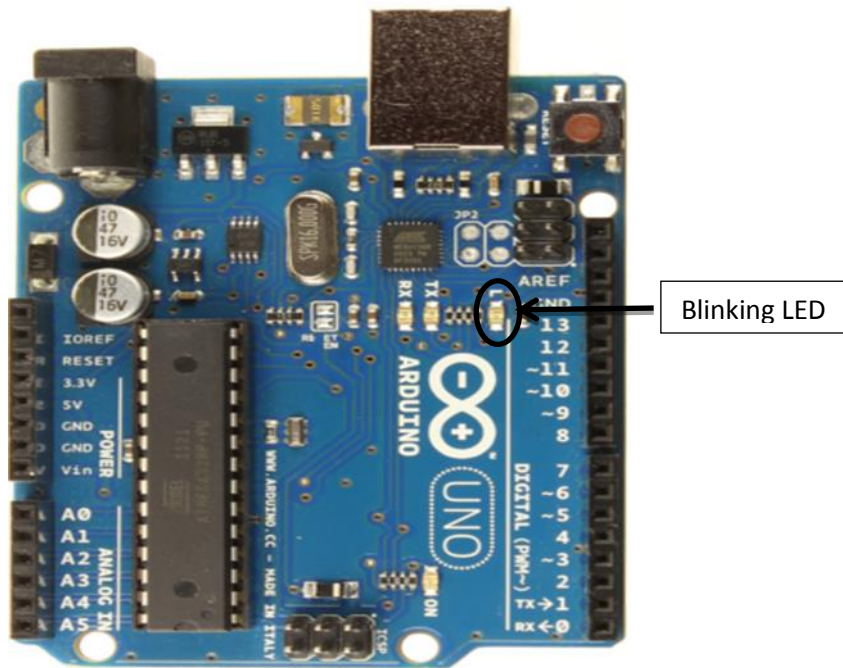
4- Choose a simple example like blink.



5- New window will open as follows click upload button then the software will compile the program for errors and then upload it to the Arduino.



After the program is successfully uploaded there will be a lamp in the Arduino board will flashes on/off continuously.



## Experiments:

The upcoming Experiments are arranged and depend on each other so experiment number 2 depending on tutorial number 1 etc.

## Experiment No. 1

### Arduino wiring and push buttons

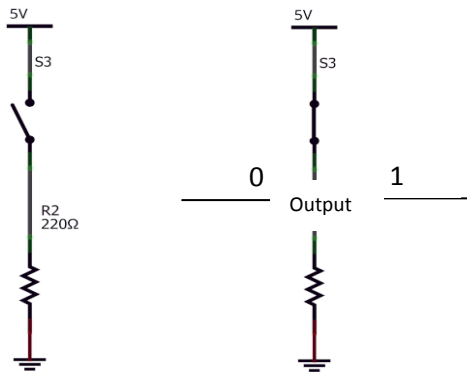
#### Objective:

- Connecting 5V power supply to Arduino Uno.
- Connecting Push button as an input to Arduino.

#### Components:

- Arduino Uno.
- Push Button.
- Resistor 330Ω.
- Regulator 7805.
- Capacitor 1000uf and 1nf.

#### Push button schematic illustration:

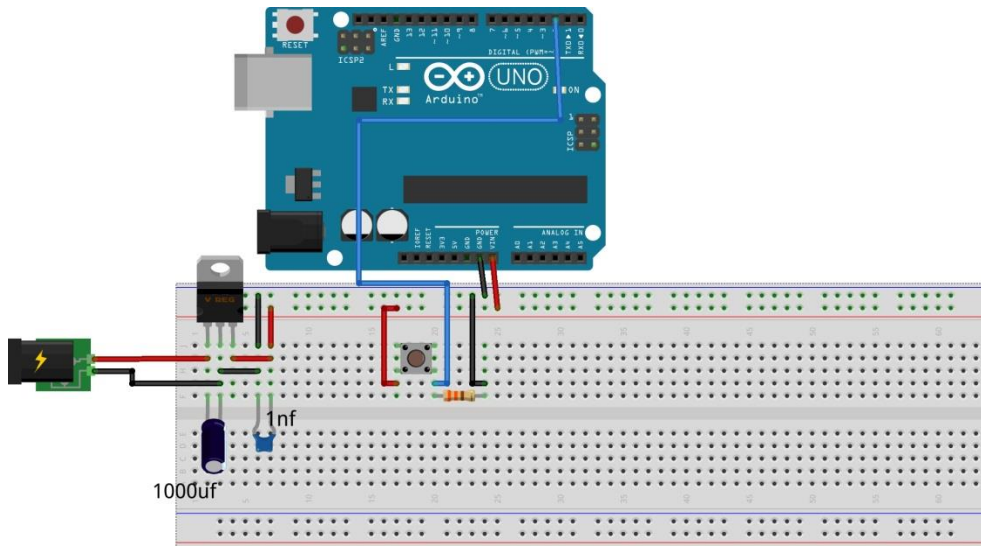


This is how a push button can be connected to any controller

#### Implementation:

The push button will be connected to the Arduino as by pushing it the LED on the Arduino board will be lit and vice versa.

