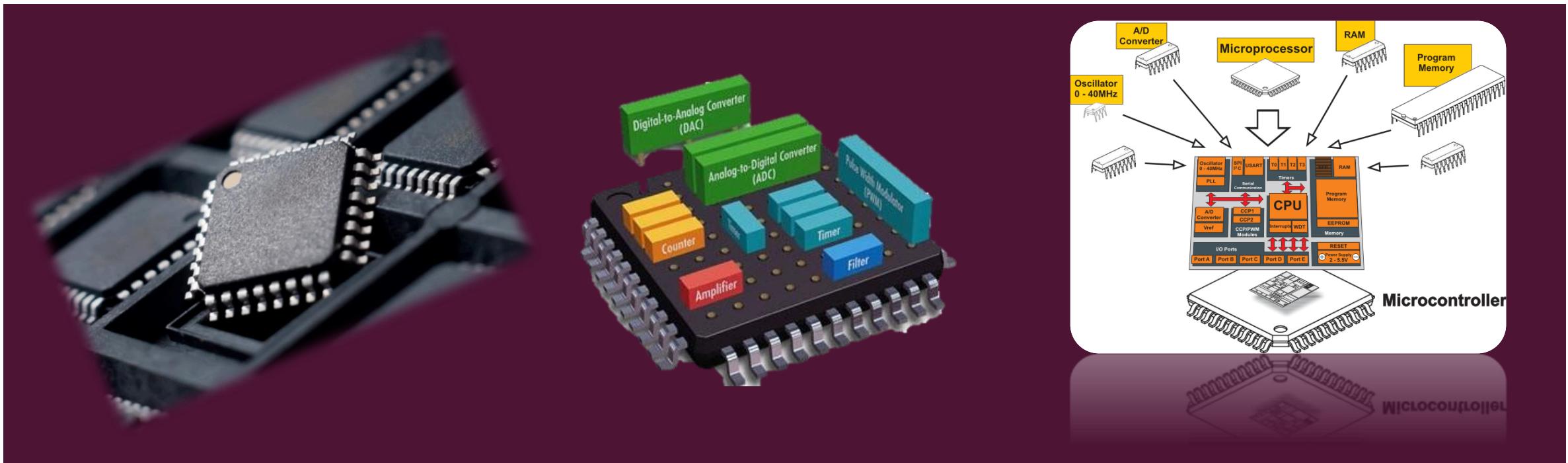


PIC MICROCONTROLLER LABS

Eng. Elaf A.Saeed



Content

01 Embedded System.

02 Microcontrollers Types & Their Applications.

03 Software & Hardware.

04 PIC Features.

05 MikroC.

06 Experiments.

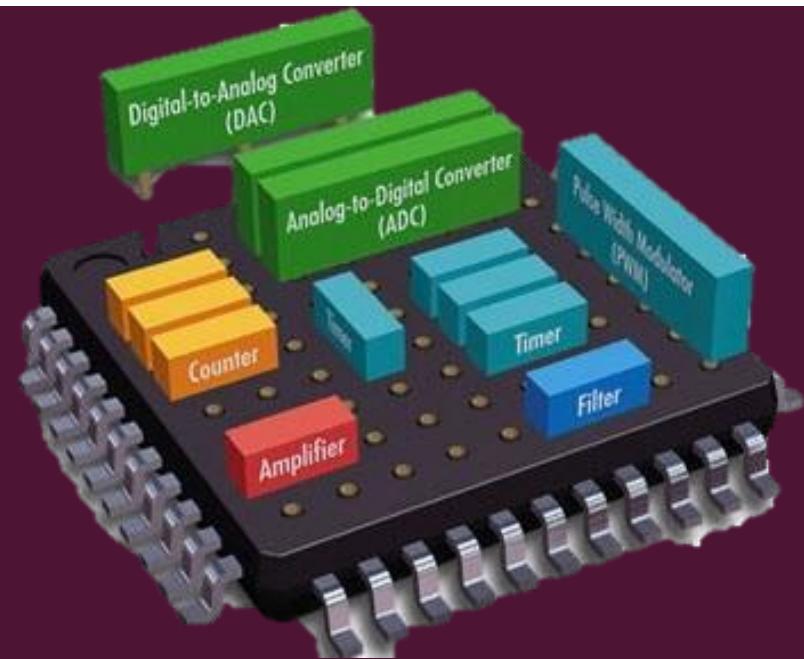
Introduction

In this book, a set of lessons related to microcontrollers and embedded systems and a set of experiments were made using the pic microcontroller. Beginners and those interested in the field of embedded systems and the use of microcontrollers can benefit from these lessons.

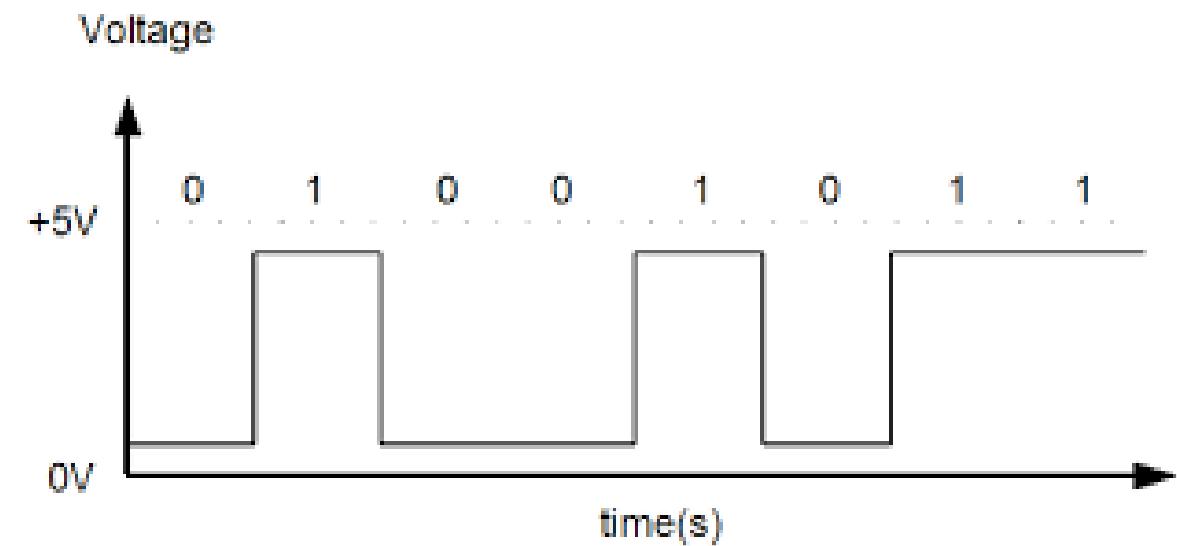
PIC MICROCONTROLLER

01 EMBEDDED SYSTEM

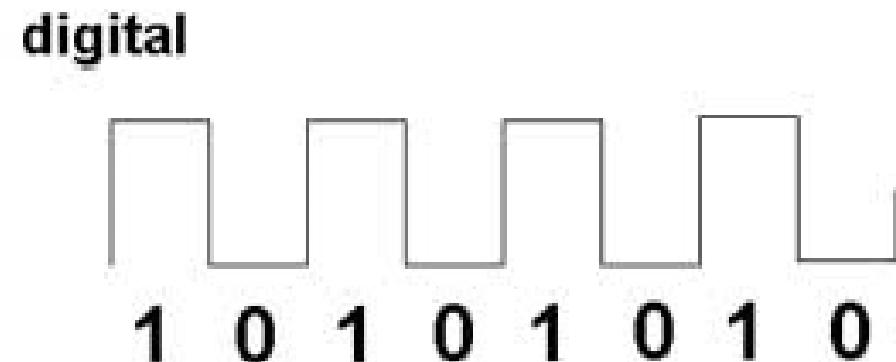
Eng. Elaf A.Saeed



- Changing the digital signal with time between two values:
 - The first value is **HIGH** and the second value is **LOW**.
- Usually the value **High** is expressed as **1**, while the number **0** is used to express the value **LOW**.



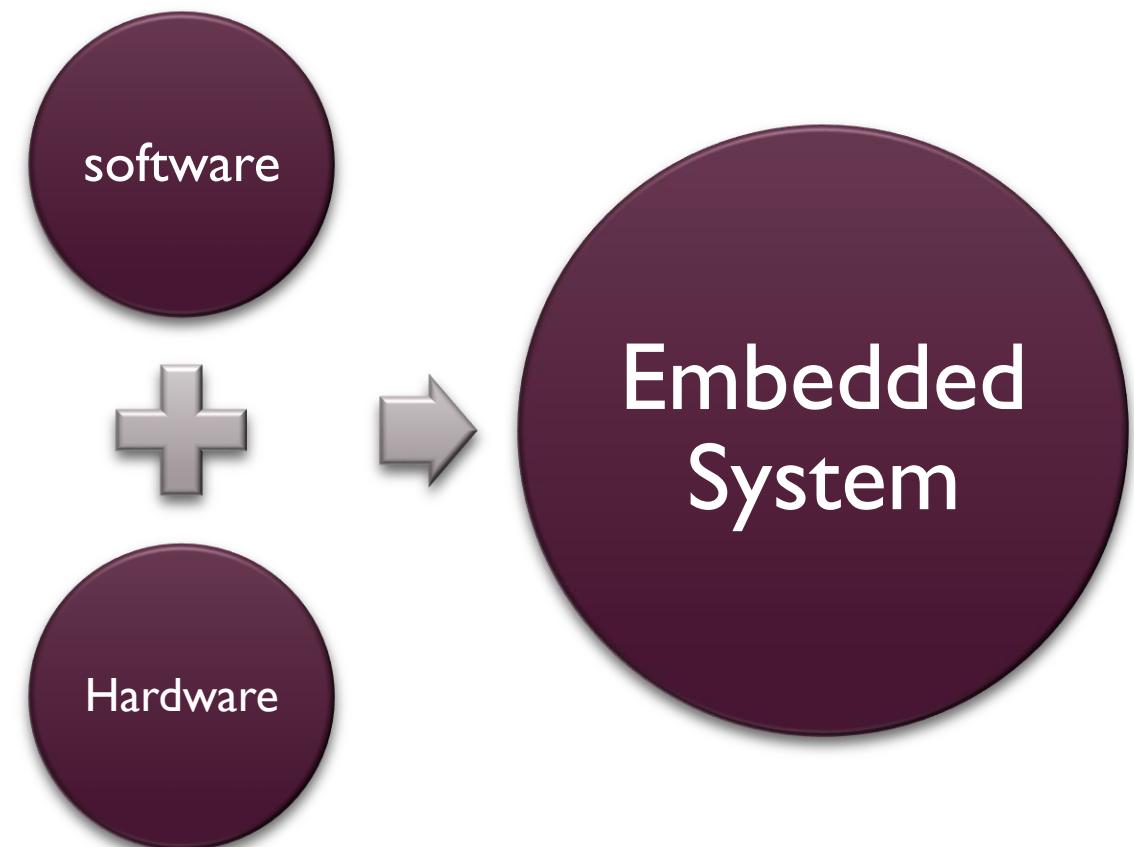
- This property can be used to create **information/statements** by merging a group of zeros and ones.
- By alternating zeros and ones, many pieces of **information/data** can be formed.



- As long as all **data/information** can be formed from **zeros** and **ones**. It is possible to change the **statement / information** by changing the zeros and ones that are composed of it.
- That is, it is **programmable**.



- Programmable digital signal → Software.
- System using software → Hardware.



■ **Embedded System**

An **Embedded System** is a computer system designed to perform one or a few dedicated functions.

■ Embedded Systems: Applications

- Mp4 Players.
- Video Game.
- Digital Camera.
- Microwave oven.
- Dishwasher.

Consumer
Electronics



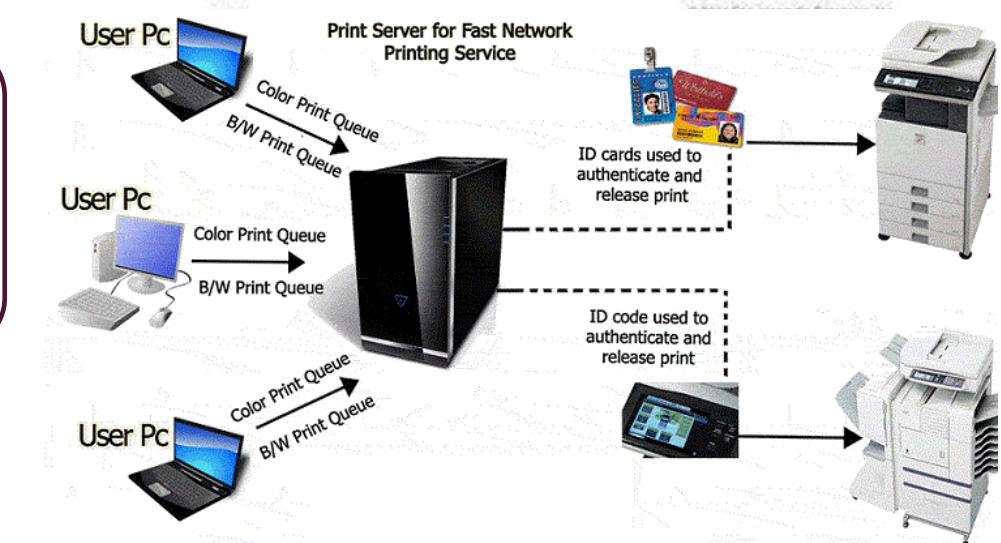
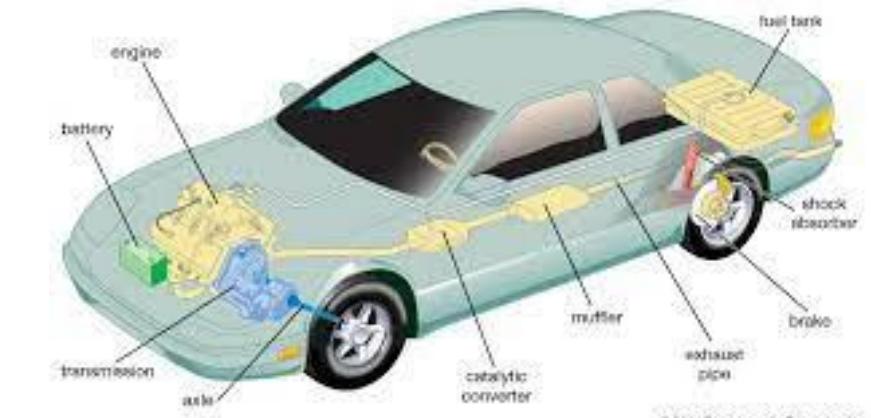
■ Embedded Systems: Applications

Automobiles

- Engine control.
- Air bags.
- Anti-lock braking system (ABS).

Office automation

- Copiers.
- Printer.
- FAX machine.



- **Embedded Systems: Applications**

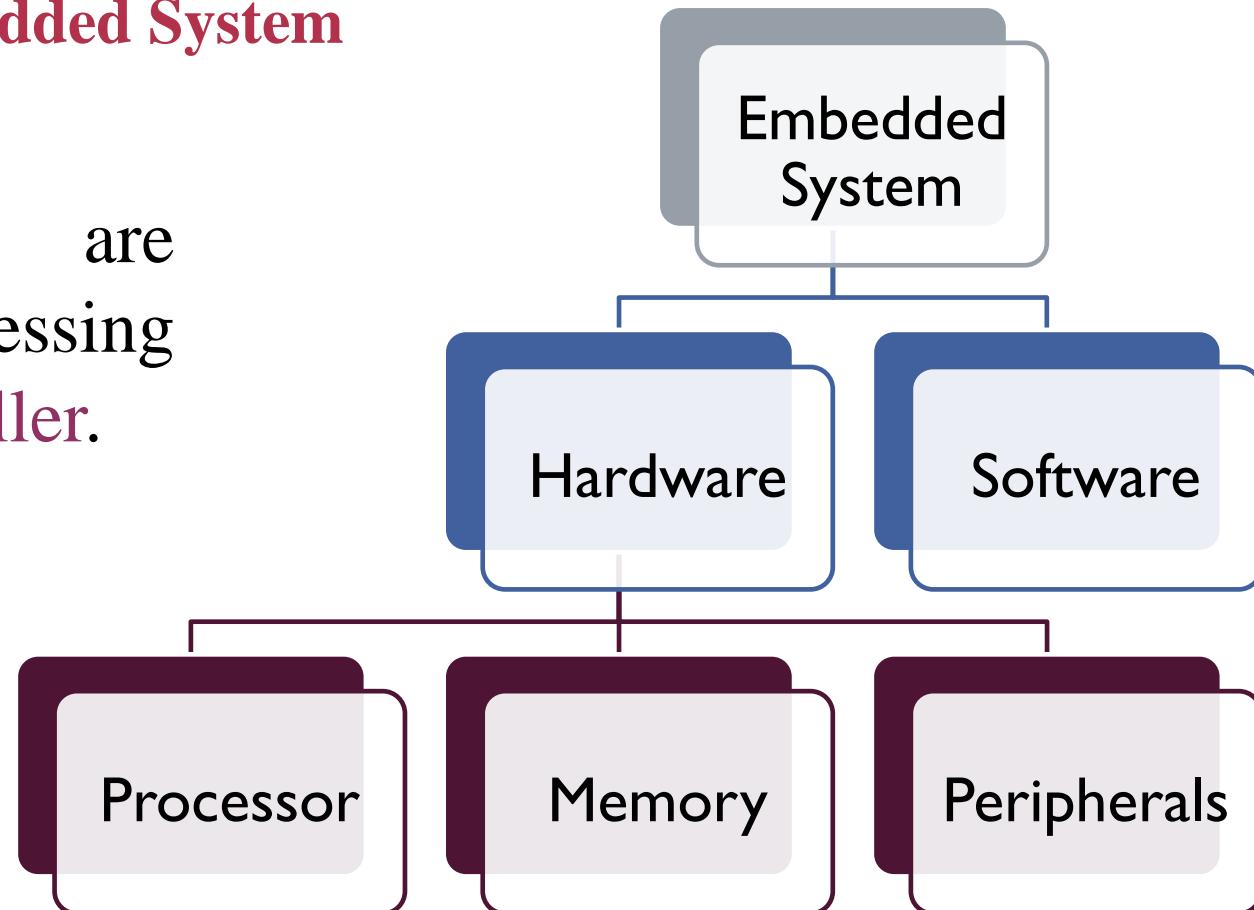


- Cellular phones.
- GPS receivers.



■ Basic Elements of an Embedded System

Embedded systems are controlled by a main processing core such as a **microcontroller**.



■ What is a Microcontroller?

A **microcontroller** is a digital chip system which contains the basic following elements:



1

- A processor core

2

- Data and program memory.

3

- Serial and parallel I/o.

4

- Timers.

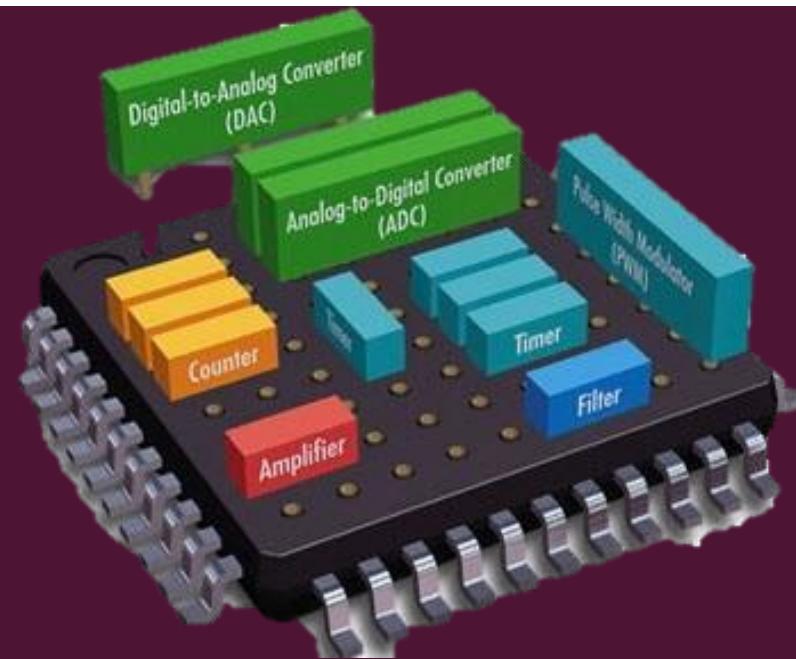
5

- External and Internal interrupt handing mechanism.

PIC MICROCONTROLLER

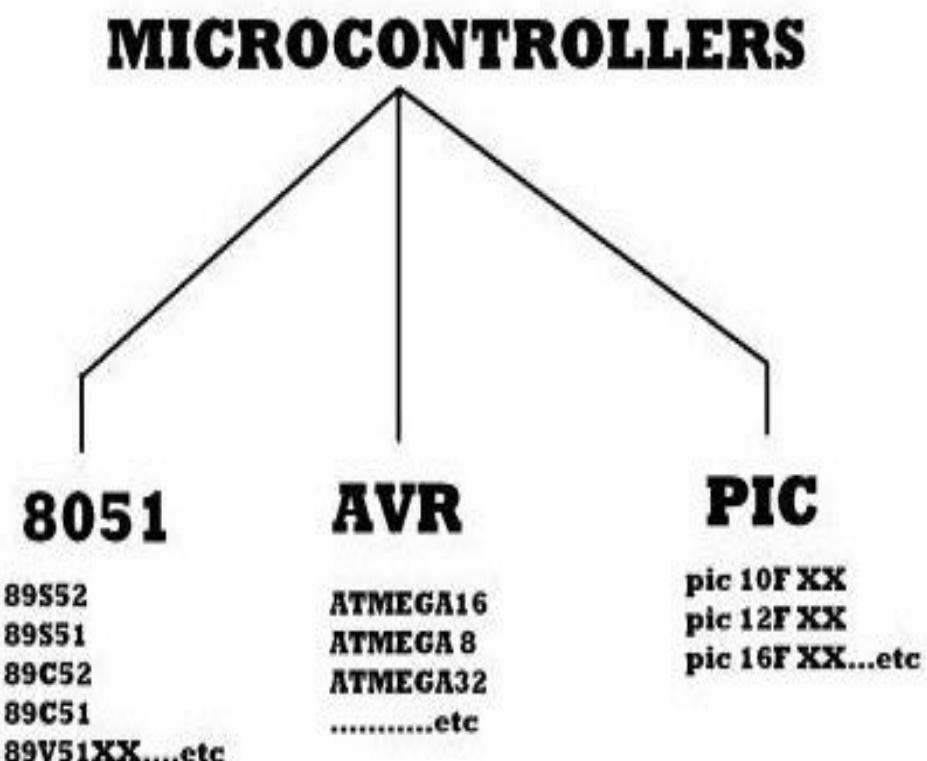
02 MICROCONTROLLERS TYPES & THEIR APPLICATIONS

Eng. Elaf A.Saeed



■ How are Microcontrollers Types Classified?