#### Wiring:

**Program:**

```
int led = 13;

void setup()
{
  // initialize the digital pins.
  pinMode(led, OUTPUT);
  pinMode(2, INPUT);
}

// the loop routine runs over and over again forever:
void loop() {
  if(digitalRead(2)==HIGH)
  {
    digitalWrite(led, HIGH);    // turn the LED on (HIGH is the voltage level)
  }
  else
  {
    digitalWrite(led, LOW);     // turn the LED off by making the voltage LOW
  }
}
```

## Experiment no. 2

### Connecting an external LED and seven segments display

#### Objective:

- Connecting an external LED as an output.
- Connecting seven segments display.

#### Components:

- LED.
- Seven segments display.
- 7447 BCD/seven segment decoder or 4543.
- Two 7414 not gate.
- Eight resistors 330Ω.

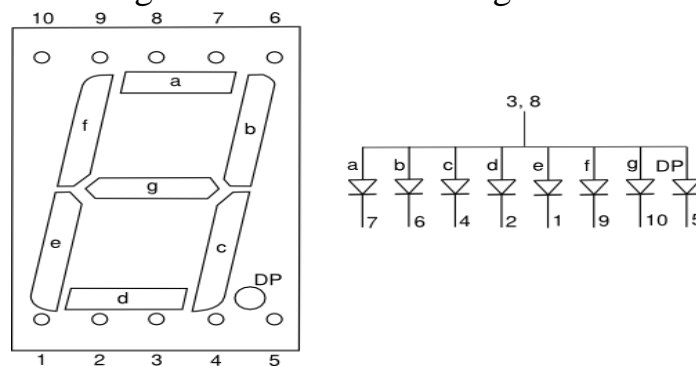
#### Seven segments illustration:

Seven segments are divided into two types which are common anode and common cathode.

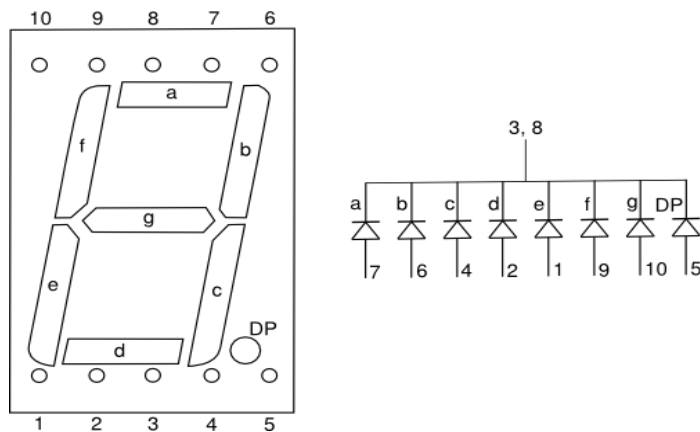
How to determine the type?

To test the 7-segment use your Digital multi-meter and set it in the ohmmeter, connect the red probe to pin 3 or 8 (com pins) of the 7-segment and run the black probe on pins 1,2,4,5,6,7,9,10 (a,b,c,d,e,f,g,dp).

When the meter is set to ohmmeter it acts as a voltage source as the positive terminal is the positive of the source and the negative terminal is the negative of the source.



The above figure shows the common anode type of the seven segments as we can see the pins 3 and 8 are common pins that have to be connected to the positive of the supply and the negative of the supply to any other pin. So by connecting the red probe of the meter to pins 3 or 8 and the other probe to any other pin then any of these segments lit so the seven segments type is anode type.

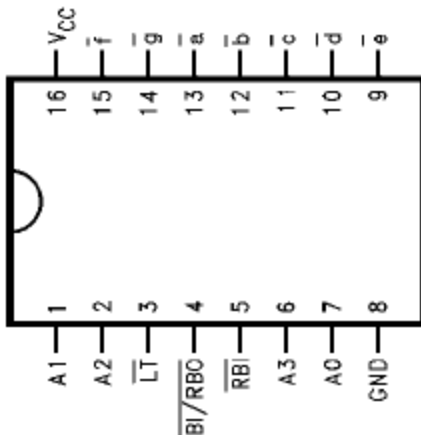


The above figure shows the common cathode type of the seven segments this type is the opposite of the first type in which the negative of the source have to be on the pins 3 and 8 and the black probe has to be on these pins (3,8) and the other probe on any other pins.

### **BCD to seven segment decoder:**

This decoder is responsible for interpreting the binary coded decimal (BCD) numbers to the seven segments as shown in the following table:

Binary Inputs				Decoder Outputs							7-Segment Display Outputs
D	C	B	A	a	b	c	d	e	f	g	
0	0	0	0	1	1	1	1	1	1	0	0
0	0	0	1	0	1	1	0	0	0	0	1
0	0	1	0	1	1	0	1	1	0	1	2
0	0	1	1	1	1	1	1	0	0	1	3
0	1	0	0	0	1	1	0	0	1	1	4
0	1	0	1	1	0	1	1	0	1	1	5
0	1	1	0	1	0	1	1	1	1	1	6
0	1	1	1	1	1	1	0	0	0	0	7
1	0	0	0	1	1	1	1	1	1	1	8
1	0	0	1	1	1	1	1	0	1	1	9

**Decoder 74LS47:**

The connection is very simple as the **VCC**,  **$\overline{LT}$** ,  **$\overline{BI/RBO}$**  and  **$\overline{RBI}$**  are connected to the **positive** of the supply and the **GND** to the **negative** of the supply. **A1, A2, A3** and **A4** pins are the four bits connected to the **controller** and the rest of the pins are connected to the seven segment.

This IC is connected to the seven segments through the  $330\Omega$  resistors. The **common anode** seven segment is connected **directly** to the IC while the **common cathode** has a different configuration, the signal coming from the IC must be **inverted** with the not gate IC then the output is connected to the seven segments.

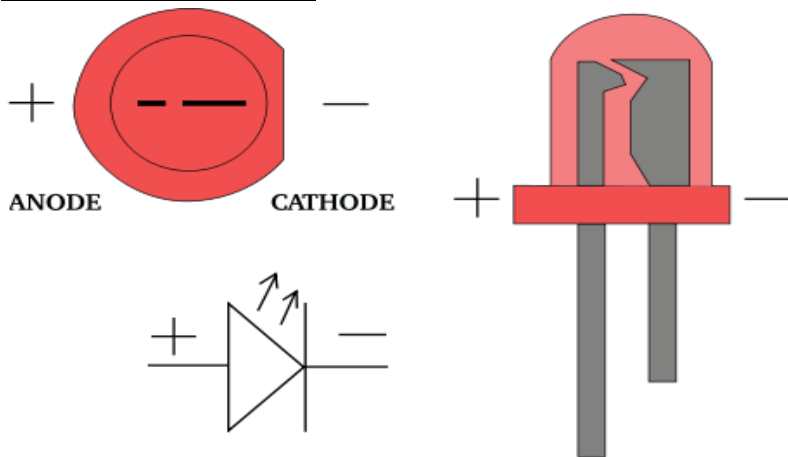
**Decoder 4543:**

This IC connection is as follows:

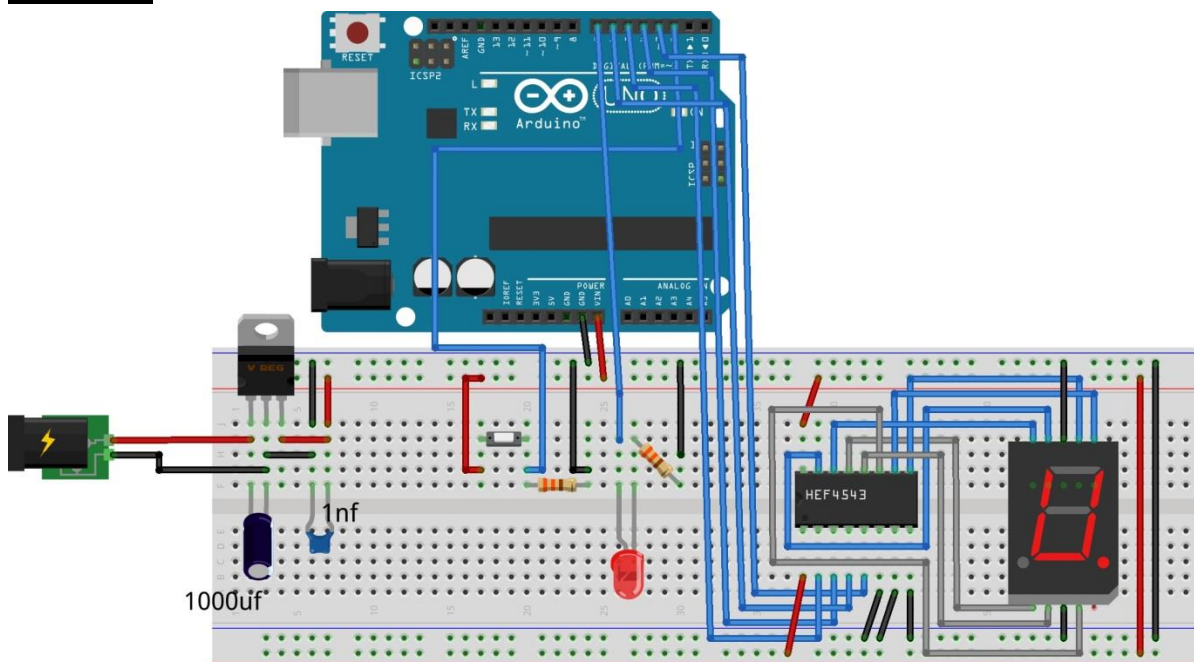
- **V<sub>DD</sub>** and  **$\overline{LE}$**  are connected to the positive terminals of the supply.
- **PH, BL** and **V<sub>ss</sub>** are connected to the negative terminal.
- **D0, D1, D2** and **D3** are connected to the controller.
- The rest are connected to the seven segments.