- The microcontrollers are characterized regarding bus-width, instruction set, and memory structure. For the same family, there may be different forms with different sources.
- The types of the microcontroller are shown in the figure, they are characterized by their bits, memory architecture, memory/devices, and instruction set.
- There are different microcontroller types like 8051, PIC, AVR, ARM



1. Microcontrollers Types According to the Number of Bits

- The bits in the microcontroller are 8-bits, 16-bits, and 32-bits microcontroller.
- In an **8-bit** microcontroller, the point when the internal bus is 8-bit then the ALU performs the arithmetic and logic operations. The examples of 8-bit microcontrollers are Intel 8031/8051, PIC1x, and Motorola MC68HC11 families.
- The **16-bit** microcontroller performs greater precision and performance as compared to the 8-bit. For example, 8-bit microcontrollers can only use 8 bits, resulting in a final range of $0\times 00 - 0xFF$ (0-255) for every cycle. In contrast, 16-bit microcontrollers with their bit data width have a range of $0\times 0000 - 0xFFFF$ (0-65535) for every cycle.
- The **32-bit** microcontroller uses the 32-bit instructions to perform the arithmetic and logic operations. These are used in automatically controlled devices including implantable medical devices, engine control systems, office machines, appliances, and other types of embedded systems. Some examples are Intel/Atmel 251 family, PIC3x.

2. Microcontrollers Types According to Memory Devices

The memory devices are divided into two types, they are:

- Embedded memory microcontroller.

Embedded Memory Microcontroller: When an embedded system has a microcontroller unit that has all the functional blocks available on a chip is called an **embedded microcontroller**.

- External memory microcontroller.

External Memory Microcontroller: When an embedded system has a microcontroller unit that has not all the functional blocks available on a chip is called an external memory microcontroller. For example, 8031 has no program memory on the chip is an **external memory microcontroller**.

4. Microcontrollers Types According to Instruction Set

- **CISC:** is a *Complex Instruction Set Computer*. It allows the programmer to use one instruction in place of many simpler instructions.
- **RISC:** The RISC stands for *Reduced Instruction set Computer*, this type of instruction sets reduces the design of microprocessor for industry standards. It allows each instruction to operate on any register or use any addressing mode and simultaneous access of program and data.

5. Microcontrollers Types According to Memory Architecture

- The memory architecture of microcontroller are two types, they are namely:

A. Harvard memory architecture microcontroller

Harvard Memory Architecture Microcontroller: The point when a microcontroller unit has a dissimilar memory address space for the program and data memory, the microcontroller has Harvard memory architecture in the processor.

B. Princeton memory architecture microcontroller

Princeton Memory Architecture Microcontroller: The point when a microcontroller has a common memory address for the program memory and data memory, the microcontroller has Princeton memory architecture in the processor.



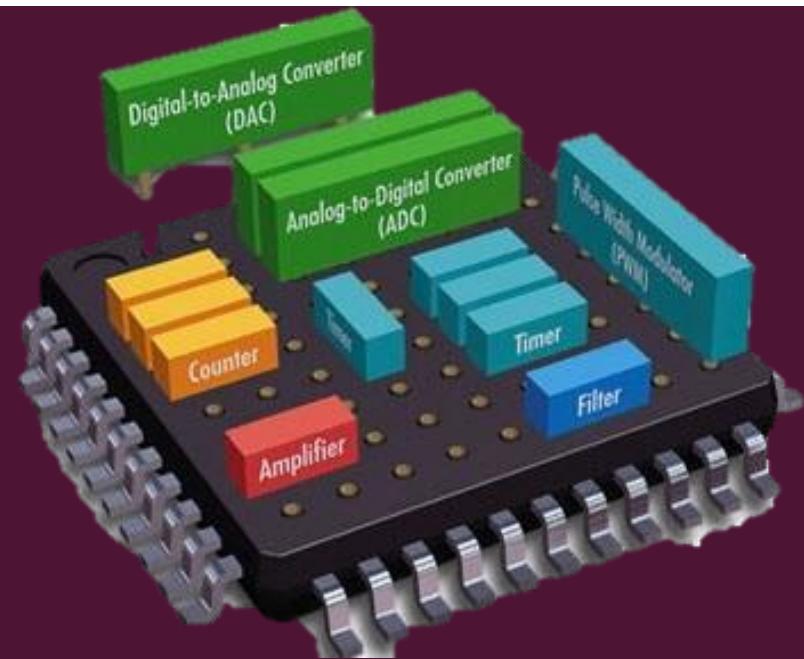
■ Microcontrollers Types

- There are different microcontroller types like 8051, PIC, AVR, ARM.
- ❖ We will use the PIC Microcontroller in experiments.

PIC MICROCONTROLLER

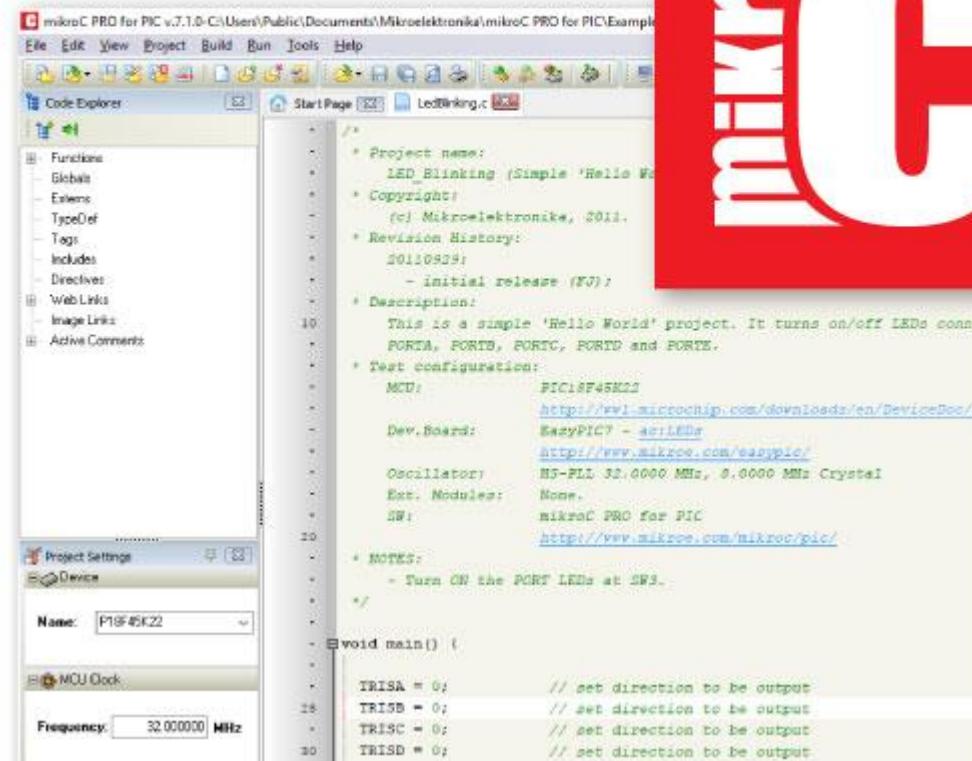
03 SOFTWARE & HARDWARE

Eng. Elaf A.Saeed



❖ Software

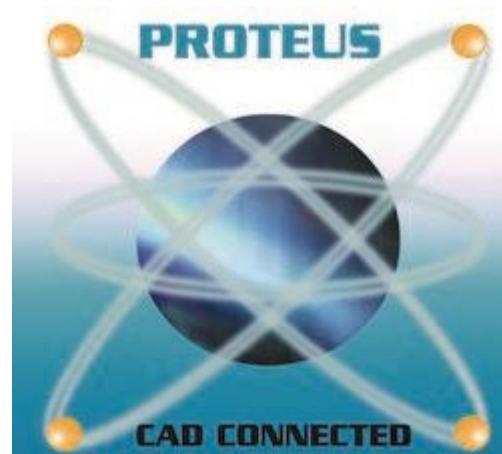
■ MikroC IDE



The screenshot shows the MikroC PRO for PIC v.7.1.0 IDE interface. The main window displays a C code file named 'LedBlinking.c' with comments describing a 'Hello World' project for a PIC18F45K22 microcontroller. The code includes definitions for PORTA, PORTB, PORTC, PORTD, and PORTE, and a main function setting the direction of pins TRISA, TRISS, TRISC, and TRISD to output. Below the code editor, the 'Project Settings' pane is open, showing the selected device as 'PIC18F45K22' and the oscillator frequency as '32.00000 MHz'. The 'Code Explorer' pane on the left lists various project components like functions, global variables, and includes.

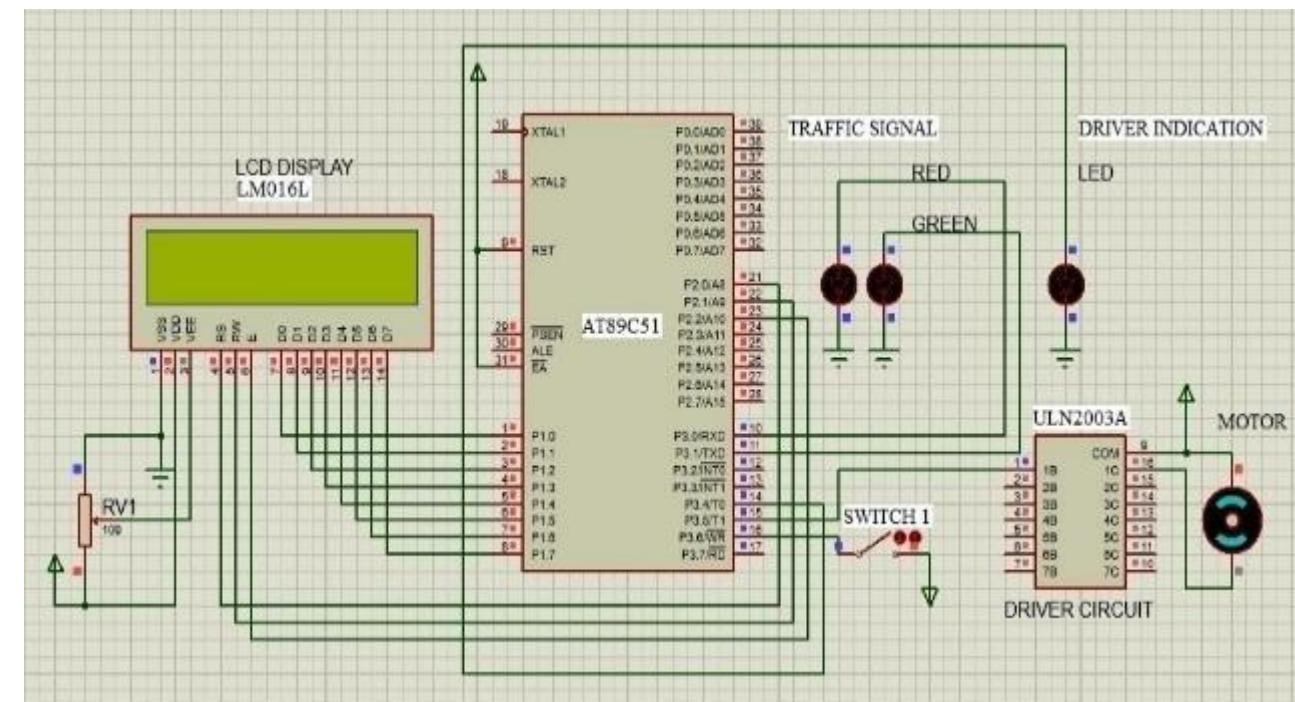
```
/* Project name:  
 * LED_Blinking (Simple 'Hello World'  
 * Copyright:  
 * (c) MikroElektronika, 2011.  
 * Revision History:  
 * 20110929  
 * - initial release (EJ)  
 * Description:  
 * This is a simple 'Hello World' project. It turns on/off LEDs connected  
 * to PORTA, PORTB, PORTC, PORTD and PORTE.  
 * Test configuration:  
 * MCU: PIC18F45K22  
 * http://www.microchip.com/downloads/en/DeviceDoc/41  
 * Dev.Board: EasyPIC7 - aciLEDDs  
 * http://www.mikroe.com/easypic/  
 * Oscillator: HS-PLL 32.0000 MHz, 8.0000 MHz Crystal  
 * Ext. Modules: None.  
 * SW: mikroC PRO for PIC  
 * http://www.mikroe.com/mikroc/pic/  
 *  
 * NOTES:  
 * - Turn ON the PORT LEDS at SW3.  
 */  
  
void main() {  
    TRISA = 0; // set direction to be output  
    TRISS = 0; // set direction to be output  
    TRISC = 0; // set direction to be output  
    TRISD = 0; // set direction to be output
```