The IC is connected to the common anode seven segment type through an inverter while the common cathode without an inverter which makes it the opposite of the 74LS47.



**LED illustration:****Implementation:**

The push button is the only input we have and we have two outputs the seven segment display and the LED. This board is for counting from 1 to 9, every time the button is pushed the number on the seven segment increases by 1 until reaching 9 after that the lamp flashes indicating reaching maximum and the counter starts again.

**Wiring:**

**Program:**

```
int buttonState = 0;
void setup() {
  pinMode(7, OUTPUT);
  pinMode(2, INPUT);
  pinMode(3, OUTPUT);
  pinMode(4, OUTPUT);
  pinMode(5, OUTPUT);
  pinMode(6, OUTPUT);  }
int i=0,j;
void loop(){
  buttonState = digitalRead(2);
  // check if the pushbutton is pressed.
  // if it is, the buttonState is HIGH:
  if (buttonState == HIGH)
  {
    i++;
    switch(i)
    {
      case 1: one();
      break;
      case 2: two();
      break;
      case 3: three();
      break;
      case 4: four();
      break;
      case 5: five();
      break;
      case 6: six();
      break;
      case 7: seven();
      break;
      case 8: eight();
      break;
      case 9: nine();
```

```
        break;
    }
    if (i>9)
    {
        for(j=0;j<5;j++)
        {
            digitalWrite(7,HIGH);
            delay(400);
            digitalWrite(7,LOW);
            delay(400);
        }
        zero();
    }
}

void zero()
{  digitalWrite(3,0);
   digitalWrite(4,0);
   digitalWrite(5,0);
   digitalWrite(6,0); }

void one()
{  digitalWrite(3,1);
   digitalWrite(4,0);
   digitalWrite(5,0);
   digitalWrite(6,0); }

void two()
{  digitalWrite(3,0);
   digitalWrite(4,1);
   digitalWrite(5,0);
   digitalWrite(6,0); }

void three()
{  digitalWrite(3,1);
   digitalWrite(4,1);
   digitalWrite(5,0);
   digitalWrite(6,0);}

void four()
{  digitalWrite(3,0);
```

```
digitalWrite(4,0);
digitalWrite(5,1);
digitalWrite(6,0); }
void five()
{ digitalWrite(3,1);
  digitalWrite(4,0);
  digitalWrite(5,1);
  digitalWrite(6,0); }
void six()
{ digitalWrite(3,0);
  digitalWrite(4,1);
  digitalWrite(5,1);
  digitalWrite(6,0); }

void seven()
{ digitalWrite(3,1);
  digitalWrite(4,1);
  digitalWrite(5,1);
  digitalWrite(6,0); }
void eight()
{ digitalWrite(3,0);
  digitalWrite(4,0);
  digitalWrite(5,0);
  digitalWrite(6,1); }
void nine()
{ digitalWrite(3,1);
  digitalWrite(4,0);
  digitalWrite(5,0);
  digitalWrite(6,1);
```

## Questions:

- Redo the experiment without using a decoder.

## Experiment No.3

### Keypad

#### Objective:

- How to interface with keypad.

#### Components:

- Keypad 4X4.
- 7 segments display.
- 7 segments decoder.

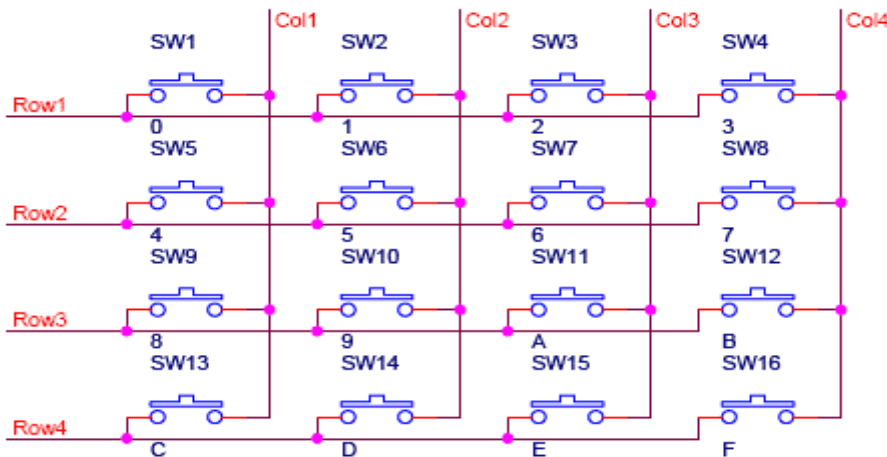
#### Keypad illustration:

The keypad is a group of push buttons connected in such a way that each push button can be recognized when it is pushed. The 4X4 keypad consists of 4 rows and 4 columns by pressing a push button a row and a column are connected identifying the push button like an address this connection is described as the following figure.

**Example:** when SW1 is pressed col1 and row1 are connected to each other.

This advantage we can use for checking which push button is pressed by checking which row is connected to which column.