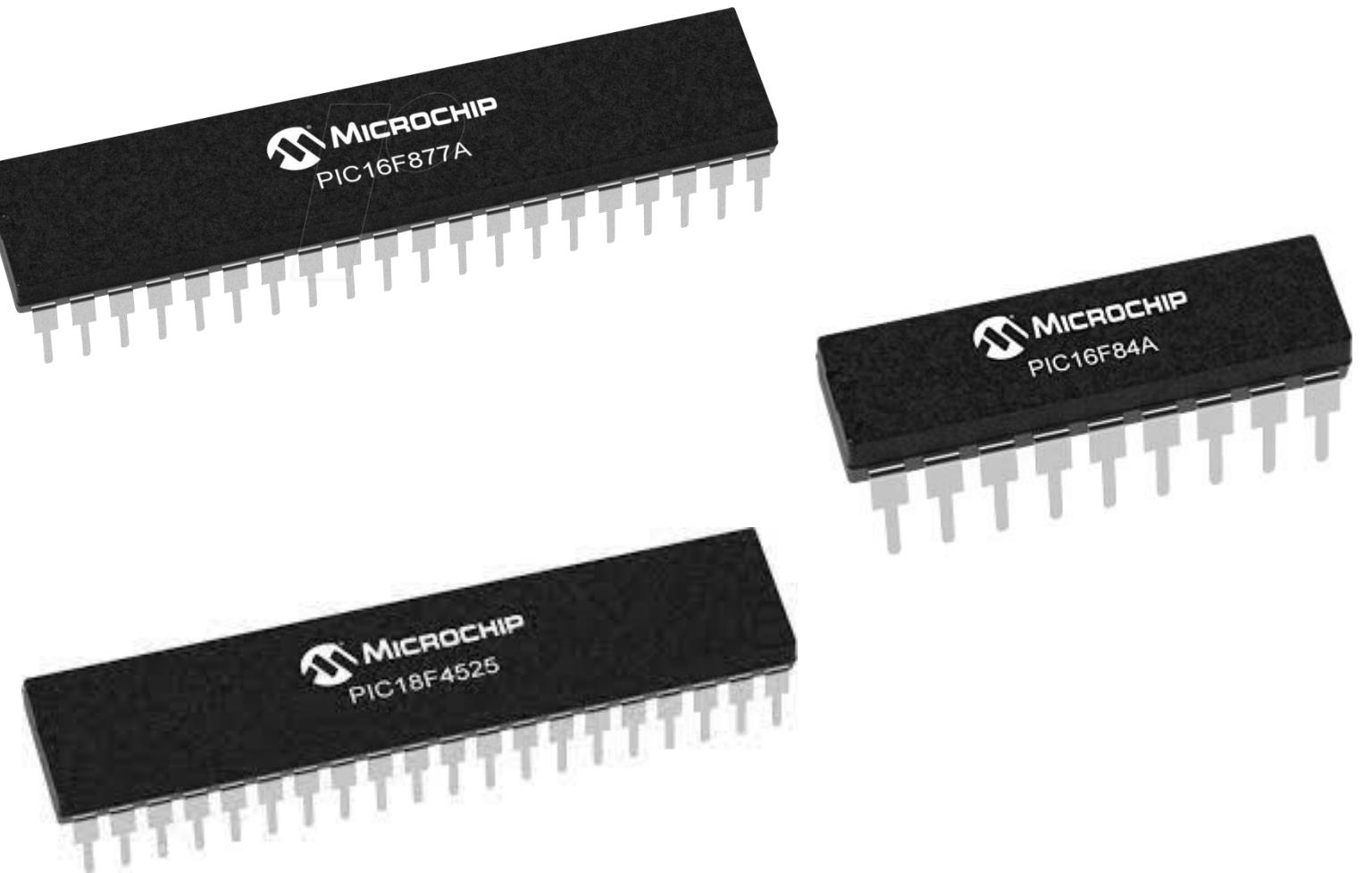
❖ Software

■ Protues Simulation



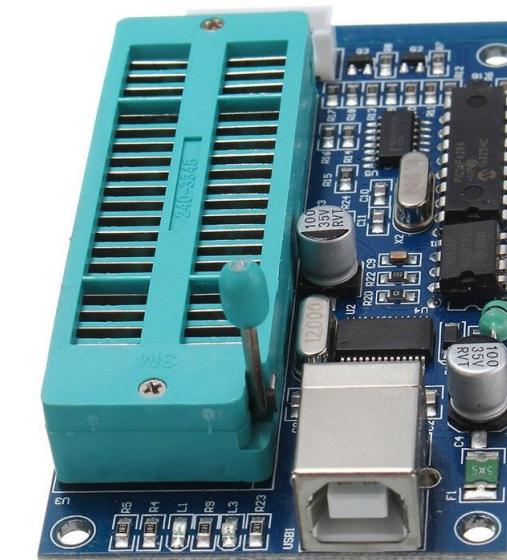
❖ Hardware Components

1. PIC Microcontroller



❖ **Hardware Components**

2. programmer



❖ **Hardware Components**

3. Crystal

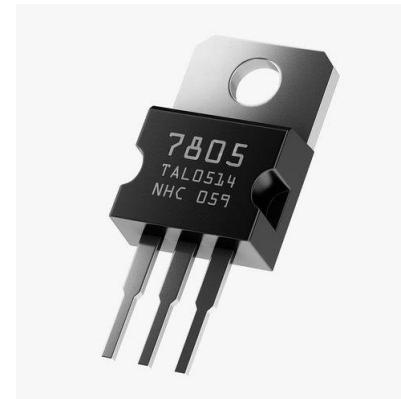


4. Capacitor 20-30 pF



❖ Hardware Components

5. Regulator



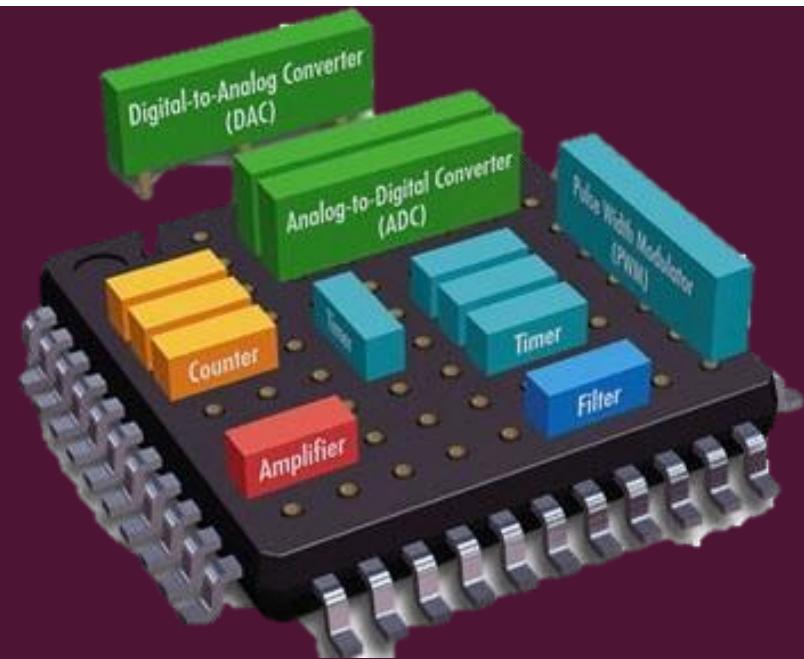
6. Wires



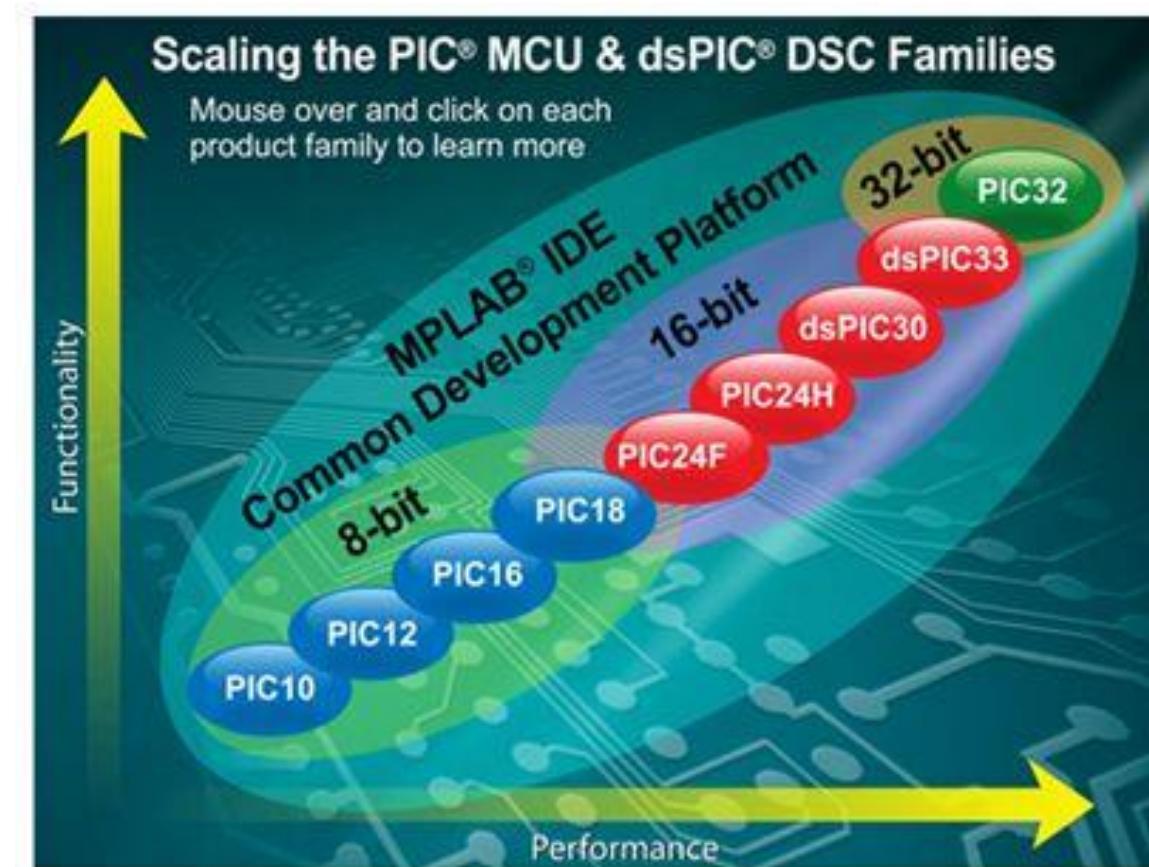
PIC MICROCONTROLLER

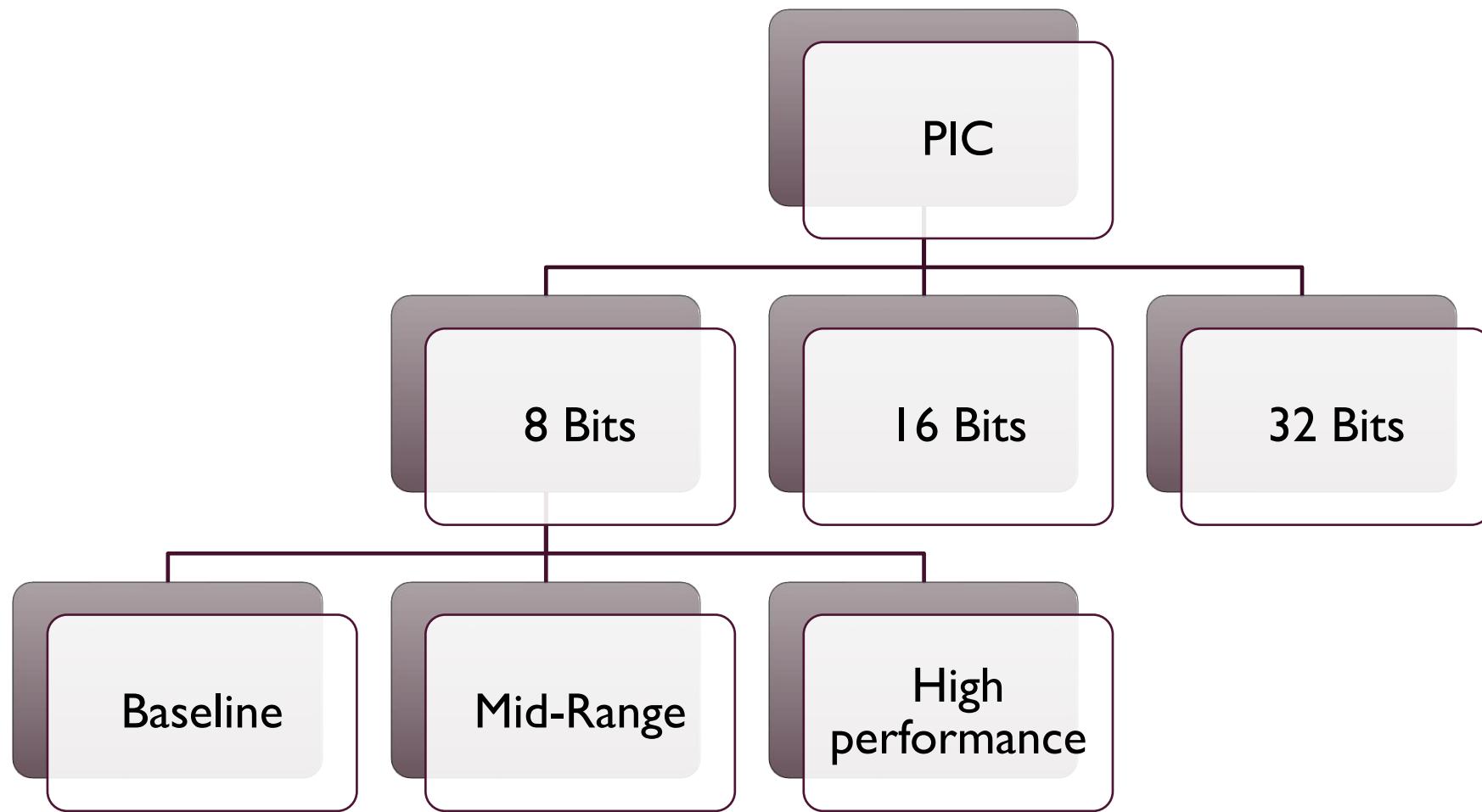
04 PIC FEATURES

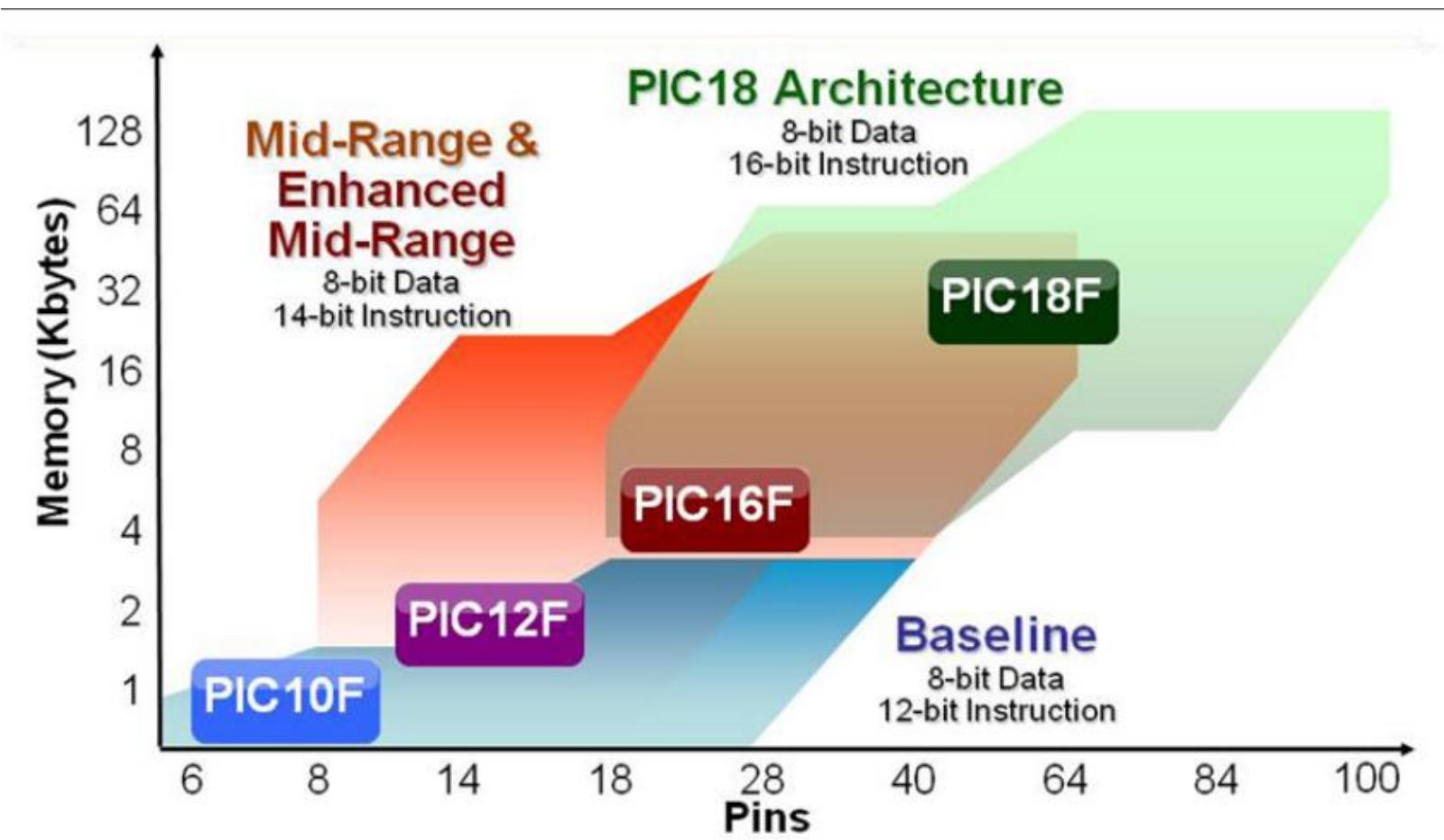
Eng. Elaf A.Saeed



- Microchip Technology Company is one of the most famous and most famous companies producing microcontrollers, and the types it produces are called **PIC** for *Peripheral Interface Controller*, or *Programmable Interface Controller*, and they produce several types of families that differ in capabilities and sizes, as the corresponding figure shows.

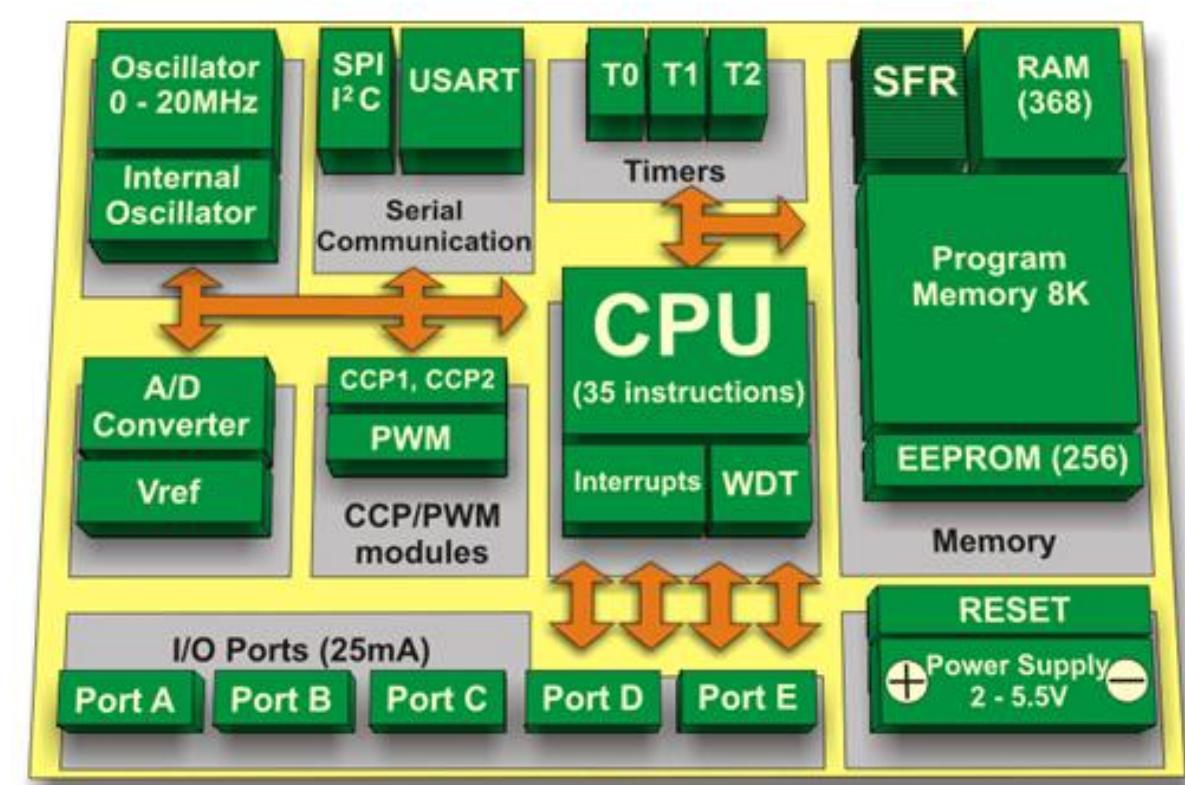






- In general, a PIC chip can contain all or some of the following units:

1. Memory Units.
2. Central Processing Unit.
3. Input – Output unit.
4. Timer unit.
5. Watchdog Timer (WDT) unit.
6. Analog to Digital converter unit.
7. Serial Communication port.
8. Analog Comparator.
9. CAN (Controller Area Network) interface.
10. USB interface.
11. Ethernet interface.

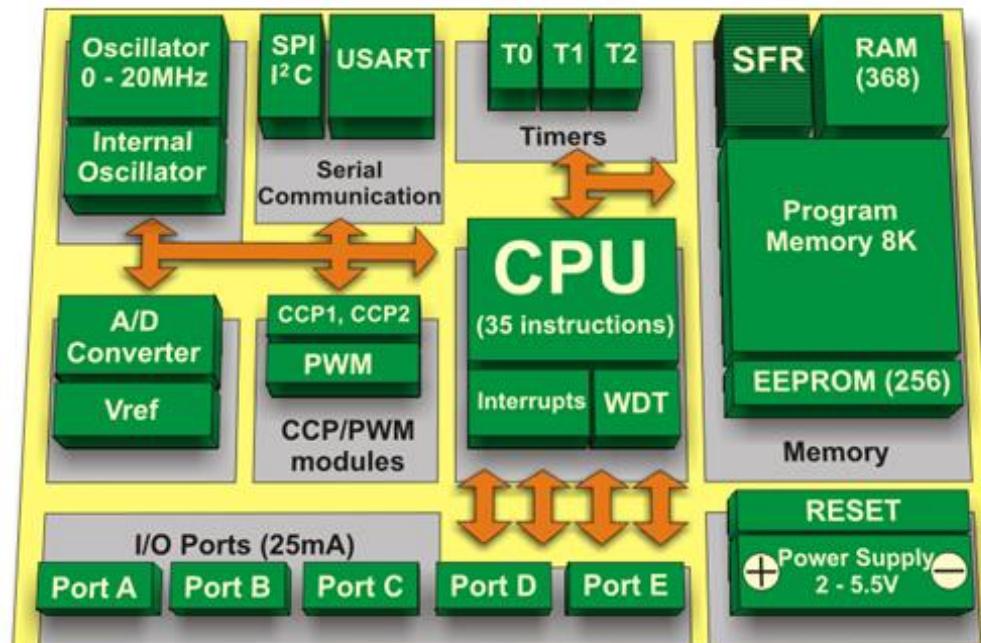


1. Memory Unit

- The purpose of memory is to **store data**, and there are several types of memory found in the PIC, each with a specific purpose, which are:

A. Flash Program Memory

And it is used to store the program that is written in one of the languages and that controls the work of the PIC, and even if the power is cut off from the circuit that feeds the PIC, the program will still be stored in it. The program can be rewritten for a number of times up to 100,000 times.

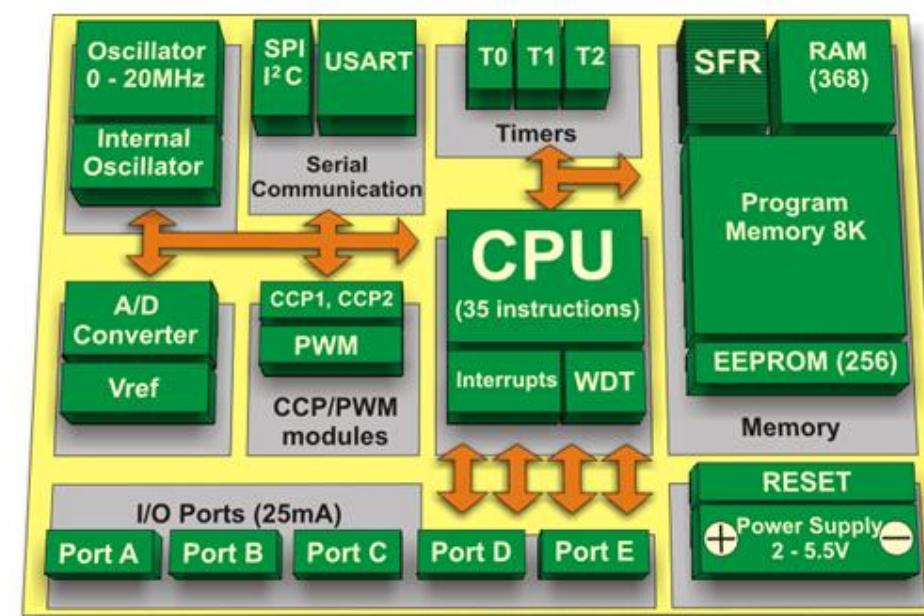


B. Random Access Memory (RAM) or (Data Memory)

This memory is used to store temporary data that results from the execution of the program, and of course, what data is lost in the event of a power outage.

C. Electrically Erasable Programmable Read Only Memory (EEPROM)

The contents of this memory remain in it even after the power is cut off from it, and it is written and read from it using the program, and its size is often small, and it can be written to a number of times up to a million times, and therefore it is used to store the data that the program needs. It does not change continuously and the data can be stored for a period of time up to 40 years.



2. Center Processing Unit (CPU)

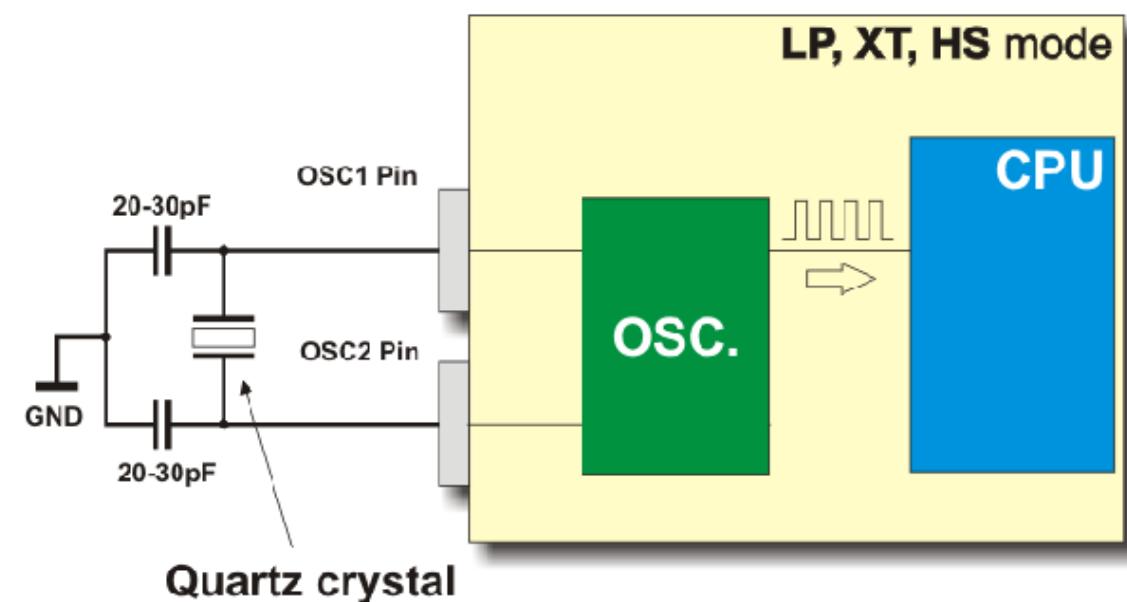
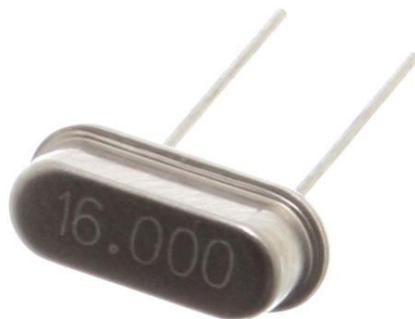
This unit is responsible for executing the program written in one of the programming languages, which is stored in flash program memory. This unit performs arithmetic and logical operations.

3. Input – Output Unit

The pins around the PIC are a means of communication with the outside world, and these terminals are connected to input and output units called **ports**, and the number of these ports varies from one chain to another, and each port is connected to a certain number of terminals, and each port takes a symbol indicating it. Such as **Port A**, **Port B**, **Port C**, and one party can be assigned to more than one function that can perform for the PIC, but only one function can be performed at a time, and through the port it is possible to determine whether a pin is a pin It is used to receive data or output data from the PIC.

4. Oscillator (clock generator)

In order for the CPU to read and execute the command in the program, and then move on to reading and executing the next command, there must be something using it to do so, and this role is played by the oscillator that generates regular pulses that operate the microcontroller. There are several patterns used to characterize the type and speed of the oscillator used. We review the following:



A. HS (High Speed Crystal / Resonator)

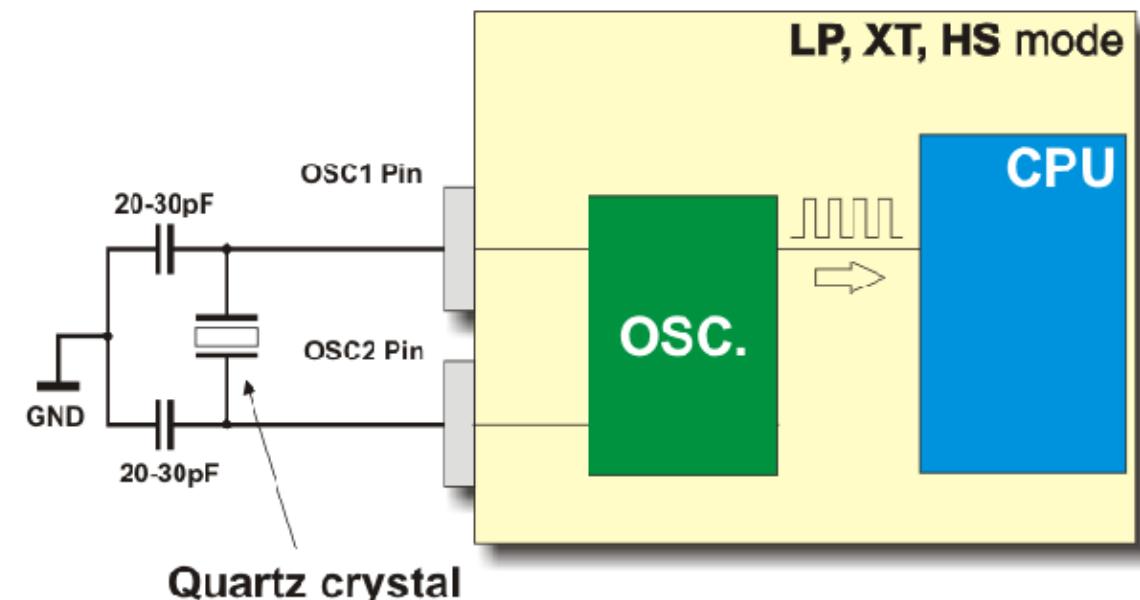
It is used with crystal oscillators that have a frequency of **4 MHZ** or more.

B. XT (Crystal / Resonator)

It is used with crystal with a frequency from 1 MHZ to 4 MHZ.

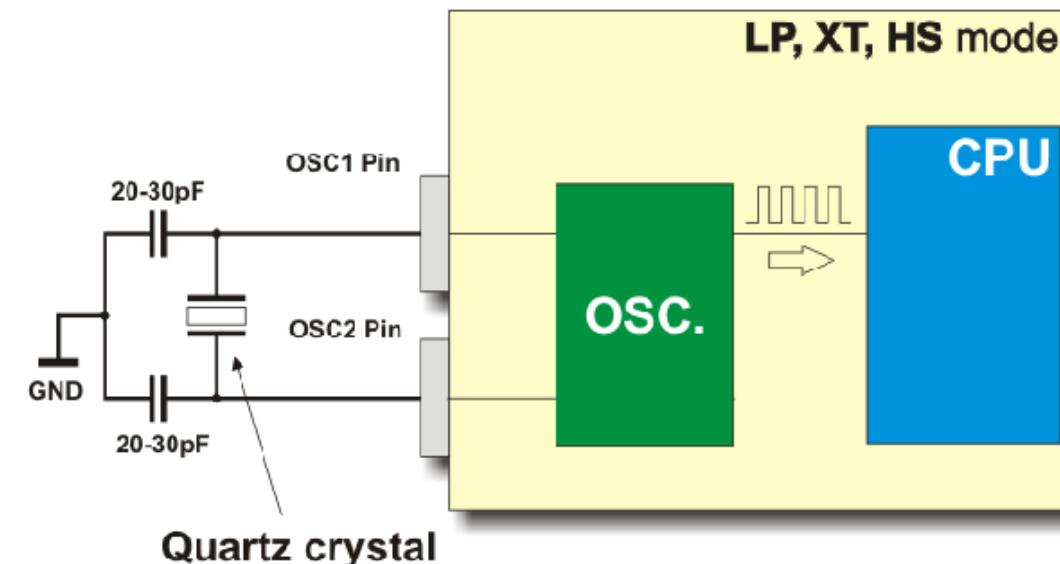
C. LP (Low power)

Used with crystal for frequency less than 200KHZ.



The two ends of the crystal are connected to the terminals of **osc1** and **osc2**, and each of them is a pin located on the body of the PIC, and we note that there are **two capacitors** connected to the oscillator to prevent interference, and the following table shows the values of the capacitor that are used with **different frequencies**.

OSC type	Frequency	Capacitor range
LP	32 KHZ	33 pF
	200 KHZ	15 pF
XT	200 KHZ	47 – 68 pF
	1 MHZ	15 pF
HS	4 MHZ	15 pF
	4 MHZ	15 pF
HS	8 MHZ	15 – 33 pF
	20 MHZ	15 – 33 pF



D. RC (External Resistor / Capacitor)

This mode is used with applications that require cost savings with no need for precise timing, in which a capacitor and a resistor are used to obtain a frequency that is less than 1MHZ, bearing in mind that if the resistance is less than $2.2\text{ k}\Omega$, the oscillator is unstable and it can stop, just as with resistors with high values of $100\text{ k}\Omega$ and more, the oscillator becomes very sensitive to the surrounding noise, so the value must be in the range from $3\text{ k}\Omega$ to $100\text{ k}\Omega$, and the value of the capacitor must be greater than 20 pF , And the resulting frequency depends on the value of the voltage used to operate the PIC, the value of the resistance and the capacitor, the tolerance ratio for each, the temperature and the amount of **parasitic capacitance** produced between the terminals of the PIC and the capacitor, so the frequency varies from one unit to another, and the following equation is an equation Approximate frequency value calculation:

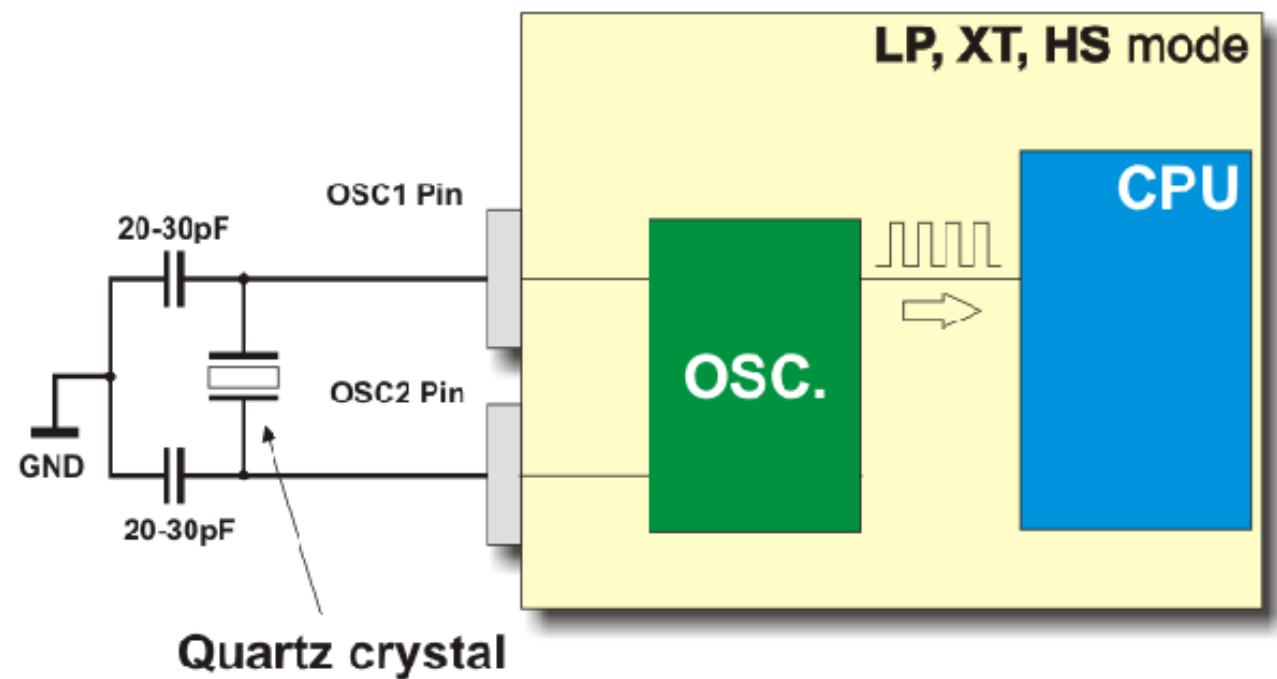
D. RC (External Resistor / Capacitor)

$$F = 1 / (4.2 R C)$$

$$F = \text{HZ}$$

$$R = \Omega$$

$$C = F$$



** Notes

- ✓ It is not necessary that all these patterns exist in the same PIC, it is possible that some patterns exist without others, so you must refer to the datasheet of the pic to confirm the patterns used and their frequency range.
- ✓ The **HS**, **XT**, and **LP** modes use crystal for accurate timing, while the rest are used for applications where timing is not important.
- ✓ The higher the frequency, the faster the PIC works, and in most cases, every operation that takes place inside the pic needs 4 clocks of the oscillator. If the oscillator frequency is **4MHz**, this leads to performing **one million operations per second**, and if the oscillator frequency is **20MHz**, this leads to performing **five million operations per second**.

** Notes (Cont.)

- ✓ The higher the frequency, the higher the electric current consumption, and the lowest electric current consumption pattern is **Low Power (LP)** and it is suitable for applications that need **low cost** with **accurate timing** and uses the **battery** as a power source.

∴ Another Features:

a) Power up Timer (PWRT)

This circuit works when starting the PIC, as it delays the operation of the PIC for a period of **72 msec** until the voltage is stabilized so that the PIC works properly, after which the program is executed, and this feature can be activated through the **configuration bits**.

b) Power on Reset (POR)

This feature puts the PIC into a reset state until all the internal circuits of the microcontroller are started and predefined values are set in their registers.

c) Brown-out Reset (BOR)

This circuit puts the PIC into a reset state when the voltage feeding the pic drops too low to work properly, until the voltage returns to its normal position and the circuit works. This feature can be activated through the configuration bits, and attention should be paid to Activating this circuit will draw more current.

∴ Another Features: (Cont.)

d) Code Protection

This feature is used for the purpose of maintaining the confidentiality of the program code, as some programmers have the ability to read the program from the PIC and store it in a **hex** file, and this feature is used to prevent the process of reading the code from the PIC.

∴ Another Features: (Cont.)

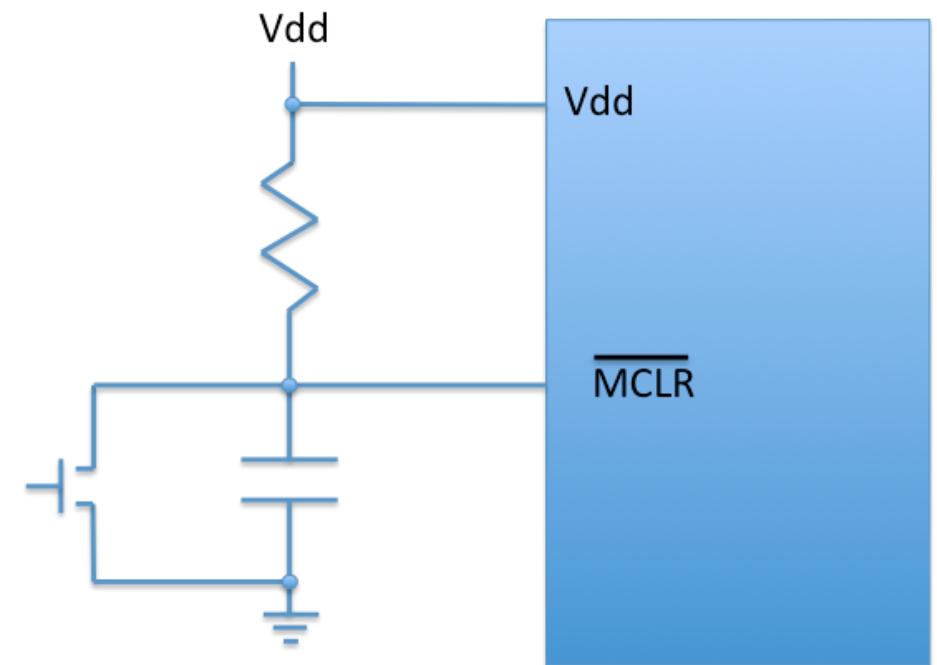
e) Master Clear (MCLR) or Reset pin

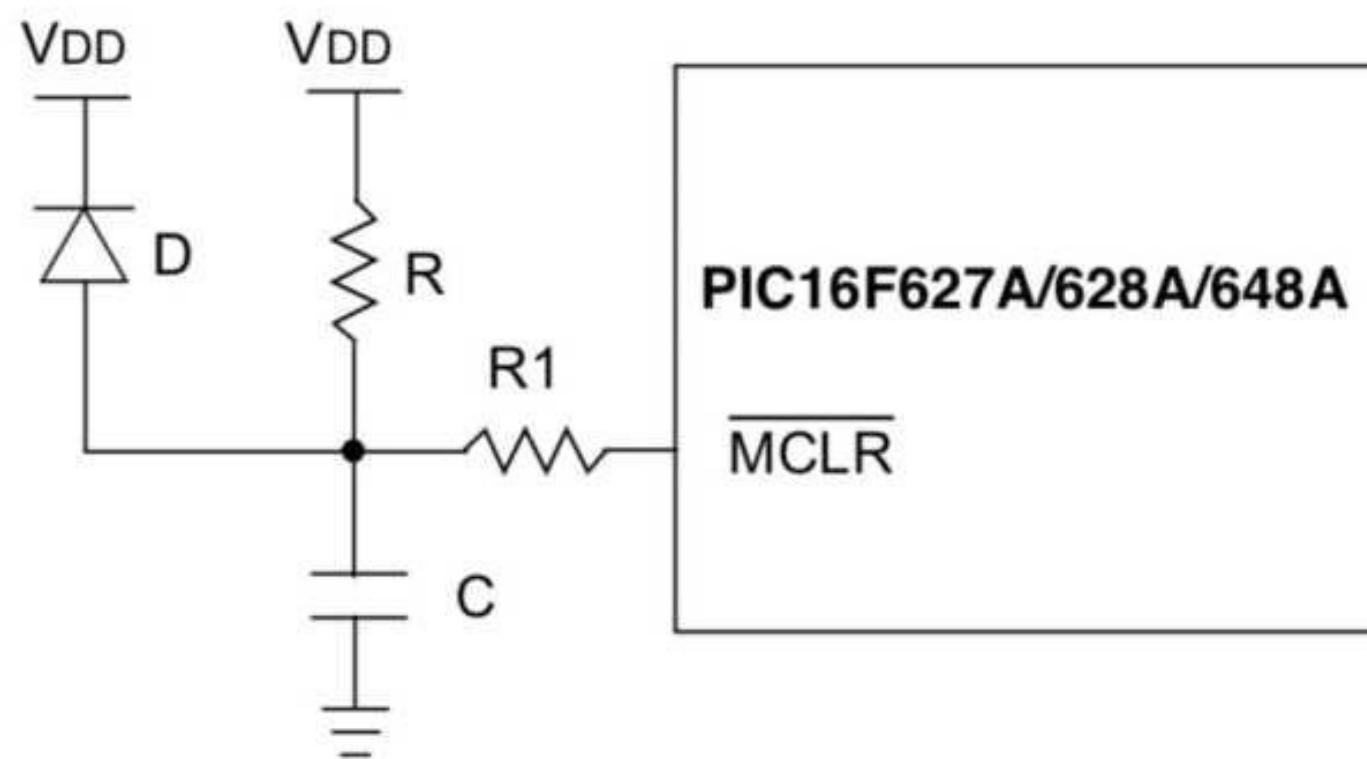
This pin is used to reset the pic through a **low voltage** connection where the microcontroller is restarted with the default values of the registers when a hang occurs or the microcontroller works unexpectedly, and in general if a reset of the pic occurs, the contents of the RAM do not Change or lose, and the following figure shows the connection of the MCLR pin.

∴ Another Features: (Cont.)

e) Master Clear (MCLR) or Reset pin

A push button is connected to the MCLR; when is not pressed, MCLR is not activated. Before starting, the LED is off. When it starts, the LED will turn on and it will remain this way.





∴ Another Features: (Cont.)

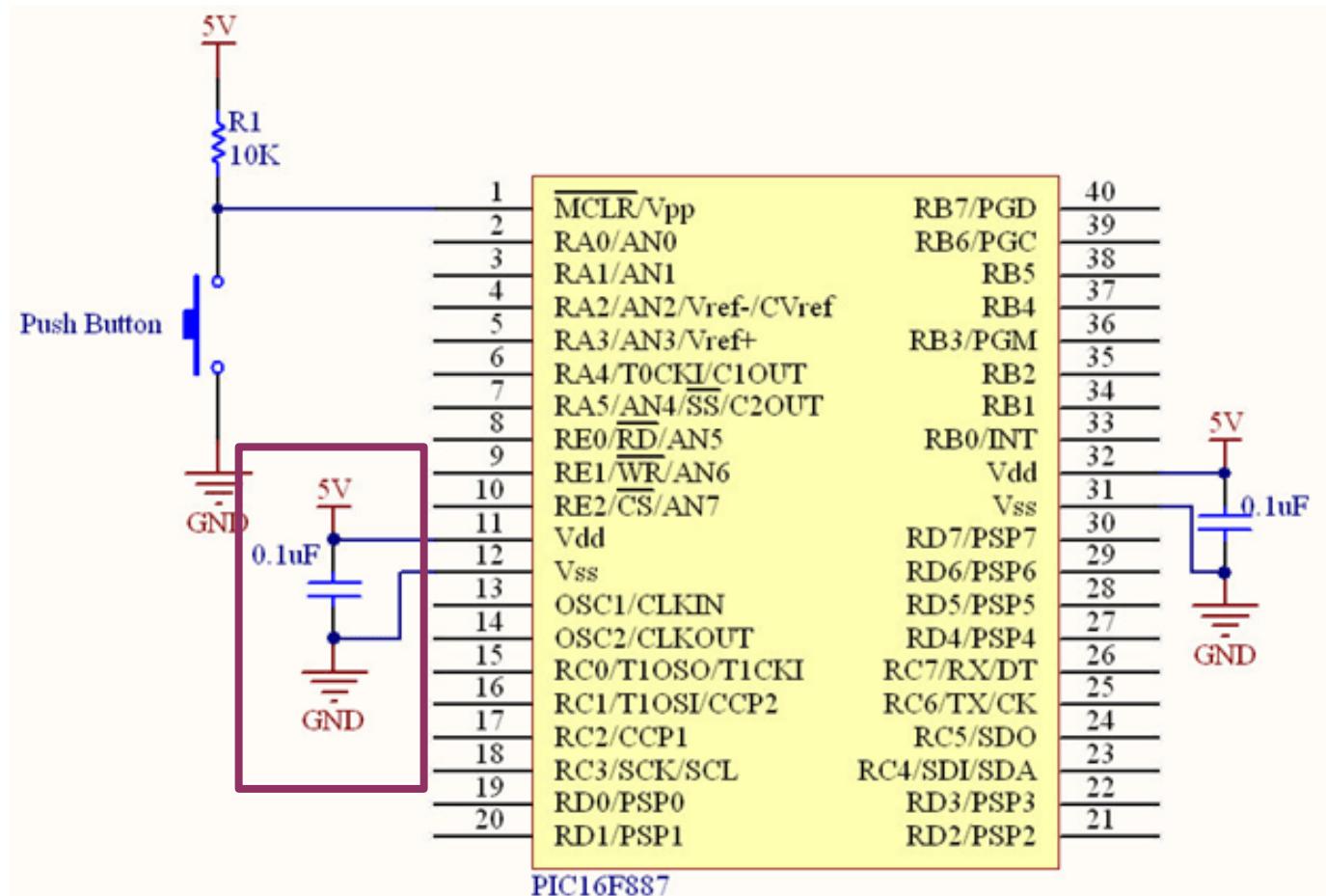
f) MCU Power

Each microcontroller has at least one **VSS** pin for **grounding**. And the **VDD** pin to connect to the **positive** voltage, and if there is more than one pin for the same voltage, they are connected internally, but each pin, **VDD**, **VSS** must be connected externally to ensure that the microcontroller works in a proper way.

Also, a decoupling capacitor must be connected to stabilize the current and overcome noise in order for the microcontroller to work properly. One end of that capacitor is connected to the VDD terminal of the microcontroller and the other end is connected to ground as shown in the following figure, and the capacitor value ranges from 0.01uF to 0.1uF It is preferable to use the **tantalum**, **ceramic disk** or **polyester** type. **Electrolytic** capacitors should not be used. In an environment with a lot of noise sources, the two values can be used in parallel to pass different frequencies.

∴ Another Features: (Cont.)

f) MCU Power

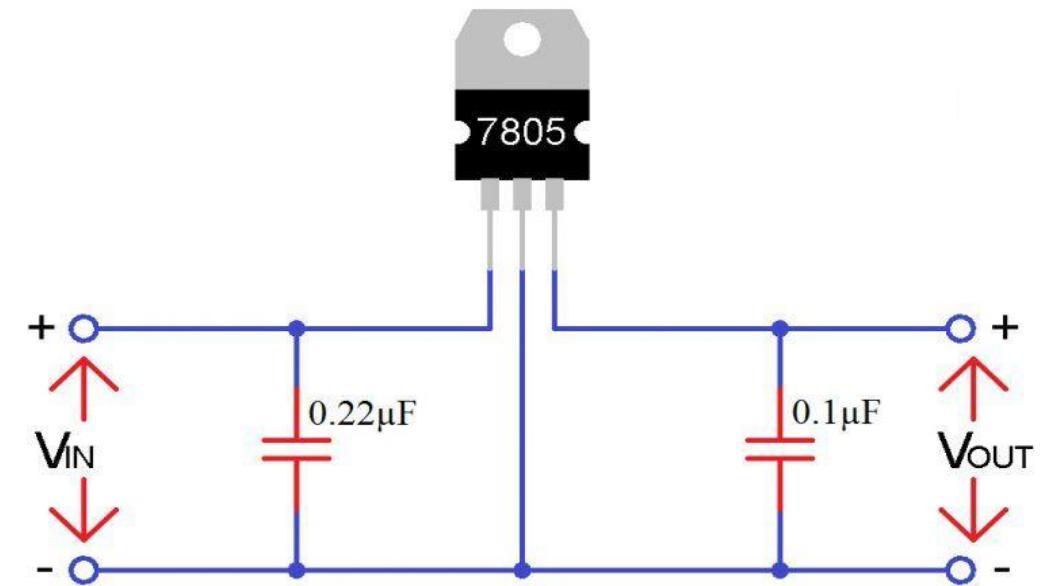
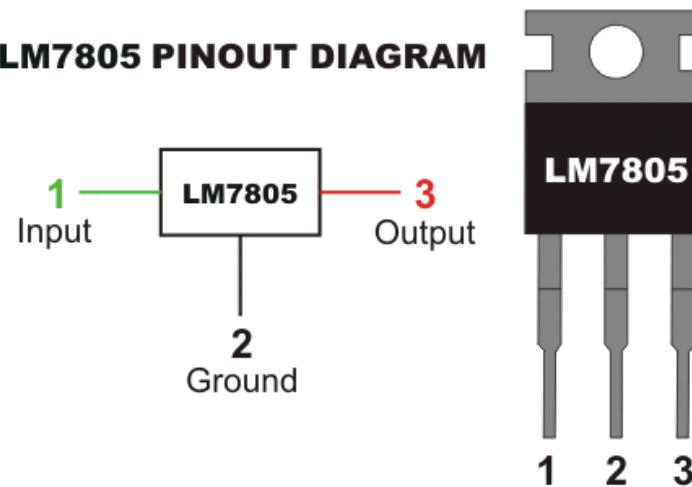


∴ Another Features: (Cont.)

f) Voltage Regulator

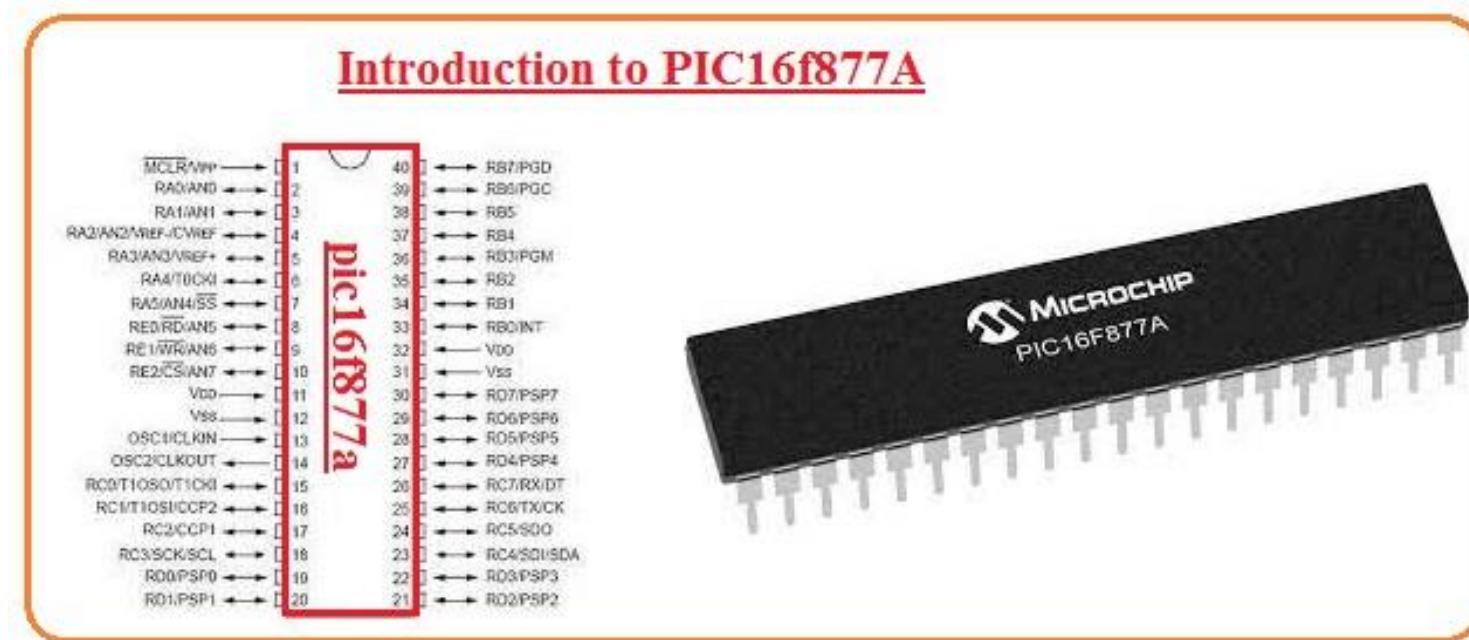
It is used to reduce the voltage and install it on a specific voltage. We use the **7805** regulator to obtain a constant voltage equal to **5** volts, and the input voltage of the regulator must be **7.5** volts or slightly higher.

LM7805 PINOUT DIAGRAM



∷ Another Features: (Cont.)