**Example:** when col3 is connected to row2 so SW7 is pushed.

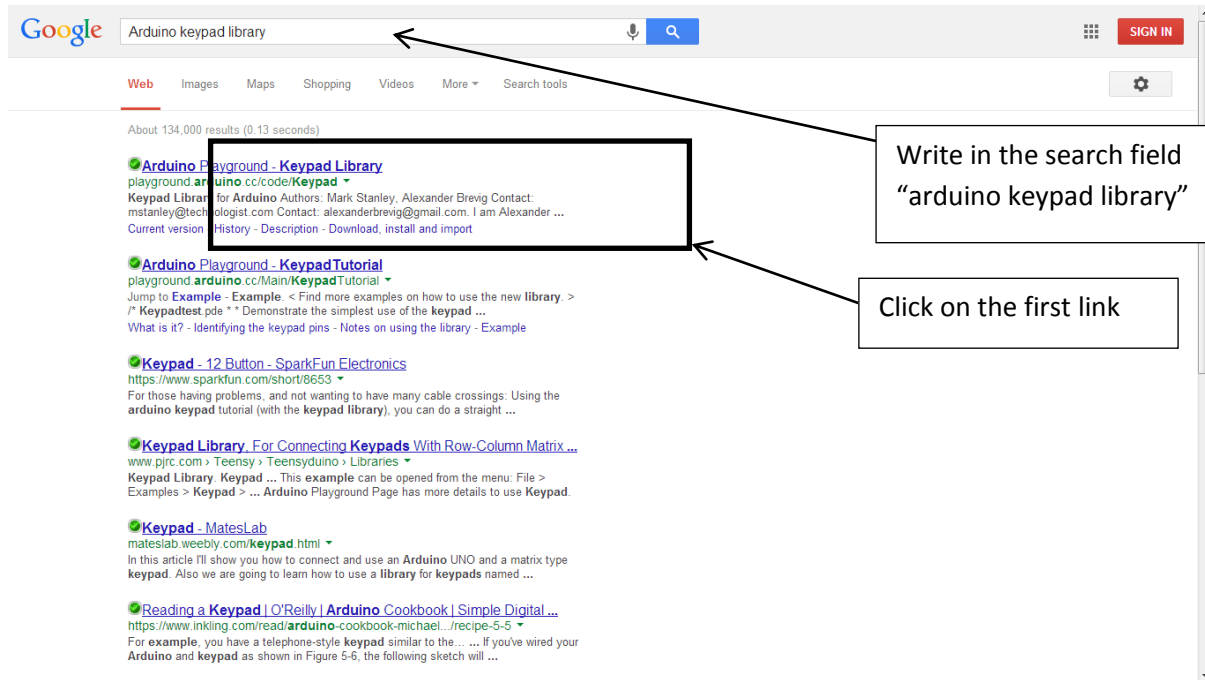


## Keypad library installation

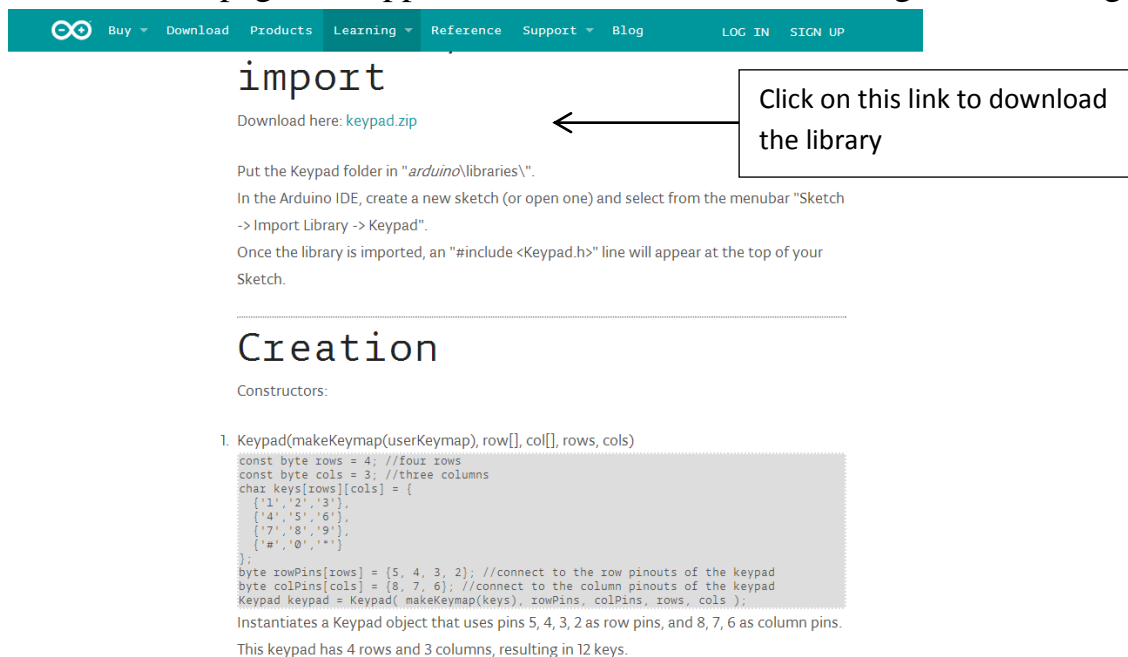
The Arduino IDE has some pre-installed installed libraries to work with and these libraries can be used and edited for the application required.