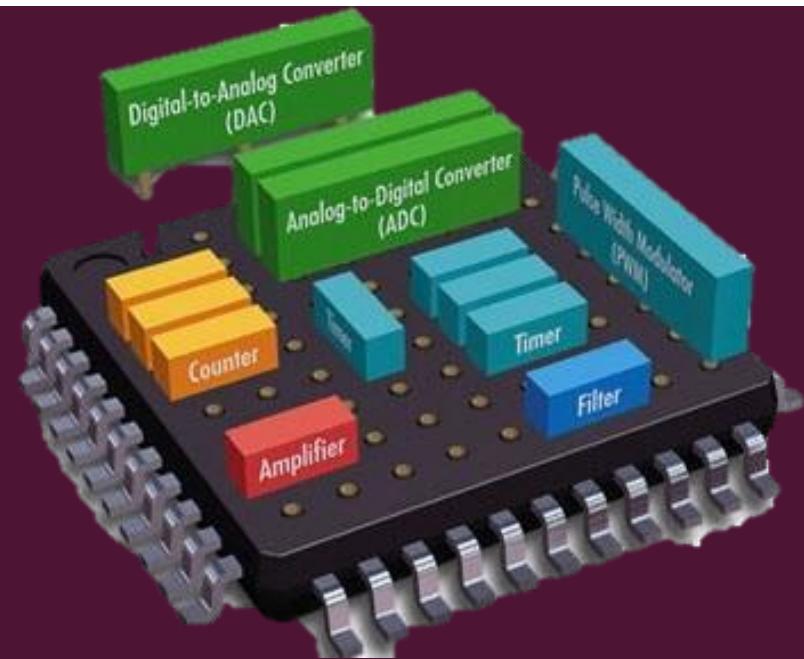
- In this book we will use the 16F874A/877A PIC and



PIC MICROCONTROLLER

05 MIKROC

Eng. Elaf A.Saeed



❖ Variables

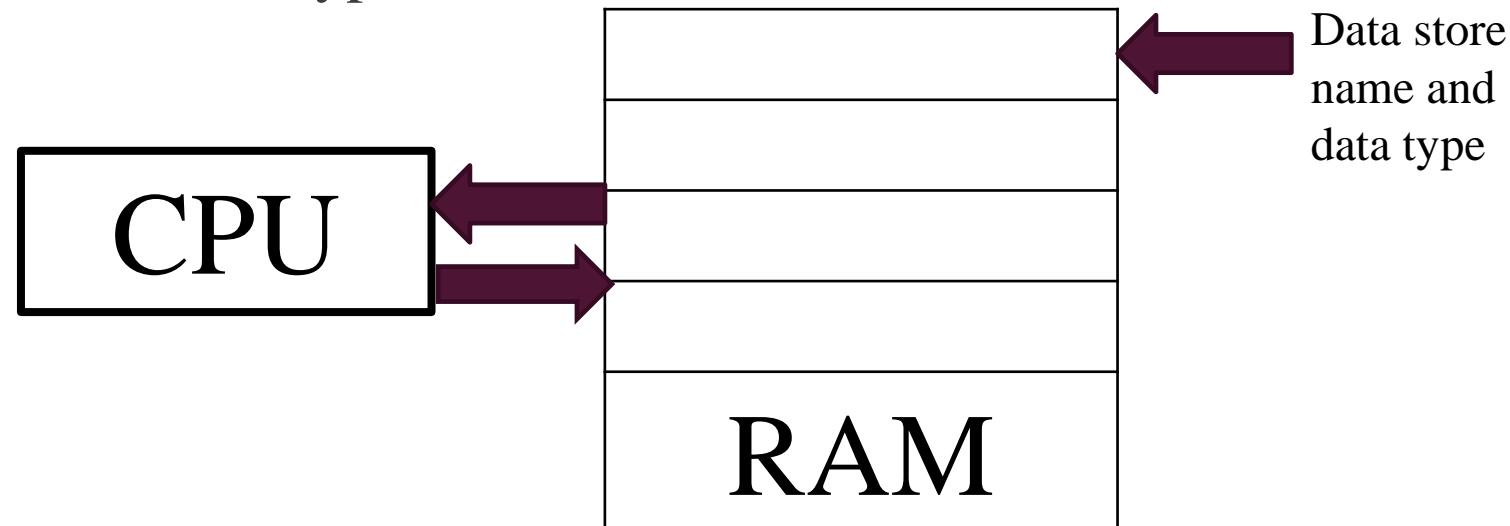
In order to execute a program that has been written, it uses **RAM** to put the data and the results of execution in it, and in order to deal with the places that contain the data in the **memory**, it is **named**, and also determines the type of statement that will be stored in this place.

$$1 + 2 = 3$$

$$a = 1$$

$$b = 2$$

$$c = a + b = 3$$



❖ Variables (Cont.)

Conditions to be met when naming:

- The name consists of the letters A → Z, uppercase or lowercase, as well as the numbers 0 → 9, and only the "_" underscore.
- The name must begin with a letter or “_” and not begin with a number.
- The name must not contain spaces.
- The name should not be a **mikroC** keyword or command.

❖ Variables (Cont.)

The following examples show proper nomenclature:

Temperature_degree

MyName

SUM_4

_ID02

The following examples illustrate the misnomer:

Your Address

4you

book.color

Space

starts with a number

It has a special

character which is the period

✿ Types

There are some basic types that are used with determining the type of variables, which are:

- **char** - Used to store an Ascii code, as a literal value, or to store a number.
- **int** - Used to store integer numbers.
- **float** - Used to store decimal numbers.

✿ Types (Cont.)

Words are also used to indicate whether the number is positive, positive and negative.

- **signed** - To indicate that the number can be negative.
- **Unsigned** - To indicate that the number should not be negative, i.e. only positive.

Words are also used to specify the largest numeric value that can be stored, which is:

- **short**
- **long**

✿ Types (Cont.)

The following table shows how to write these types

Type	Size (bits)	Size (bytes)	Range
char	8	1	-128 to 127
unsigned char	8	1	0 to 255
int	16	2	- 2^{15} to $2^{15}-1$
unsigned int	16	2	0 to $2^{16}-1$
short int	8	1	-128 to 127
unsigned short int	8	1	0 to 255
long int	32	4	- 2^{31} to $2^{31}-1$
unsigned long int	32	4	0 to $2^{32}-1$
float	32	4	3.4E-38 to 3.4E+38
double	64	8	1.7E-308 to 1.7E+308
long double	80	10	3.4E-4932 to 1.1E+4932

✿ Types (Cont.)

○ Examples

```
unsigned short int a = 1;
```

```
unsigned short int a ;
```

```
a = 1;
```

```
unsigned short int a;
```

```
unsigned short int b;
```

```
unsigned short int a , b;
```

```
a = 1;
```

```
b = 2;
```

```
unsigned short int a = 1,b=2;
```

```
-----
```

```
Unsigned char count, initial;
```

```
count = 120;
```

```
initial = 'T';
```

```
signed char first, count;
```

```
first = -180;
```

```
count = 25;
```

```
bit on, off;
```

```
on = 1;
```

```
off = 0;
```

```
-----
```

```
sbit LED at PORTB.B1;
```

❖ Arrays

Arrays are used to define data of the same type and type and are accessed using an index directory. The array containing variable data is declared as follows:

Type array_name [constant-expression]

*/*Declare an array which holds number of days in each month: */*

```
int days [7] = {31, 28, 31, 30, 66, 50, 99};
```

/ This declaration is identical to the previous one */*

```
int days [] = {31, 28, 31, 30, 66, 50, 99} ;
```

```
unsigned int Total[15];
```

```
unsigned int[1] = 25;
```

```
Temp = Total[2];
```

❖ Arithmetic Operators

Arithmetic operators are used in arithmetic computations.

/ for example */*

`7 / 4 /* equals 1 */`

`7 / 4 /* equals 5 */`

/ but: */*

`7. *3. / 4.;` */* equals 5.25 because*

*we are working with float */*

`9 % 3 /*equals 0 */`

`7 % 3 /*equals -1*/`

`-7 % 3 /*equals -1*/`

Operator	Description	Example
+	Adds two operands.	$A + B = 30$
-	Subtracts second operand from the first.	$A - B = -10$
*	Multiplies both operands.	$A * B = 200$
/	Divides numerator by de-numerator.	$B / A = 2$
%	Modulus Operator and remainder of after an integer division.	$B \% A = 0$
++	Increment operator increases the integer value by one.	$A++ = 11$
--	Decrement operator decreases the integer value by one.	$A-- = 9$

❖ Arithmetic Operators

Arithmetic operators are used in arithmetic computations.

/ Post-increment operator */*

```
j = 4;
```

```
k = j++; // k = 4, j=5;
```

/ pre-increment operator */*

```
j = 4;
```

```
k = ++j; // k = 5, j = 5
```

/ post-decrement operator */*

```
j = 12;
```

```
k = j--; //k = 12, j = 11
```

Operator	Description	Example
+	Adds two operands.	A + B = 30
-	Subtracts second operand from the first.	A - B = -10
*	Multiplies both operands.	A * B = 200
/	Divides numerator by de-numerator.	B / A = 2
%	Modulus Operator and remainder of after an integer division.	B % A = 0
++	Increment operator increases the integer value by one.	A++ = 11
--	Decrement operator decreases the integer value by one.	A-- = 9

❖ Arithmetic Operators

Arithmetic operators are used in arithmetic calculations.

/ pre-decrement operator */*

j = 12;

k = -- j ; //k = 11, j = 11

Operator	Description	Example
+	Adds two operands.	A + B = 30
-	Subtracts second operand from the first.	A - B = -10
*	Multiplies both operands.	A * B = 200
/	Divides numerator by de-numerator.	B / A = 2
%	Modulus Operator and remainder of after an integer division.	B % A = 0
++	Increment operator increases the integer value by one.	A++ = 11
--	Decrement operator decreases the integer value by one.	A-- = 9

❖ Constants

Constants represent fixed values (numeric or character) in programs that cannot be changed.

Constant are stored in the flash program memory of the pic microcontroller, thus not wasting variable and limited RAM memory.

```
const MAX = 100;
```

```
const Min = 0b11110000;
```

```
const TOTAL = 0xFF;
```

```
const CNT = 017;
```

```
const TEMP = 37.50;
```

```
const First_Alpha = 'A';
```

❖ Constants (Cont.)

In case of array of char, you can use a shorter string literal notation. For example:

/ The two declaration are identical: */*

```
const char msg1[] = {'T', 'e', 's', 't', '\0'};
```

```
const char msg2[] = "Test";
```

Relational Operators

Use Relational Operators to test equality or inequality of expression. If an expression evaluates to be true, it returns 1; otherwise it returns 0.

x = 10

x > 8 //returns 1

x == 10 //returns 1

x < 100 //returns 1

x > 20 //returns 0

x != 10 //returns 0

x >= 10 //returns 1

x <= 10 //returns 1

Operator	Description
==	equal to
!=	not equal to
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to

❖ Logical Operators

Logical operators are used in logical and arithmetic comparisons, and they return TRUE (i.e.. Logical 1) if the expression evaluates to nonzero and FALSE (i.e.. Logical 0) if the expression evaluates to zero.

/ Logical AND */*

x = 7;

x > 0 && x < 10

//returns 1

x > 0 || x < 10

//returns 1

x >= 0 && x <= 10

//returns 1

x >= 0 && x < 5

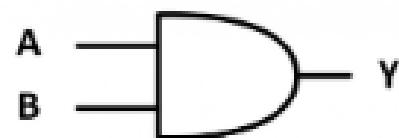
//returns 0

Operator	Name	Example
<code>&&</code>	AND	(A && B) is False
<code> </code>	OR	(A B) is True
<code>!</code>	NOT	!(A && B) is True

Logic Gates

AND Gate

Inputs		Output
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1



$$Y = A \cdot B$$

A

1	1	0	0	1	0	0	1
---	---	---	---	---	---	---	---

B

0	1	1	0	1	0	0	0
---	---	---	---	---	---	---	---

↓

Y

0	1	1	0	1	0	0	0
---	---	---	---	---	---	---	---

Logic Gates

OR Gate



A	B	C
0	0	0
0	1	1
1	0	1
1	1	1

A

1	1	0	0	1	0	0	1
---	---	---	---	---	---	---	---

↓

B

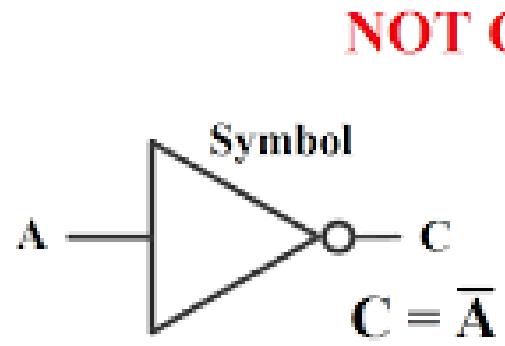
0	1	1	0	1	0	0	0
---	---	---	---	---	---	---	---

C

1	1	1	0	1	0	0	1
---	---	---	---	---	---	---	---

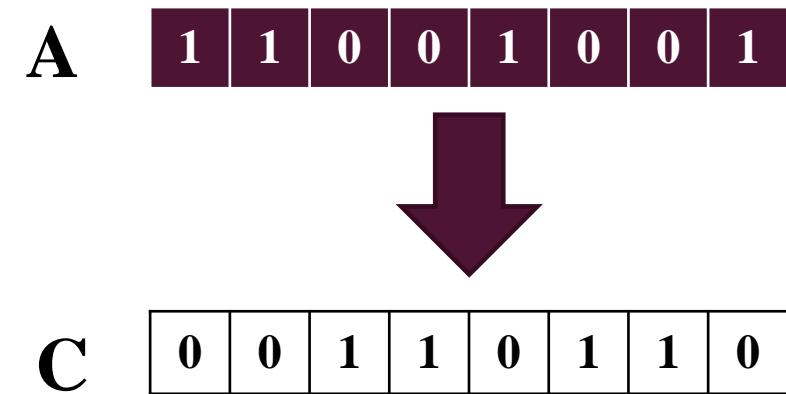
Logic Gates

NOT Gate



Truth Table

INPUT	OUTPUT
A	NOT A
0	1
1	0



❖ Bitwise Operators

Bitwise Operators are used to modify the bits of a variable.

Operator	Description
&	Bitwise AND
	Bitwise OR
^	Bitwise XOR
~	Bitwise NOT
<<	Bitwise left shift
>>	Bitwise right shift

❖ Bitwise Operators

Bitwise Operators are used to modify the bits of a variable.

0x14 >> 1 returns 0x08 (shift 0x14 right by 1 digit)

0x14: 0001 0100

>>1: 0000 1010

0xA: 0000 1010

0x14 >> 2 returns 0x05 (shift 0x14 right by 2 digits)

0x14: 0001 0100

>> 2: 0000 0101

0x05: 0000 0101

❖ Bitwise Operators

Bitwise Operators are used to modify the bits of a variable.

`0x235A >> 1` returns `0x46B4` (shift left `0x235A` left by 1 digit)

`0x235A:` 0010 0011 0101 1010

`<<1:` 0100 0110 1011 0100

`0x46B4:` 0100 0110 1011 0100

`0x1A << 3` returns `0xD0` (shift left `0x1A` by 3 digits)

`0x1A:` 0001 1010

`<<3:` 1101 0000

`0xD0` 1101 0000

❖ Compound Assignment Operators

j = j + k;

j += k;

p = p * m;

p *= m;

The following component operators can be used in mikroC programs:

+ = - = * = / = % =

& = | = ^ = >> = << =

❖ Conditional Operators

The syntax of a conditional operator is:

Result = Expression 1 ? Expression2 : Expression3

max = (x > y) ? x:y;

✿File Include

#include “file”

The mikroC will look for the file in the following locations, in this particular order:

1. The project folder (folder which contains the project file .mcppi)
2. The mikroC PRO for pic installation folder > “**include**” folder.
3. User’s custom search paths.

#include “C:\temp\first.h”

❖ If Statement

```
if (condition){  
    statements:  
}
```

```
if (x > 5) {  
    statements:  
}
```

```
if (condition){  
    statements:  
} else{  
    statements:  
}
```

```
if (x > 5) {  
    statements:  
} else{  
    statements:  
}
```

```
if (x > 5) {  
    statements:  
} else if (x > 3){  
    statements:  
} else{  
    statements:  
}
```

Switch Statement

```
switch (expression) {  
    case constant-expression_1:  
        break;  
    case constant-expression_1:  
        break;  
    .....  
    case constant-expression_n:  
        statement_n;  
    {default :  
        statement;}  
}
```

```
Switch (Cnt)  
{  
    case 1:  
        A = 1;  
        break;  
    case 10:  
        B = 1;  
        break;  
    case 100:  
        C = 1;  
        break;  
    default:  
        D = 1;
```

❖For Statement

**for (initial expression; condition expression; increment expression)
statements;**

```
for (i=0; i < 10; i++)  
statement;
```

❖ While Statement

while (condition)

statements;

k = 0;

while (k < 10)

statements;

k++;

❖ Do Statement

do

{

statements;

}while (condition)

/ Execution 10 times */*

k = 0;

do

{

statements;

k++;

} while (k < 10)

❖ Goto Statement

goto label identifier;

/ Execution 10 times */*

k = 0;

Loop:

statements;

k ++;

if (k < 10) goto Loop;

❖ Break and Continue Statement

/ Calculate sum of numbers 1,2,3,4,5,6,7,8,9,10 */*

sum = 0;

i = 1;

for (i = 1; i <= 10; i++)

 if (i == 5) continue; // skip number 5

 sum = sum + i;

❖ Break and Continue Statement

```
/* Calculate sum of numbers 1,2,3,4,5,6,7,8,9,10*/
```

```
sum = 0;
```

```
i = 1;
```

```
for (i = 1; i <= 10; i++)
```

```
    if (i == 5) break; // stop loop if i > 5
```

```
    sum = sum + i;
```

❖ White Space

The spacing between the lines does not affect the program.

```
int j;
```

```
char j;
```

```
int i; int j;
```

```
int i;
```

```
char j;
```

```
i = j + 2;
```

```
i = j
```

```
+2;
```

Line Spacing with Backslash(\)

“mikroC PRO \

For PIC compiler “

❖ Case Sensitivity

MikroC variables are not case sensitive

total TOTAL Total ToTal total totaL

Keywords remain case sensitive and they must be written in **lower case**.

❖ Comments

Long comments start with the character “*/**” and terminate with the character “**/*”.

Short comments start with the character “*//*” and do not need a terminate character.

❖ INPUT / OUTPUT PORTS

We have mentioned that the pins of the microcontroller for data output and entry are connected to units called ports, and in the corresponding drawing you will find that it has several units for input and output from them:

PortA consists of 6 pins: RA0, RA1, RA2, RA3, RA4, RA5.

PortB consists of 8 pins: RB0, RB1, RB2, RB3, RB4, RB5, RB6, RB7.

And these units are dealt with through **Special Function Register (SFR)**.

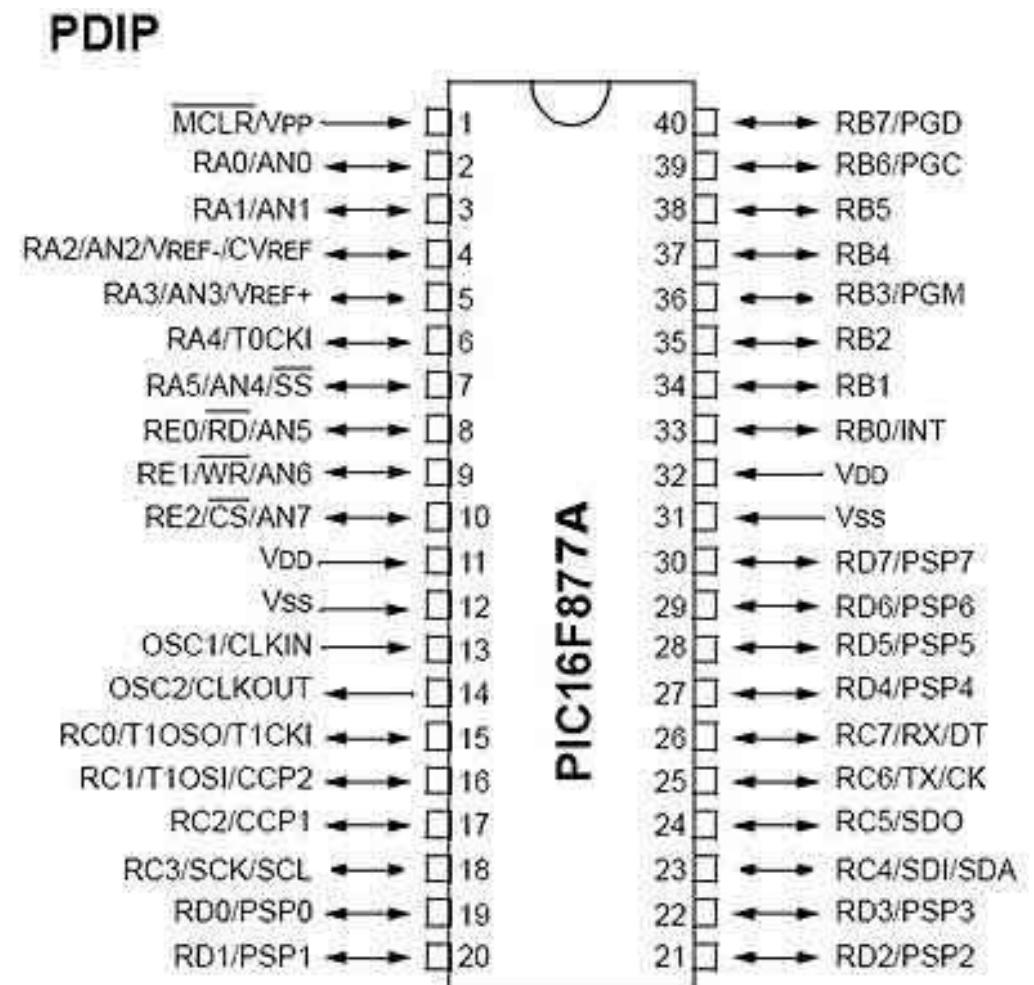
And for each port 2 registers there is a record through which it is dealt.

The **first register** is **TRISx** and specifies which port terminals will be used for data output and which port terminals will be used for input.