Keypad library is not installed so we will discuss how we can download a new library to Arduino from the Internet.

**First:** search for the desired library which will be Keypad library



**Second:** This page will appear then scroll down till reaching the following image.



Name : \_\_\_\_\_

Reg. No. : \_\_\_\_\_

**Third:** Extract the downloaded to a folder named “Keypad” then copy the folder and paste it to the destination where Arduino was downloaded in a folder named “Libraries”.

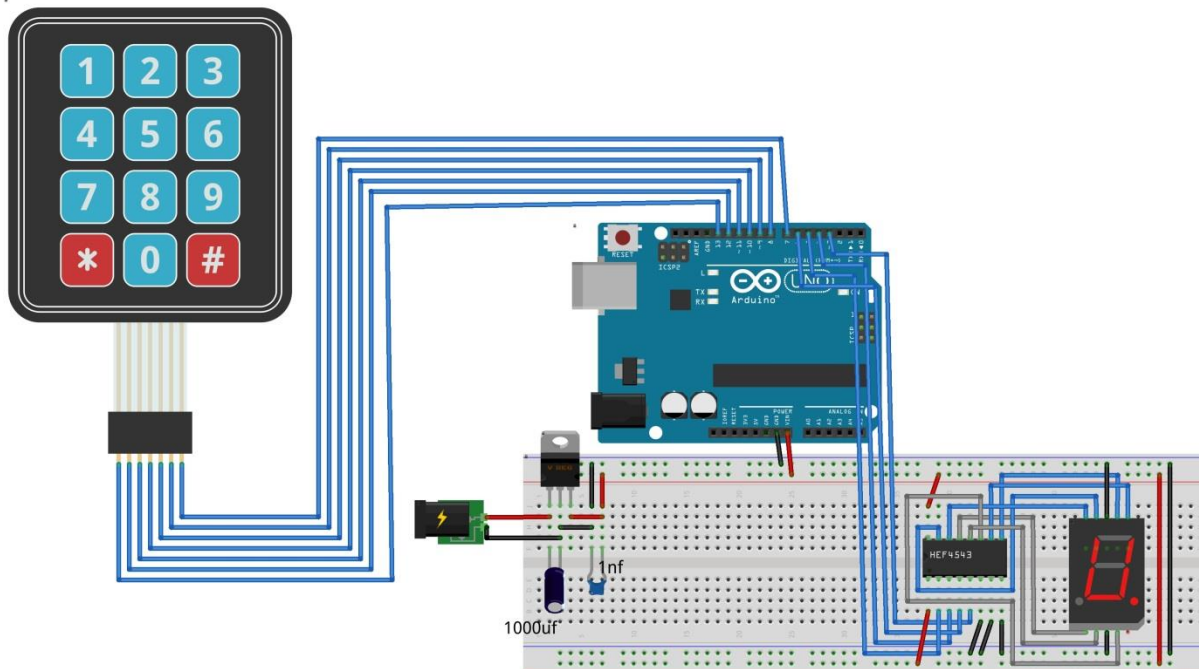
Example: The default destination after downloading and installing will be “C:\Program Files\Arduino\libraries” and if the computer’s processor is 64bit then the destination will be “C:\Program Files (x86)\Arduino\libraries” .

**Fourth:** The keypad library will appear in the Arduino program that it can be important as shown before.

### **Implementation:**

The 7 segments will display the number pressed by the keypad.

### **Wiring:**



For the 4X4 keypad the fourth column won't be connected which will be the fourth from the left.

**Program:**

```
#include <Keypad.h>
const byte ROWS = 4; //four rows
const byte COLS = 3; //four columns
//define the symbols on the buttons of the keypads
char hexaKeys[ROWS][COLS] = {
    {'1','2','3'},
    {'4','5','6'},
    {'7','8','9'},
    {'*','0','#'}
};
byte rowPins[ROWS] = {10, 9, 8, 7};
byte colPins[COLS] = {14, 13, 12, 11};
Keypad customKeypad = Keypad( makeKeymap(hexaKeys), rowPins, colPins, ROWS,
COLS);
void setup(){
    pinMode(2, INPUT);
    pinMode(3,OUTPUT);
    pinMode(4,OUTPUT);
    pinMode(5,OUTPUT);
    pinMode(6,OUTPUT);
}
void loop(){
    char customKey = customKeypad.getKey();

    if (customKey){
        switch(customKey)
        {
            case '0': zero();
            break;
            case '1': one();
            break;
            case '2': two();
            break;
```



```
        case '3': three();
        break;
        case '4': four();
        break;
        case '5': five();
        break;
        case '6': six();
        break;
        case '7': seven();
        break;
        case '8': eight();
        break;
        case '9': nine();
        break;
    }
}
}
void zero()
{  digitalWrite(3,0);
  digitalWrite(4,0);
  digitalWrite(5,0);
  digitalWrite(6,0);  }
void one()
{  digitalWrite(3,1);
  digitalWrite(4,0);
  digitalWrite(5,0);
  digitalWrite(6,0);  }
void two()
{
  digitalWrite(3,0);
  digitalWrite(4,1);
  digitalWrite(5,0);
  digitalWrite(6,0);
}
```

```
void three()
{
    digitalWrite(3,1);
    digitalWrite(4,1);
    digitalWrite(5,0);
    digitalWrite(6,0);
}
void four()
{
    digitalWrite(3,0);
    digitalWrite(4,0);
    digitalWrite(5,1);
    digitalWrite(6,0);
}
void five()
{
    digitalWrite(3,1);
    digitalWrite(4,0);
    digitalWrite(5,1);
    digitalWrite(6,0);
}
void six()
{
    digitalWrite(3,0);
    digitalWrite(4,1);
    digitalWrite(5,1);
    digitalWrite(6,0);
}
void seven()
{    digitalWrite(3,1);
    digitalWrite(4,1);
    digitalWrite(5,1);
    digitalWrite(6,0); }
void eight()
```

```
{  
    digitalWrite(3,0);  
    digitalWrite(4,0);  
    digitalWrite(5,0);  
    digitalWrite(6,1);  
}  
void nine()  
{  
    digitalWrite(3,1);  
    digitalWrite(4,0);  
    digitalWrite(5,0);  
    digitalWrite(6,1);  
}
```

**Questions:**

- Redo the experiments using 4X4 Keypad

## Experiment no.4

### Liquid crystal display

#### Objective:

Interface with Liquid Crystal display.

#### Components:

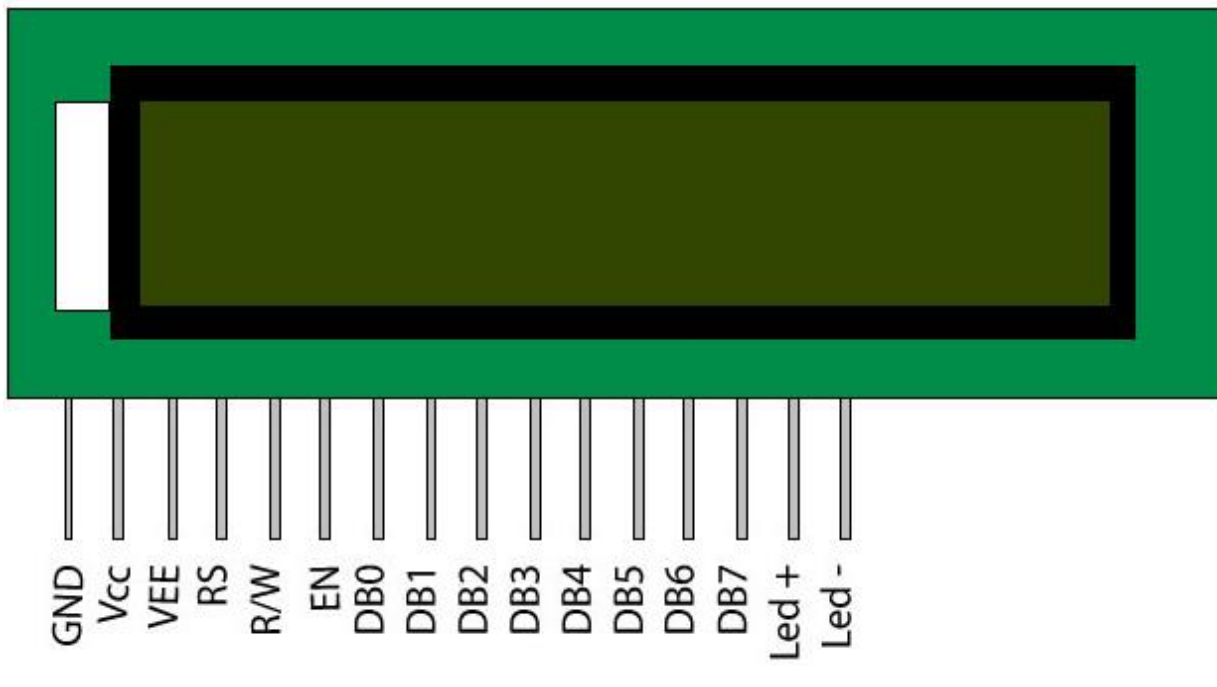
- LCD 16X2.
- Potentiometer 10K.

#### LCD 16X2 description:

A 16x2 LCD means it can display 16 characters per line and there are 2 such lines. In this LCD each character is displayed in 5x7 pixel matrix. This LCD has two registers, namely, Command and Data.

The command register stores the command instructions given to the LCD. A command is an instruction given to LCD to do a predefined task like initializing it, clearing its screen, setting the cursor position, controlling display etc. The data register stores the data to be displayed on the LCD. The data is the ASCII value of the character to be displayed on the LCD. Click to learn more about internal structure of a LCD.