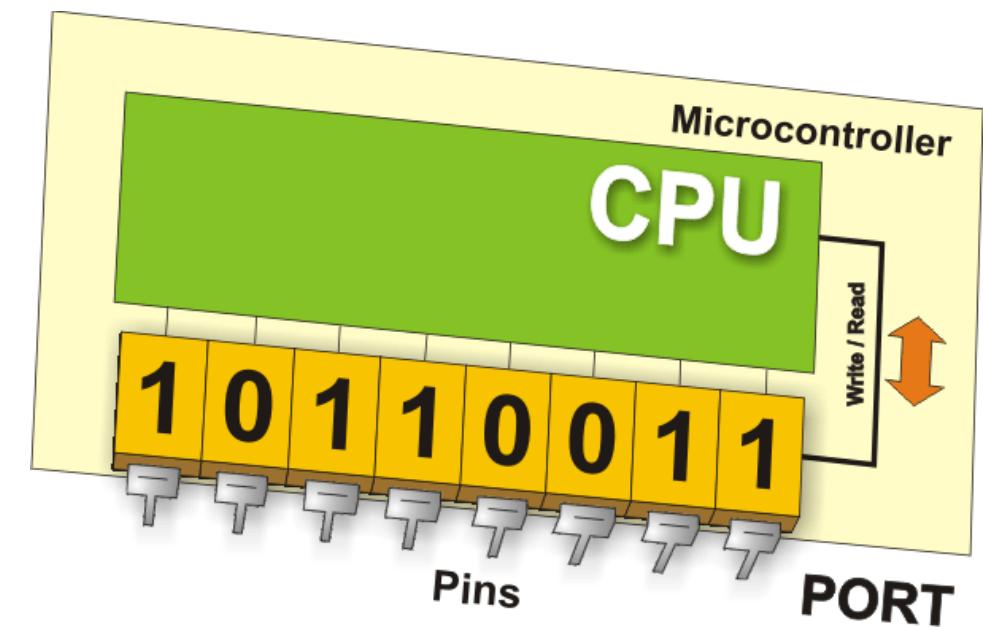
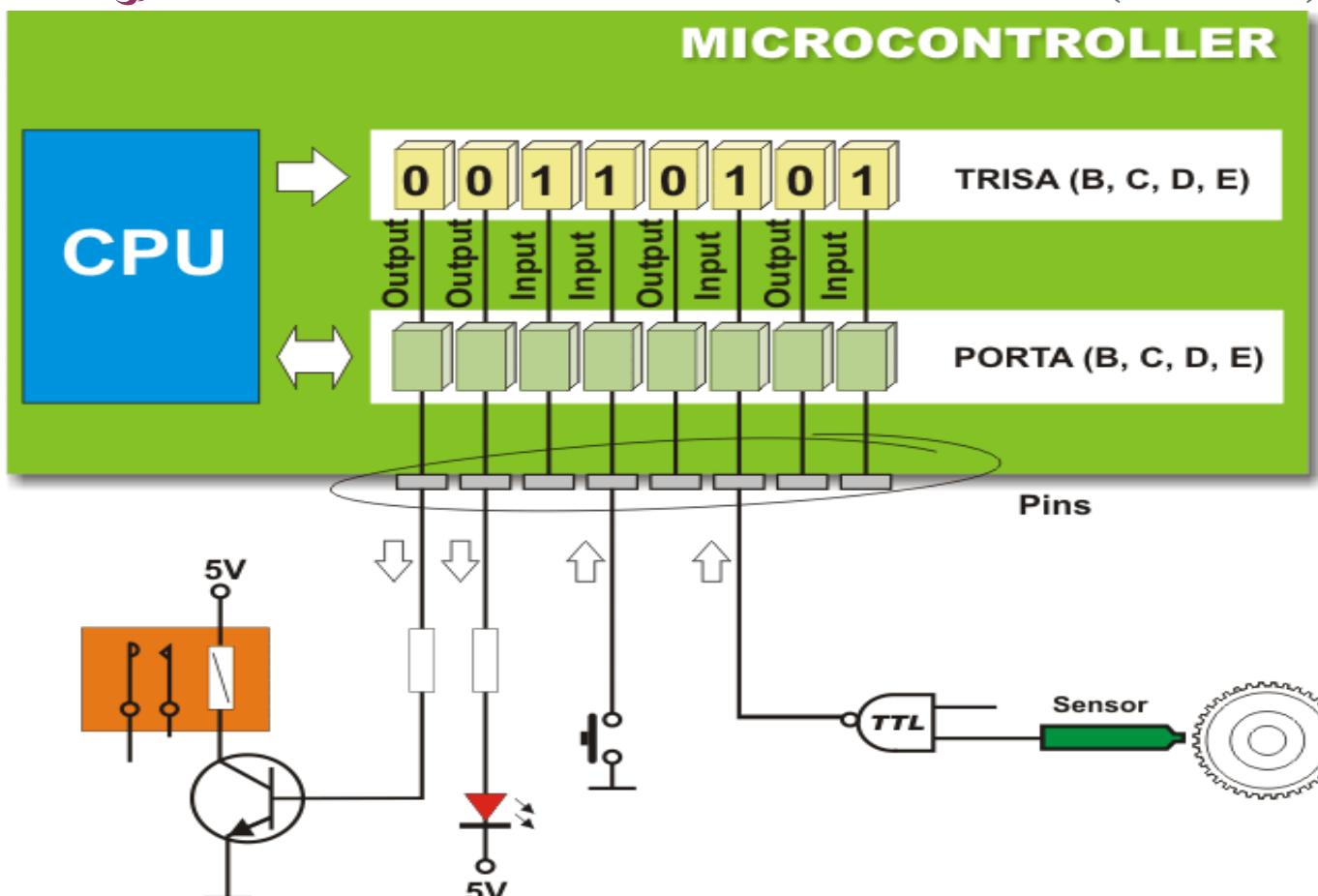
INPUT / OUTPUT PORTS (Cont.)

And the second register is PORT_x, through which data can be placed on the terminals in the case of output, or read from it in the case of input.

These registers are treated as consisting of a set of bits, where each pin is associated with a bit in the register.



INPUT / OUTPUT PORTS (Cont.)



❖ INPUT / OUTPUT PORTS (Cont.)

And if we want to make all sides of PortB to output data only, it can be written in one of the following forms:

TRISB = 0B00000000 //*Binary System*

TRISB = 0x00 //*Hex*

TRISB = 0 //*Decimal System*

Also, each bit can be treated individually as follows:

TRISB.B0 = 0; //RB0

TRISB.B1 = 0; //RB1

TRISB.B2 = 0; //RB2

TRISB.B3 = 0; //RB3

TRISB.B4 = 0; //RB4

TRISB.B5 = 0; //RB5

TRISB.B6 = 0; //RB6

TRISB.B7 = 0; //RB7

❖ INPUT / OUTPUT PORTS (Cont.)

Some of the sides can be made for input and others for output as follows:

`TRISB = 0b10011011`

And in the case of data output, we use the **PORTx** register in the same way we use the **TRISx** register, putting the binary number **0** in the bit connected to one end means that the voltage on this pin is **low**, and putting the binary number **1** means that the voltage on this pin is **high**, The following example puts different efforts on the sides.

`PORTB = 0b10110011`

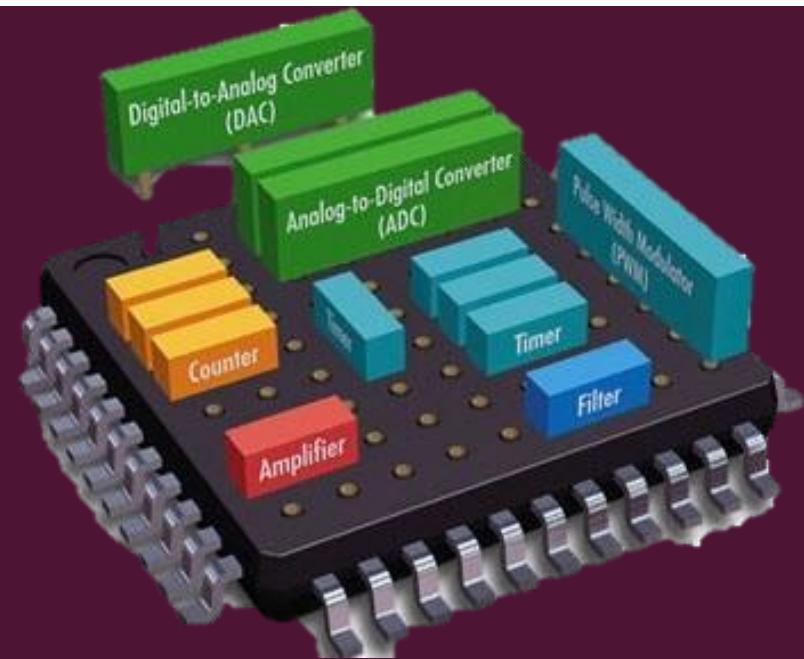
While the following example defines the voltage on a specific pin.

`PORTB.B2 = 1`

PIC MICROCONTROLLER

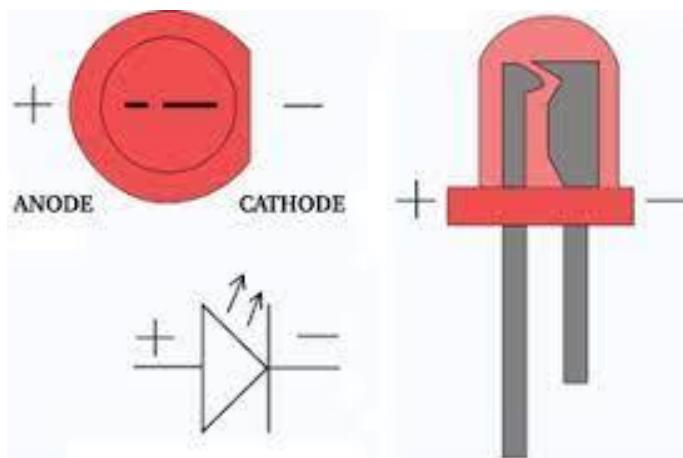
06 EXPERIMENTS

Eng. Elaf A.Saeed



EXPERIMENT 01

LED Flasher



❖ Experiment Object

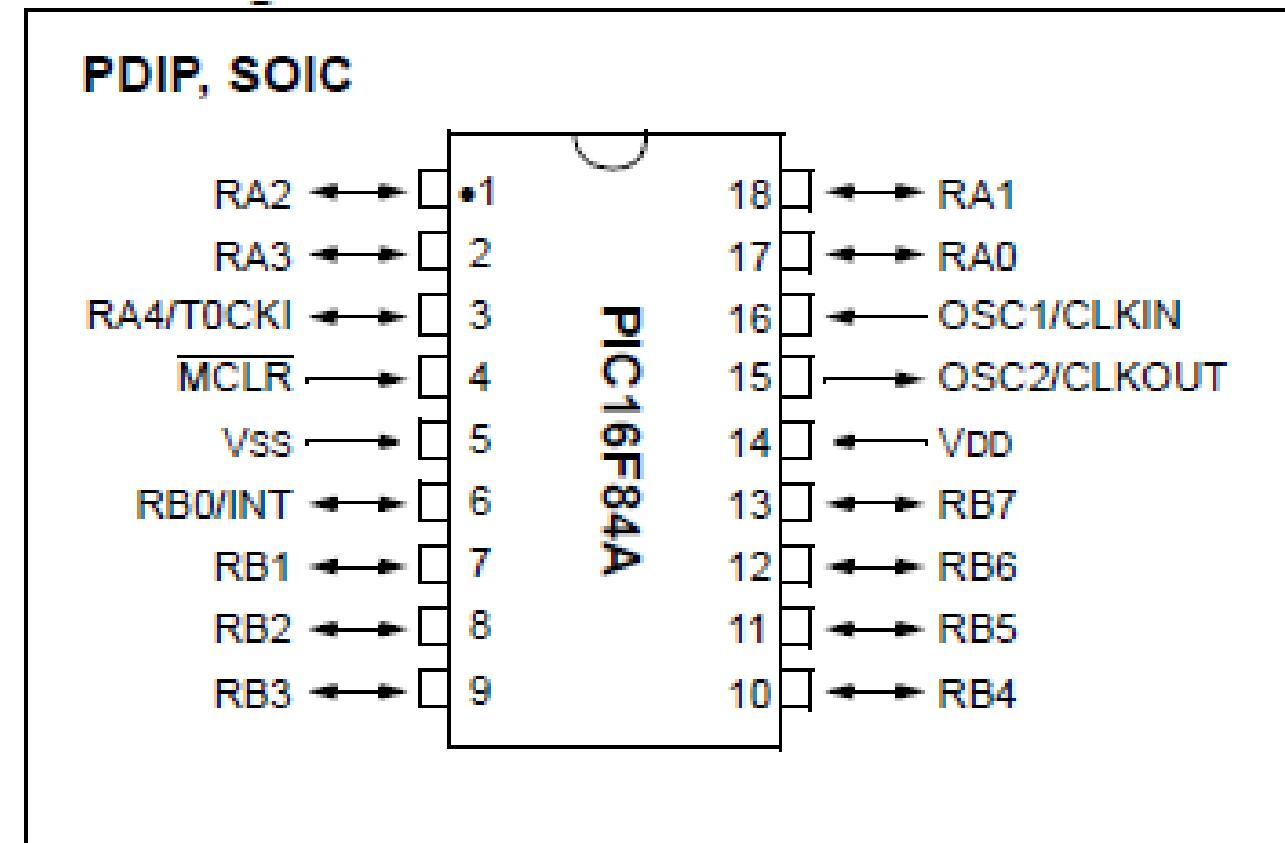
The object of this experiment is to learn the output of the microcontroller and use it to turn the led.

❖ The Experiment Components

- Microcontroller Pic 16F84A.
- Led.
- Resistor 150 ohm.

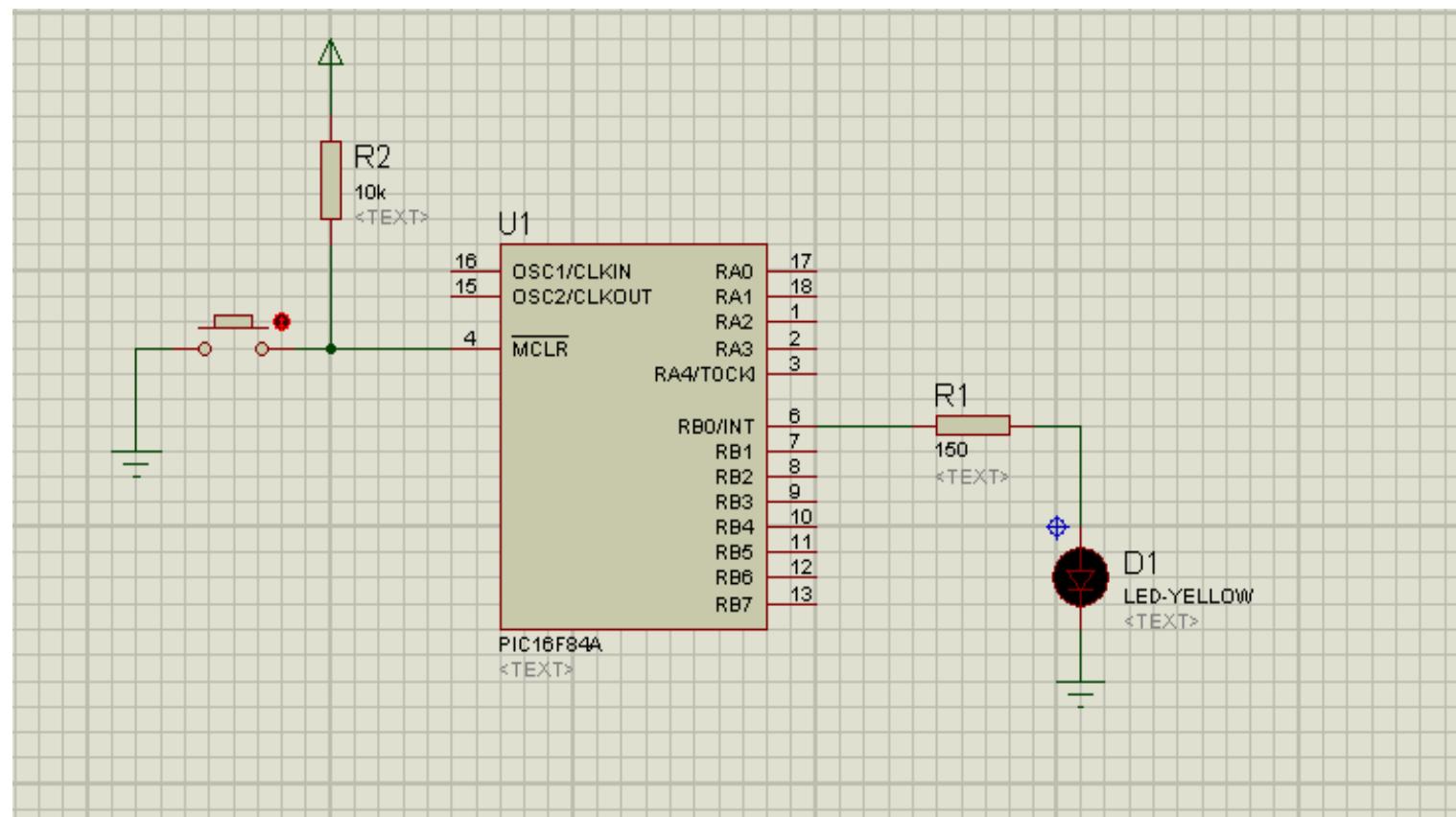
Microcontroller Pic 16F84A

- In this experiment we use the 16F84A pic microcontroller.
- Consists of 18 ports.
- portA → RA0 – RA4.
- portB → RB0 → RB7.
- OSC1 & OSC2.
- VDD & VSS.
- MCLR.



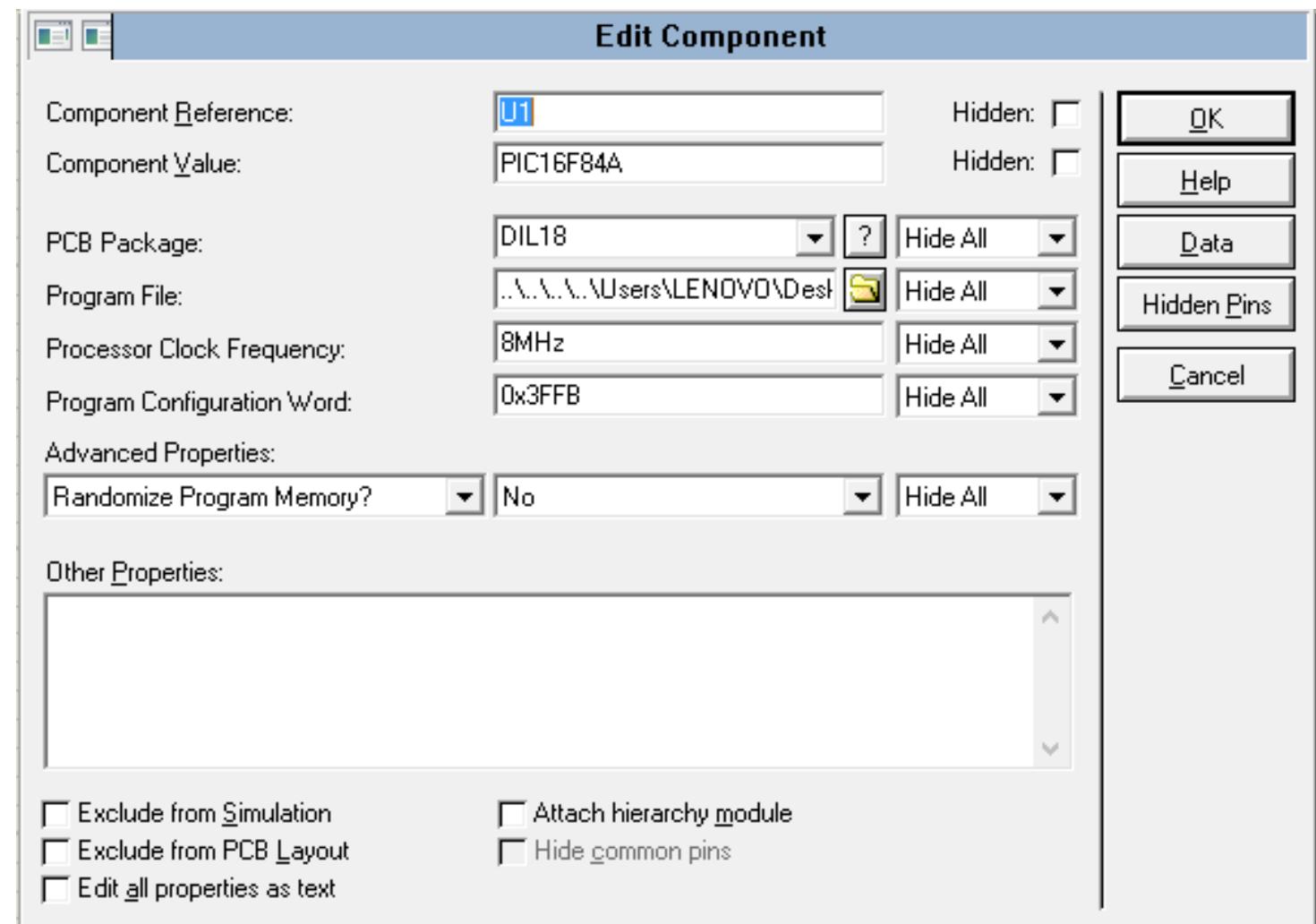
Procedure

Connect the circuit as shown in the figure.



Procedure (Cont.)

Double click on the **pic16F84A**, then the window in the figure show to download the file of extension of (**.hex**) and select the **processor clock frequency** (in this experiment we select the frequency is **8MHZ** the same frequency of the pic in the **mikroC** program.



❖ Procedure (Cont.)

Write the code in MikroC program

```
void main() {  
    trisb.b0=0;    //b0 as output  
    portb.b0=1;    //output 1 b0  
}
```

❖ Procedure (Cont.)

If to make the led turn ON and off every 1000 ms the write code

```
void main() {  
    trisb.b0=0;    //b0 as output  
  
loop:  
    portb.b0=1;    //output 1 b0  
    delay_ms(1000);  
    portb.b0=0;    //output 1 b0  
    delay_ms(1000);  
    goto loop;  
}
```

Discussion

- 1- Write and design the program to turn two LEDs ON and OFF every 2s by using protues and mikriC.

EXPERIMENT 02

Input Port



Experiment Object

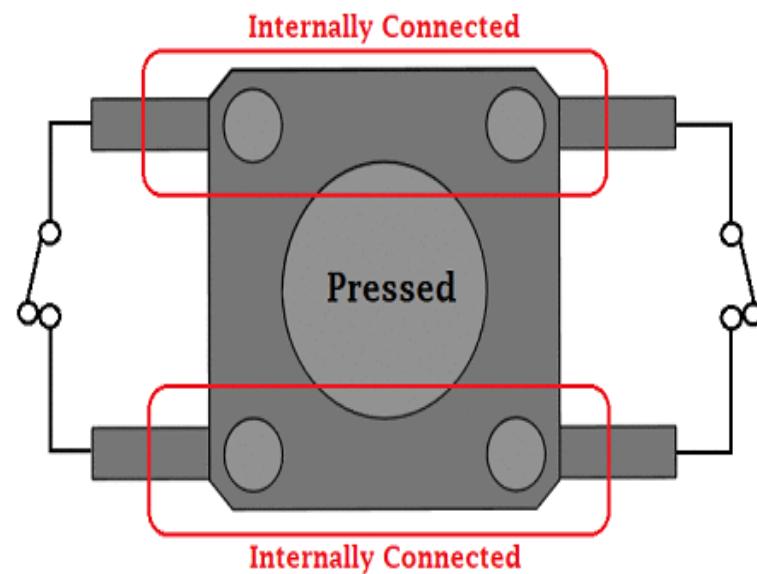
The object of this experiment is to learn the input of microcontroller and use it to turn make the counter.

Theory

Push-Buttons

Push-Buttons are normally-open tactile switches. Push buttons allow us to power the circuit or make any particular connection only when we press the button. Simply, it makes the circuit connected when pressed and breaks when released. A push button is also used for triggering of the SCR by gate terminal. These are the most common buttons which we see in our daily life electronic equipment's. Some of the applications of the Push button are mentioned at the end of the article.