#### Pin Diagram:



**Pin Description:**

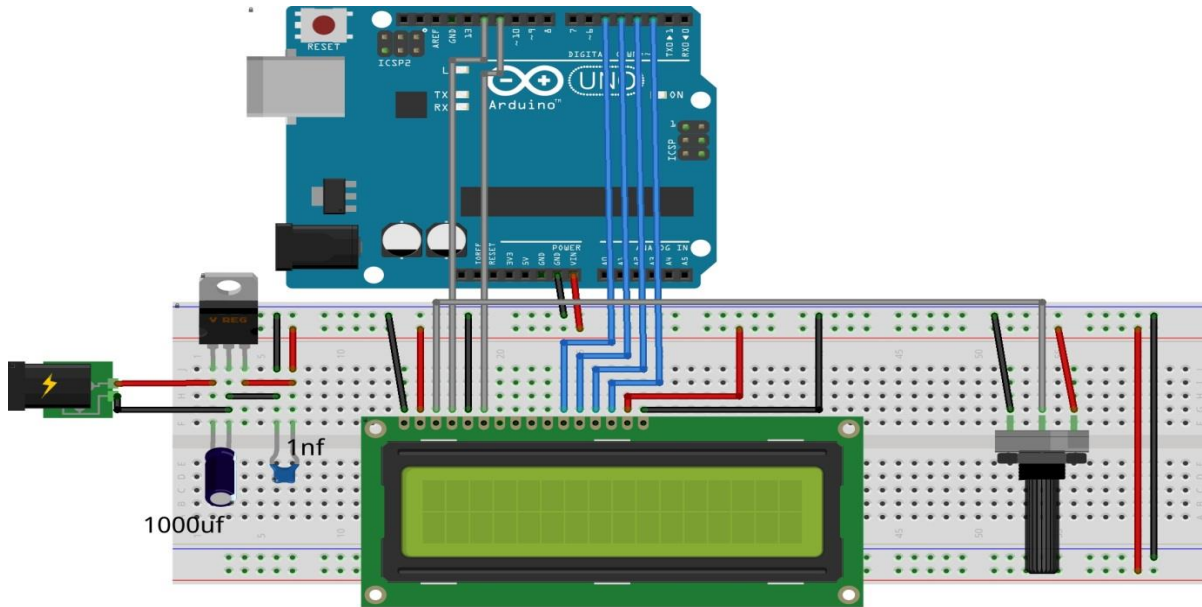
Pin No	Function	Name
1	Ground (0V)	Ground
2	Supply voltage; 5V (4.7V – 5.3V)	V <sub>cc</sub>
3	Contrast adjustment; through a variable resistor	V <sub>EE</sub>
4	Selects command register when low; and data register when high	Register Select
5	Low to write to the register; High to read from the register	Read/write
6	Sends data to data pins when a high to low pulse is given	Enable
7	8-bit data pins	DB0
8		DB1
9		DB2
10		DB3
11		DB4
12		DB5
13		DB6
14		DB7
15	Backlight V <sub>CC</sub> (5V)	Led+
16	Backlight Ground (0V)	Led-

**Implementation:**

Arduino will be connected to the LCD to learn about LCD interface and to be familiar with the Pins.

Name : \_\_\_\_\_

Reg. No. : \_\_\_\_\_

**Wiring:****Program:**

```
#include <LiquidCrystal.h>
// initialize the library with the numbers of the interface pins
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

void setup() {
  // set up the LCD's number of columns and rows:
  lcd.begin(16, 2);
  // Print a message to the LCD.
  lcd.print("hello, world!");
}

void loop() {
  // set the cursor to column 0, line 1
  // (note: line 1 is the second row, since counting begins with 0):
  lcd.setCursor(0, 1);
}
```

**Questions:**

- Type another sentence on the LCD.
- Connect the LCD with different pins in the Arduino.

## Experiment no.5

### Liquid crystal display commands

**Objective:**

Get familiar with LCD commands.

**Important Commands:**

- clear()
- home()
- setCursor()
- print()
- noDisplay()
- display()

**Description:**

Use each of the previous commands and fill in the following results table

**Results:**

Command	Use	Sketch	
		Before	After
clear()			
home()			
setCursor()			
print()			
noDisplay()			
diplay()			

Name : \_\_\_\_\_

Reg. No. : \_\_\_\_\_

## **References**

- <http://www.arduino.cc/>
- <http://startingelectronics.com/beginners/components/7-segment-display/>
- <http://macao.communications.museum/eng/exhibition/secondfloor/moreinfo/Displays.html>
- 74LS47 Datasheet
- HEF3543B Datasheet
- [http://www.societyofrobots.com/electronics\\_led\\_tutorial.shtml](http://www.societyofrobots.com/electronics_led_tutorial.shtml)
- <http://www.8051projects.net/keypad-interfacing/introduction.php>
- <http://www.engineersgarage.com/electronic-components/16x2-lcd-module-datasheet>