□ Push-Buttons

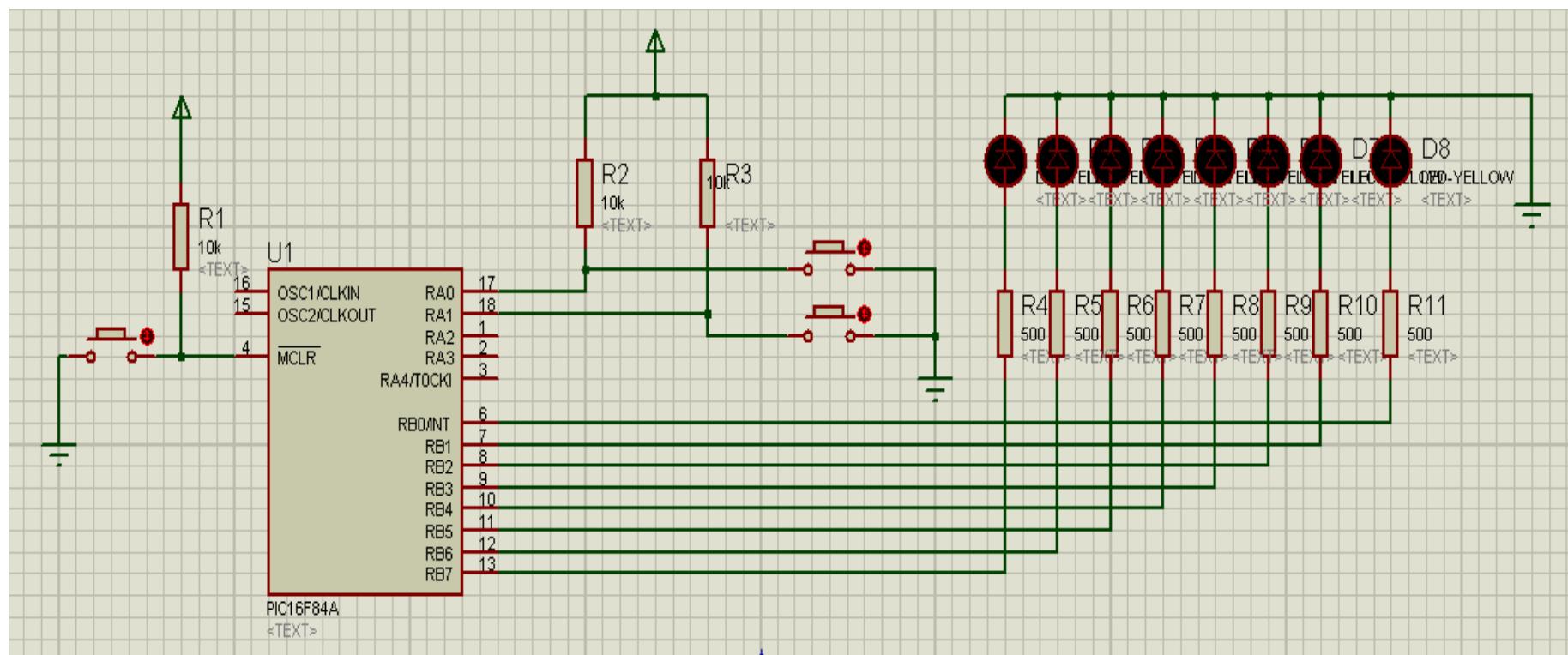


✿ The Experiment Components

- Microcontroller Pic 16F84A.
- Led.
- Resistor 150 ohm.
- Push Button.

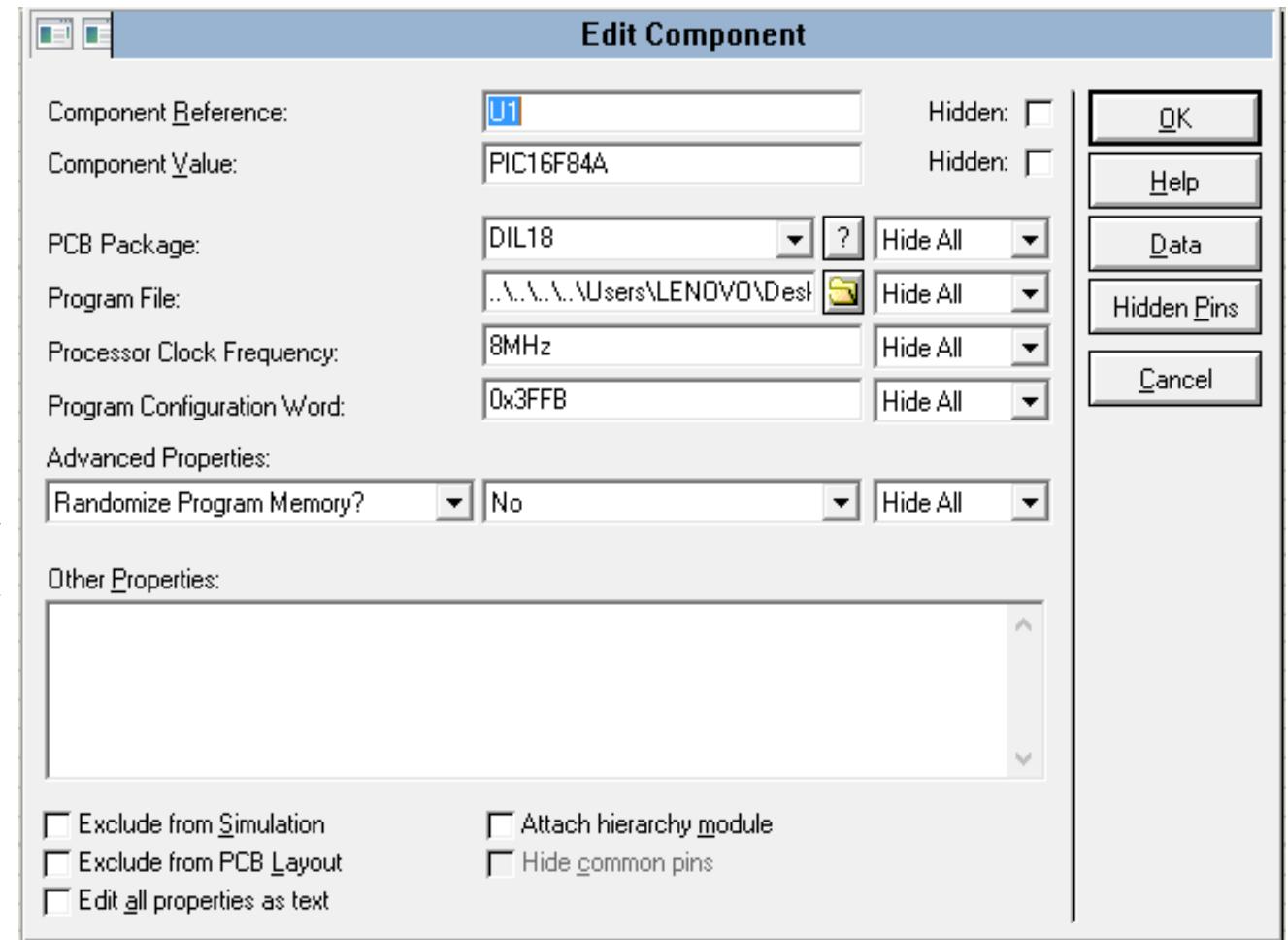
Procedure

Connect the circuit as shown in the figure.



Procedure (Cont.)

Double click on the **pic16F84A**, then the window in the figure show to download the file of extension of (**.hex**) and select the processor clock frequency (in this experiment we select the frequency is **8MHZ** the same frequency of the pic in the mikroC program.



Procedure (Cont.)

Write the code in MikroC program.

```
char counter=0;

void main() {
    trisa.b0=1; //A0 is input
    trisa.b1=1; //A1 is input
    trisb=0; //port b is output
loop:
    if(porta.b0==0) //if A0 was pressed
    {
        counter++; //increment counter
        delay_ms(500); //delay because of debounce issue
    }
    if(porta.b1==0) //if A0 was pressed
    {
        counter--; //increment counter
        delay_ms(500); //delay because of debounce issue
    }
    portb=counter; //send counter value to port b
    goto loop;
}
```

Discussion

- 1- Write and design the same procedure but make it down counter by using protues and mikriC.

EXPERIMENT 03

LCD

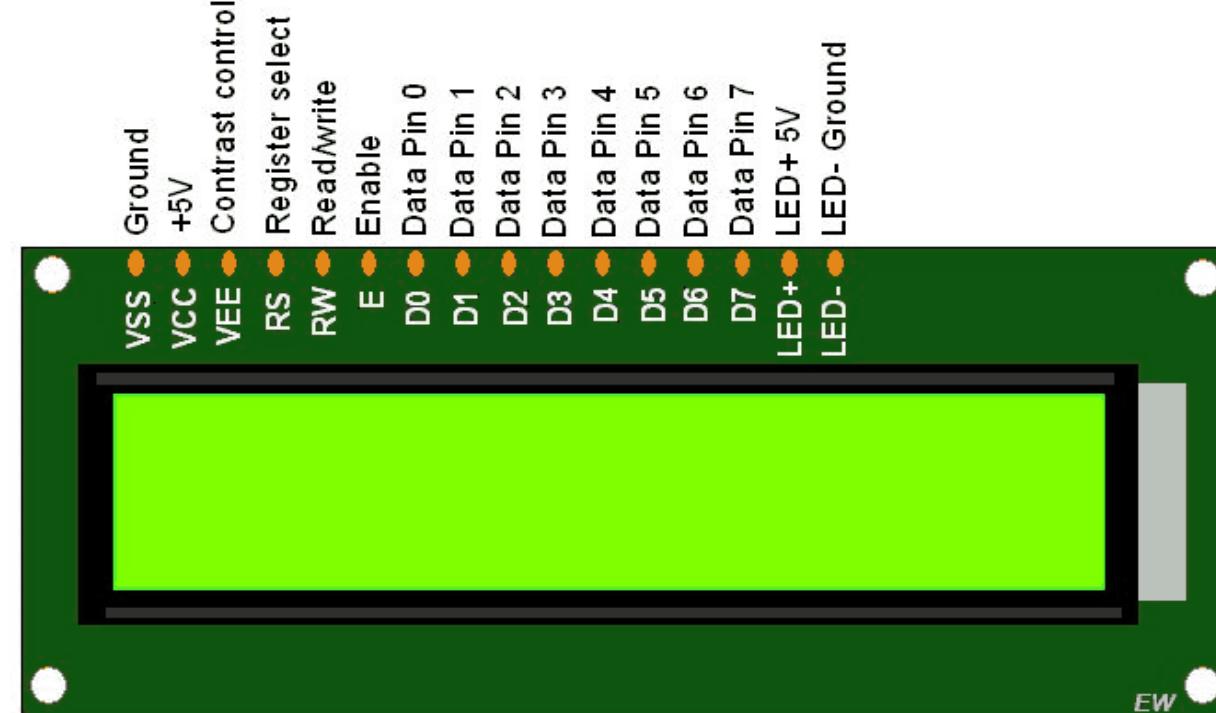


Experiment Object

The object of this experiment is to learn the **LCD** with microcontroller and use it to display.

Theory

LCD 16x2



❖ Theory

□ LCD 16x2

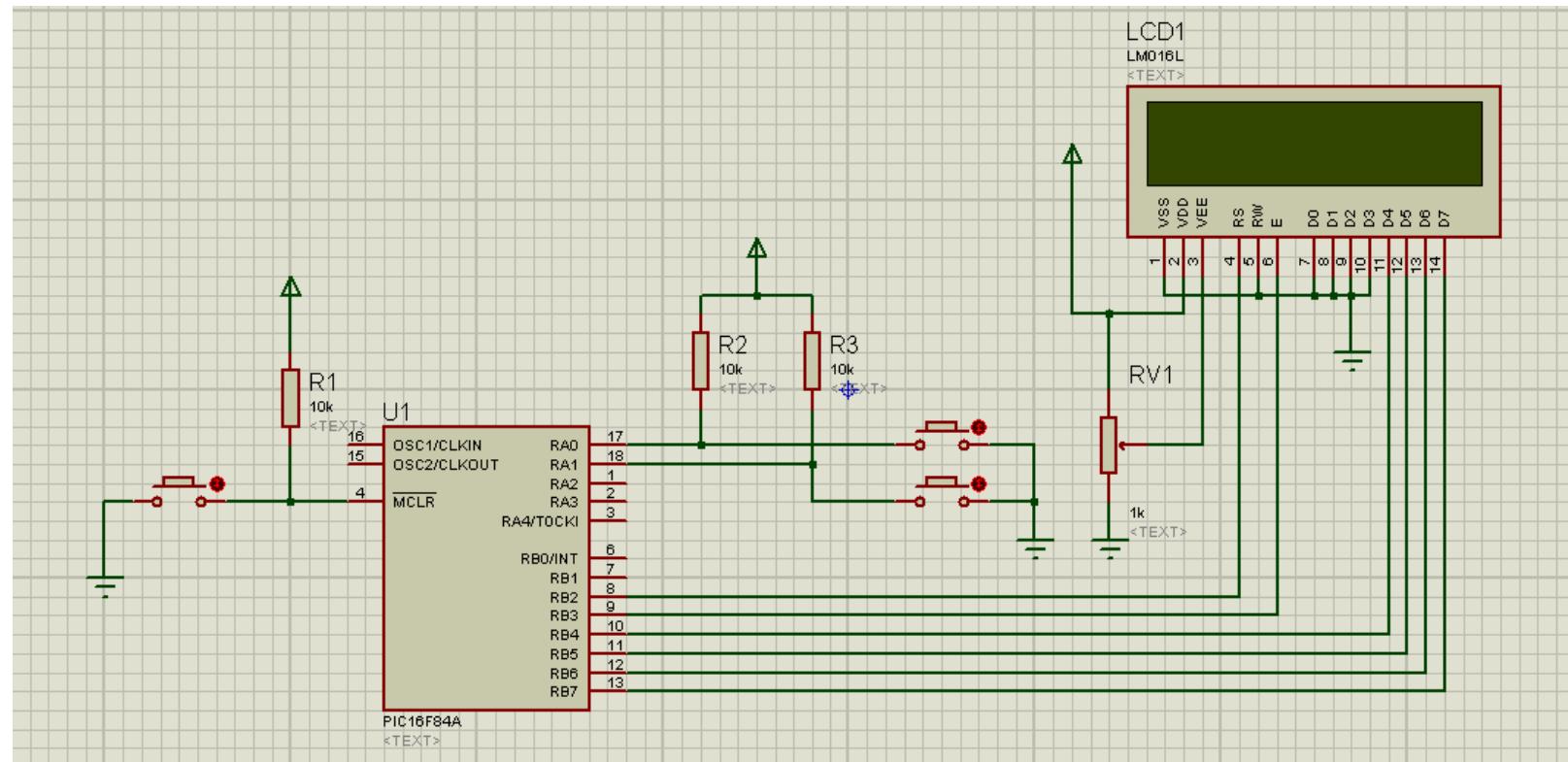
- LCDs (Liquid Crystal Displays) are used in embedded system applications for displaying various parameters and status of the system.
- LCD 16x2 is a 16-pin device that has 2 rows that can accommodate 16 characters each.
- LCD 16x2 can be used in 4-bit mode or 8-bit mode.
- It is also possible to create custom characters.
- It has 8 data lines and 3 control lines that can be used for control purposes.
- For more information about LCD 16x2 and how to use it, refer the topic LCD 16x2 module in the sensors and modules section.

✿ The Experiment Components

- Microcontroller Pic 16F84A.
- LCD.
- Resistor 150 ohm.
- Push Button.

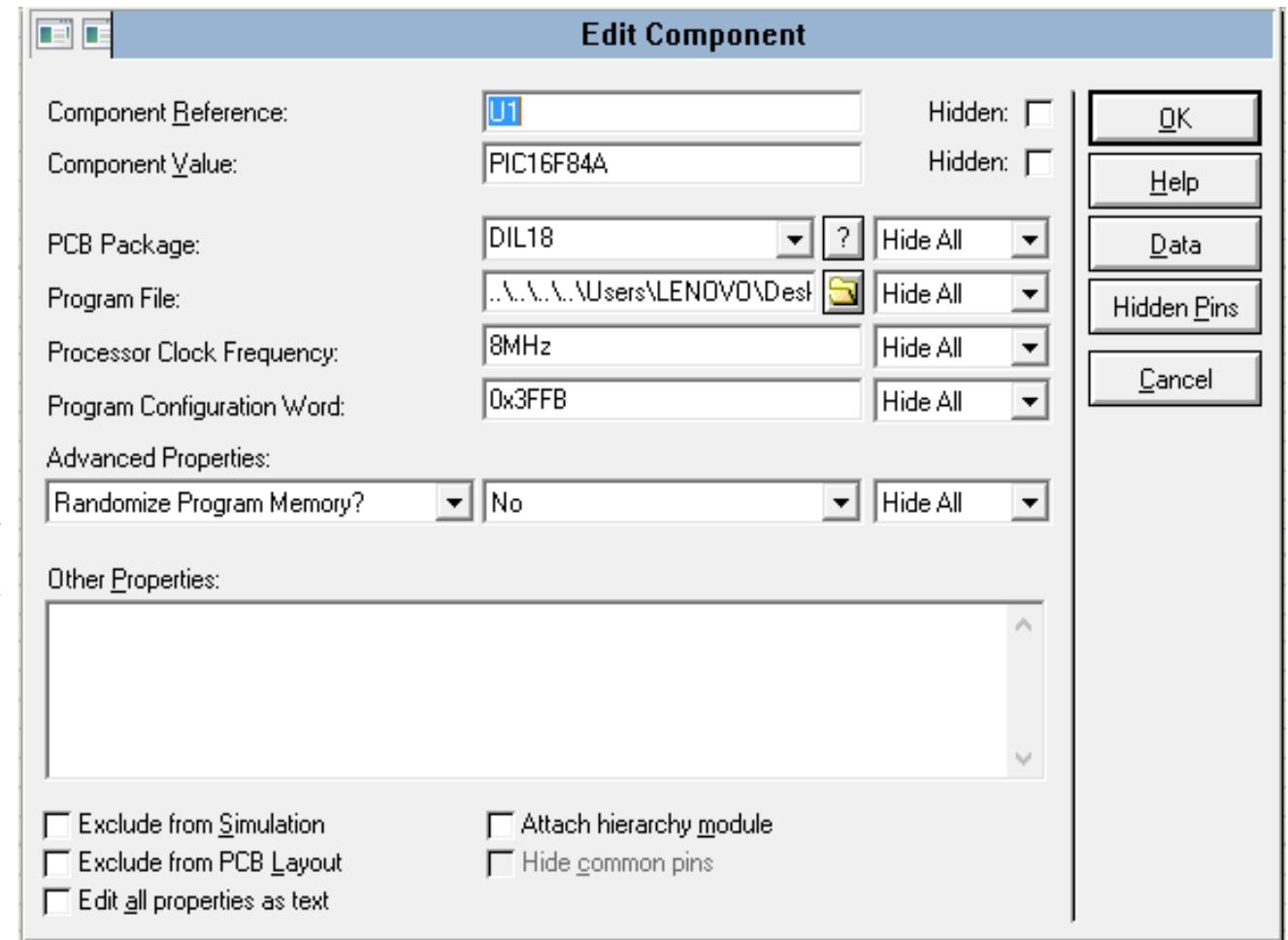
Procedure

Connect the circuit as shown in the figure.



Procedure (Cont.)

Double click on the **pic16F84A**, then the window in the figure show to download the file of extension of (**.hex**) and select the processor clock frequency (in this experiment we select the frequency is **8MHZ** the same frequency of the pic in the mikroC program.



Procedure

Write the code in MikroC program.

```
// LCD module connections
sbit LCD_RS at RB2_bit;
sbit LCD_EN at RB3_bit;
sbit LCD_D4 at RB4_bit;
sbit LCD_D5 at RB5_bit;
sbit LCD_D6 at RB6_bit;
sbit LCD_D7 at RB7_bit;

sbit LCD_RS_Direction at TRISB2_bit;
sbit LCD_EN_Direction at TRISB3_bit;
sbit LCD_D4_Direction at TRISB4_bit;
sbit LCD_D5_Direction at TRISB5_bit;
sbit LCD_D6_Direction at TRISB6_bit;
sbit LCD_D7_Direction at TRISB7_bit;
// End LCD module connections
```

```
void main()
{
    trisa.b0 = 1; //A0 as input
    trisa.b1 = 1; //A1 as input

    Lcd_Init(); // Initialize LCD

    Lcd_Cmd(_LCD_CLEAR); // Clear display

    Lcd_Cmd(_LCD_CURSOR_OFF); // Cursor off

    lcd_out(1,1,"Welcome");
    delay_ms(2000);
    Lcd_Cmd(_LCD_CLEAR); // Clear display

    while(1) //infinite loop
    {
        if(porta.b0 == 0) //if A0 was clicked
        {
            Lcd_Cmd(_LCD_CLEAR); // Clear display
            lcd_out(1,1,"Switch 1");
        }
    }
}
```

Procedure

Write the code in MikroC program.

```
    delay_ms(1000);
    Lcd_Cmd(_LCD_CLEAR); // Clear display

    delay_ms(200);
}

if(porta.b1 == 0) //if A1 was clicked
{
    Lcd_Cmd(_LCD_CLEAR); // Clear display
    lcd_out(1,1,"Switch 2");
    delay_ms(1000);
    Lcd_Cmd(_LCD_CLEAR); // Clear display

    delay_ms(200);
}

}
```

Discussion

- 1- Write and design the program to display on the LCD by using protues and mikriC. Display on the 1st row (Hello) word after 1s clear the LCD and display on the 2nd row (Guys)word. This display continues in the infinite loop.
- 2- Write and design the program to display on the LCD by using protues and mikriC. Display the all aphanitic letters on the LCD, letter after 0.5s the another letter.

EXPERIMENT 04

7 Segments



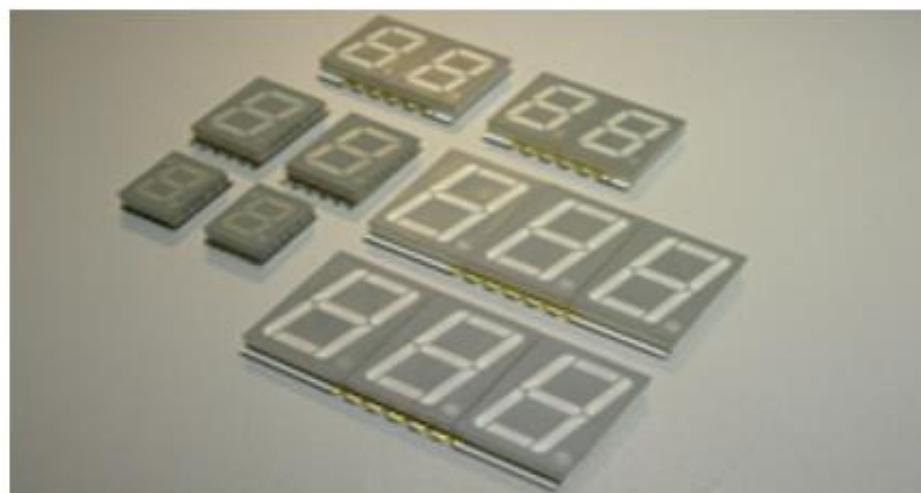
Experiment Object

The object of this experiment is to learn how to use 7segments with microcontroller.

Theory

7Segments

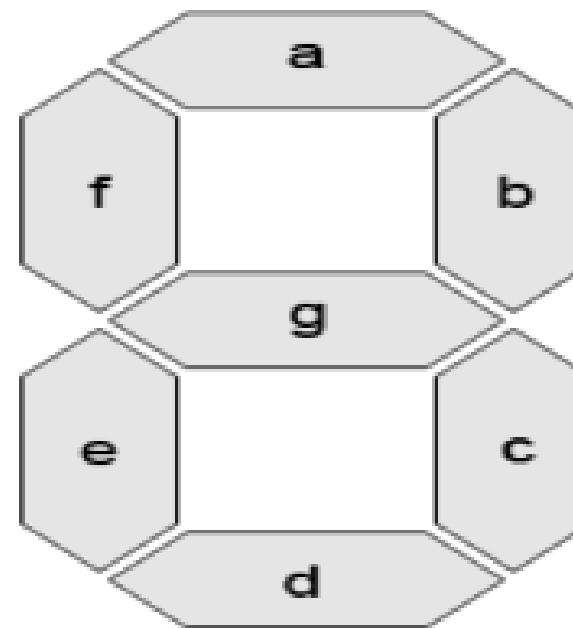
The object of this experiment is to learn how to use 7segments with microcontroller.



✿ Theory (Cont.)

□ 7Segments (Cont.)

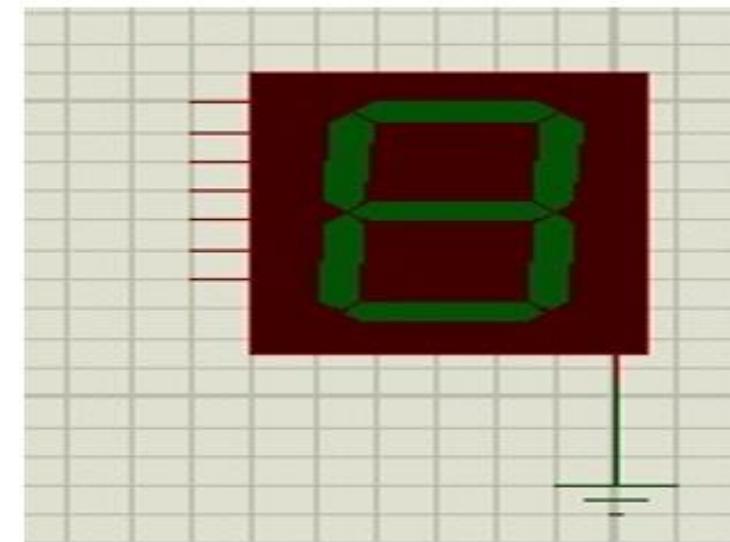
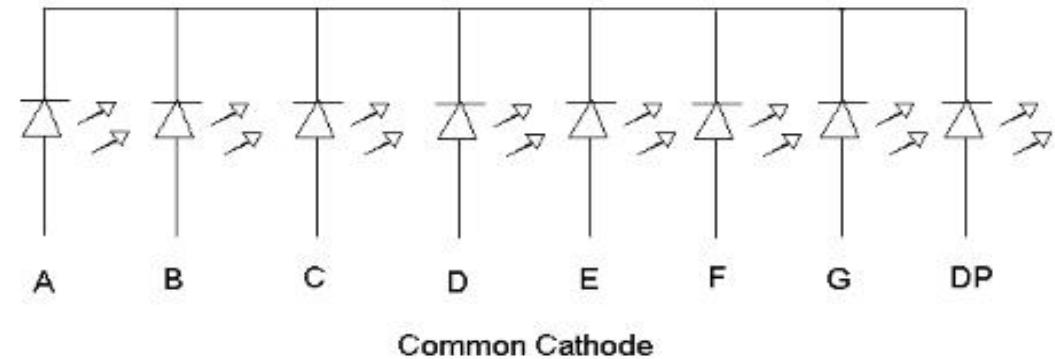
Is a seven internally connected LEDs used to display values.



✿ Theory (Cont.)

□ 7Segments (Cont.)

The internal connection is either grounded, all the ends of the negative LEDs are grounded.

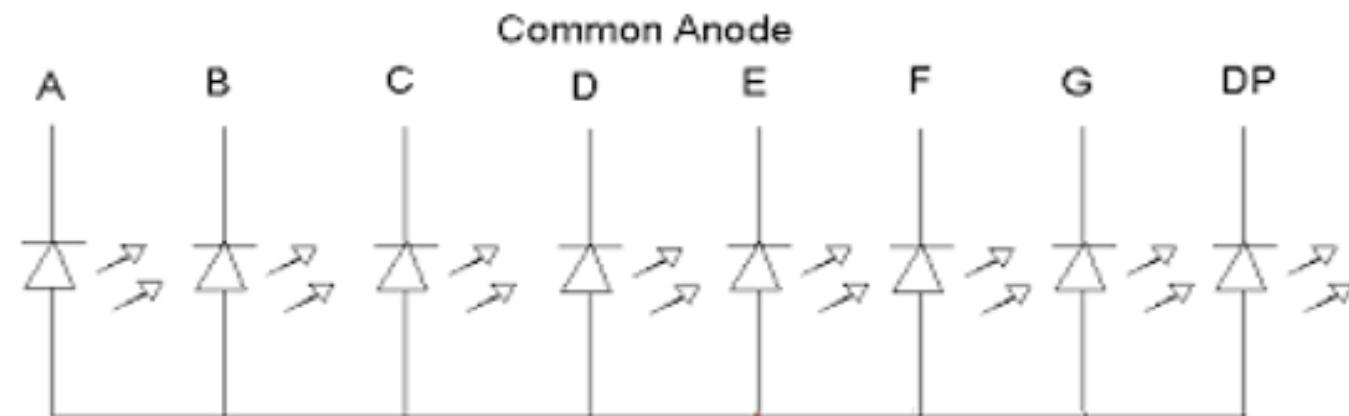


❖ Theory (Cont.)

□ 7Segments (Cont.)

In this case the LEDs are operated under the source voltage of any logical one.

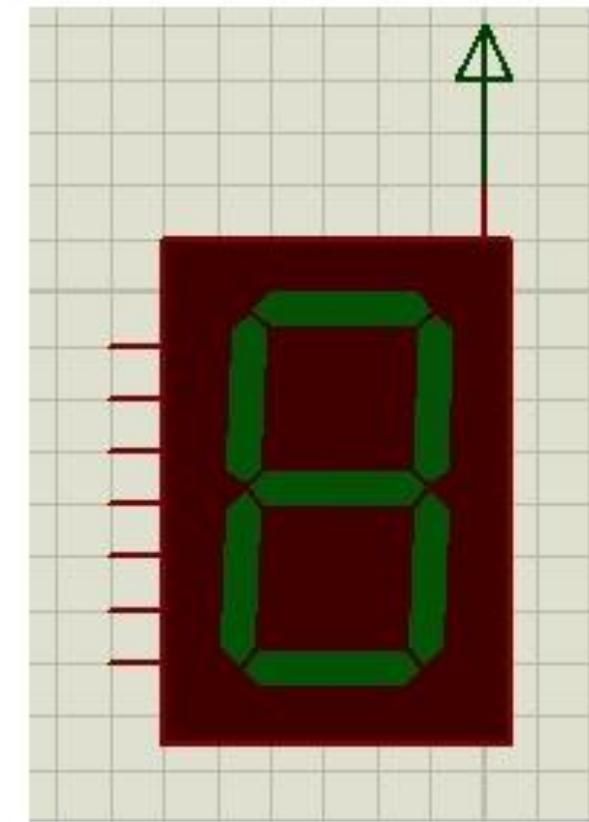
If the internal conductivity of the females is positive for the source voltage, that is, all the ends of the positive LEDs are shared.



✿ Theory (Cont.)

□ 7Segments (Cont.)

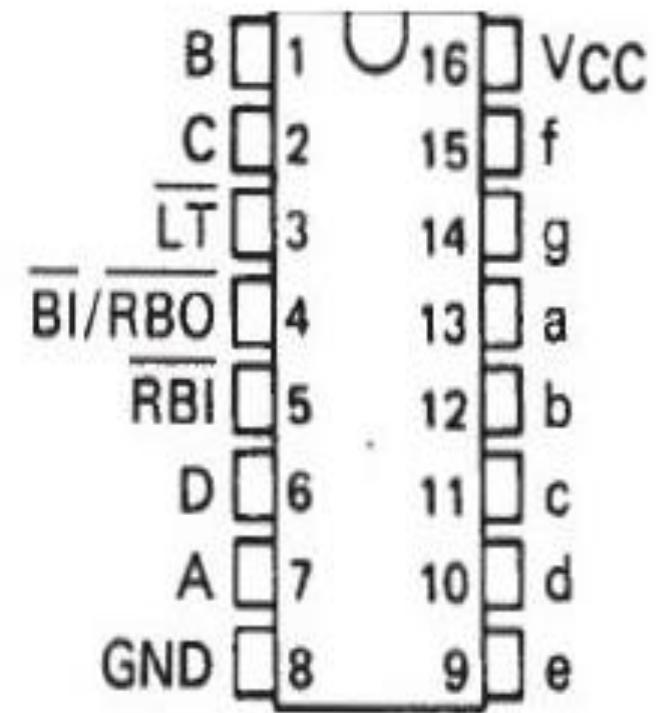
In this case, the LEDs are run with a logical value of 0.



✿ Theory (Cont.)

Decoder

Is an integrated circuit that encrypts (that is, converts binary numbers to decimal numbers and outputs them to seven segments display).



✿ Theory (Cont.)

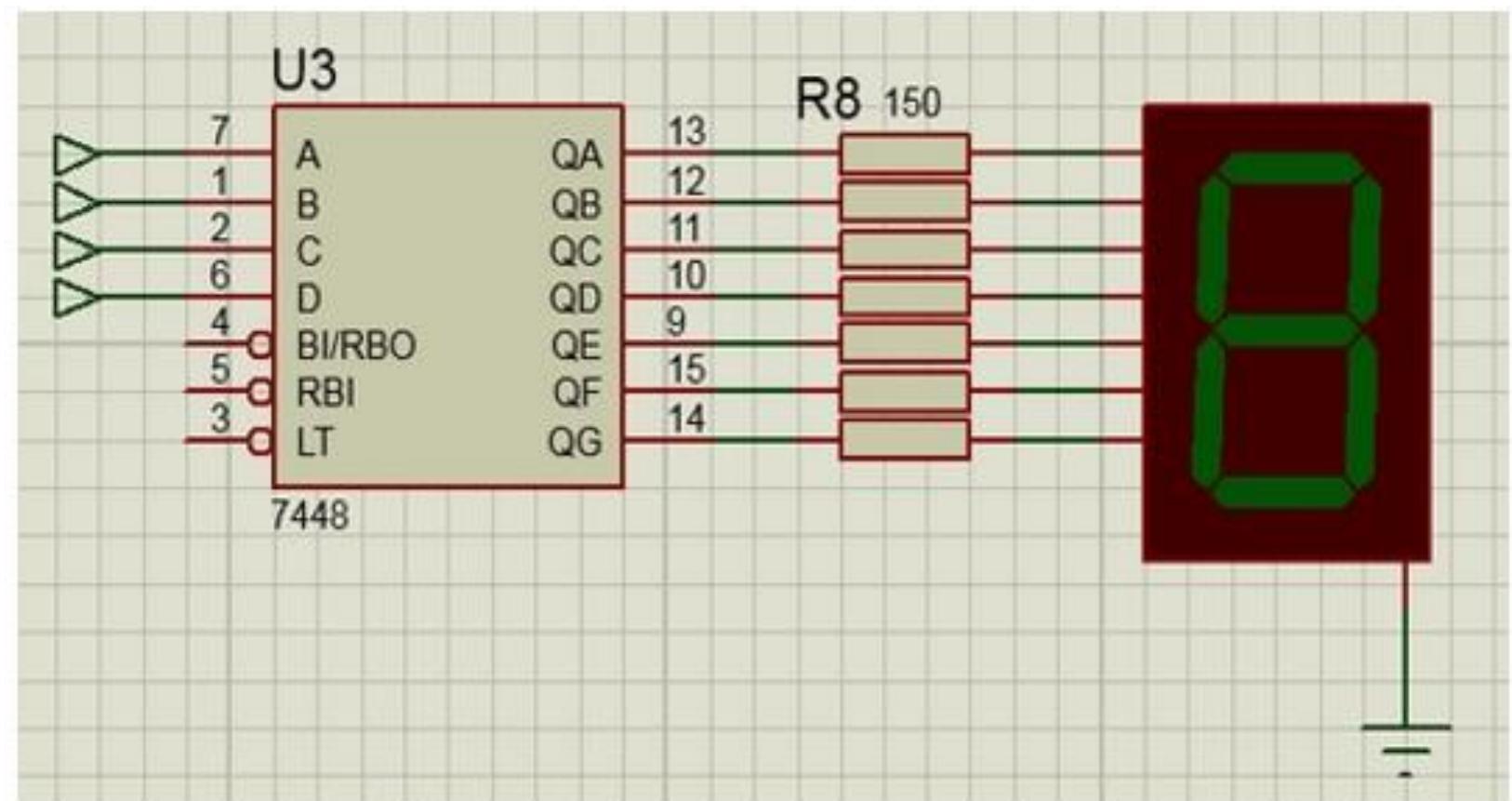
□ Decoder (Cont.)

Advantage:

- Display values easily
- Provide microcontroller electrodes

Connect the encoder with the microcontroller in the case that the 7segments comment cathode:

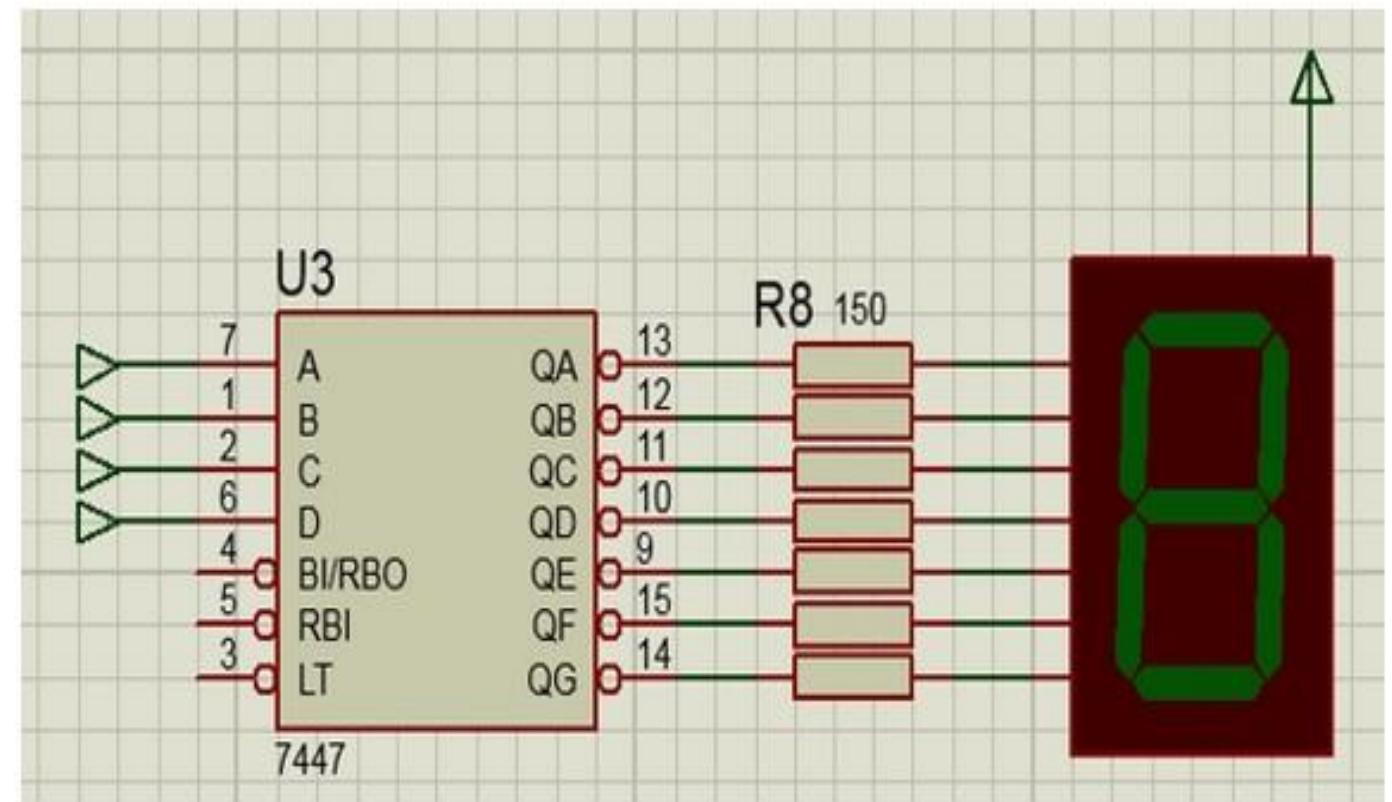
✿ Theory (Cont.)
□ Decoder with 7 Seg.



✿ Theory (Cont.)

□ Decoder with 7 Seg.

Connect the encoder with the microcontroller in the case that the 7segments comment cathode:

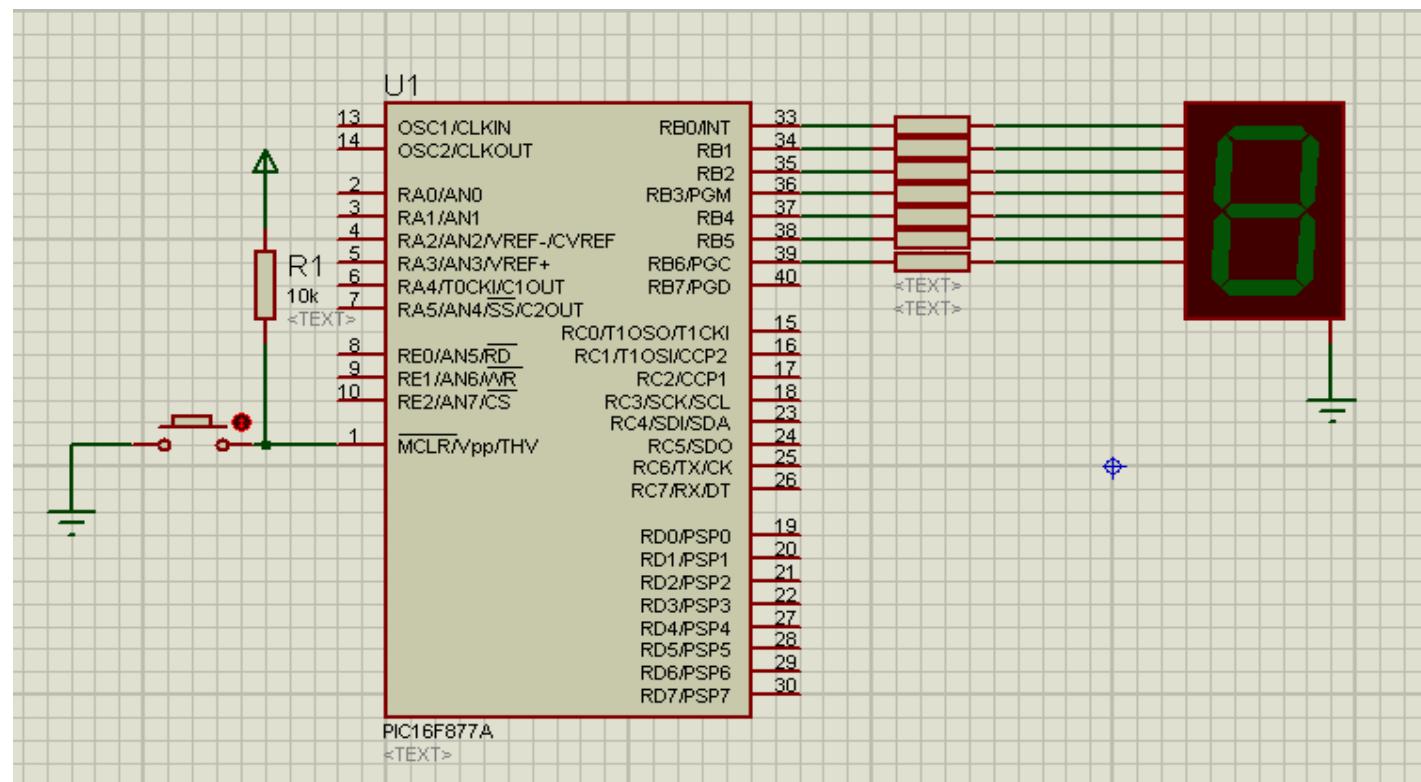


✿ The Experiment Components

- Microcontroller Pic 16F877A.
- 7segments (common cathode).
- Resistor 150 ohm.
- Push Button.

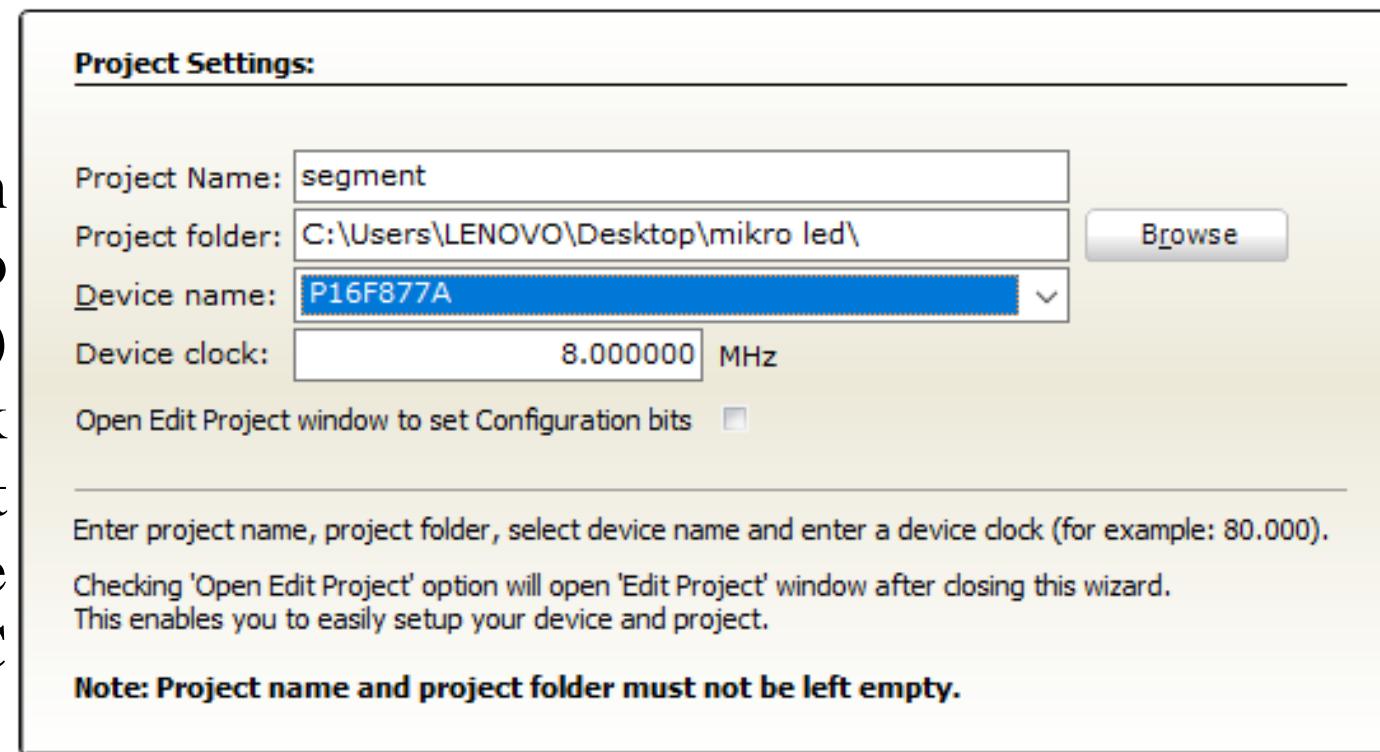
Procedure

Connect the circuit as shown in the figure.



Procedure (Cont.)

Double click on the **pic16F877A**, then the window in the figure show to download the file of extension of (**.hex**) and select the processor clock frequency (in this experiment we select the frequency is **8MHZ** the same frequency of the pic in the mikroC program.



```
char array[10] = {63, 6, 91, 79, 102, 109, 124, 7, 127, 103}; //stores numbers in array
char index; //index points to array elements

// initialization function
void init_function()
{
    trisb = 0b00000000; //port B as output
    portb = 0b00000000; //initial value for port B
}

void main()
{
    init_function(); //start with the initializations

    for(index = 0; index < 10; index++)
    {
        portb = array[index];
        delay_ms(500);
    }
}
```

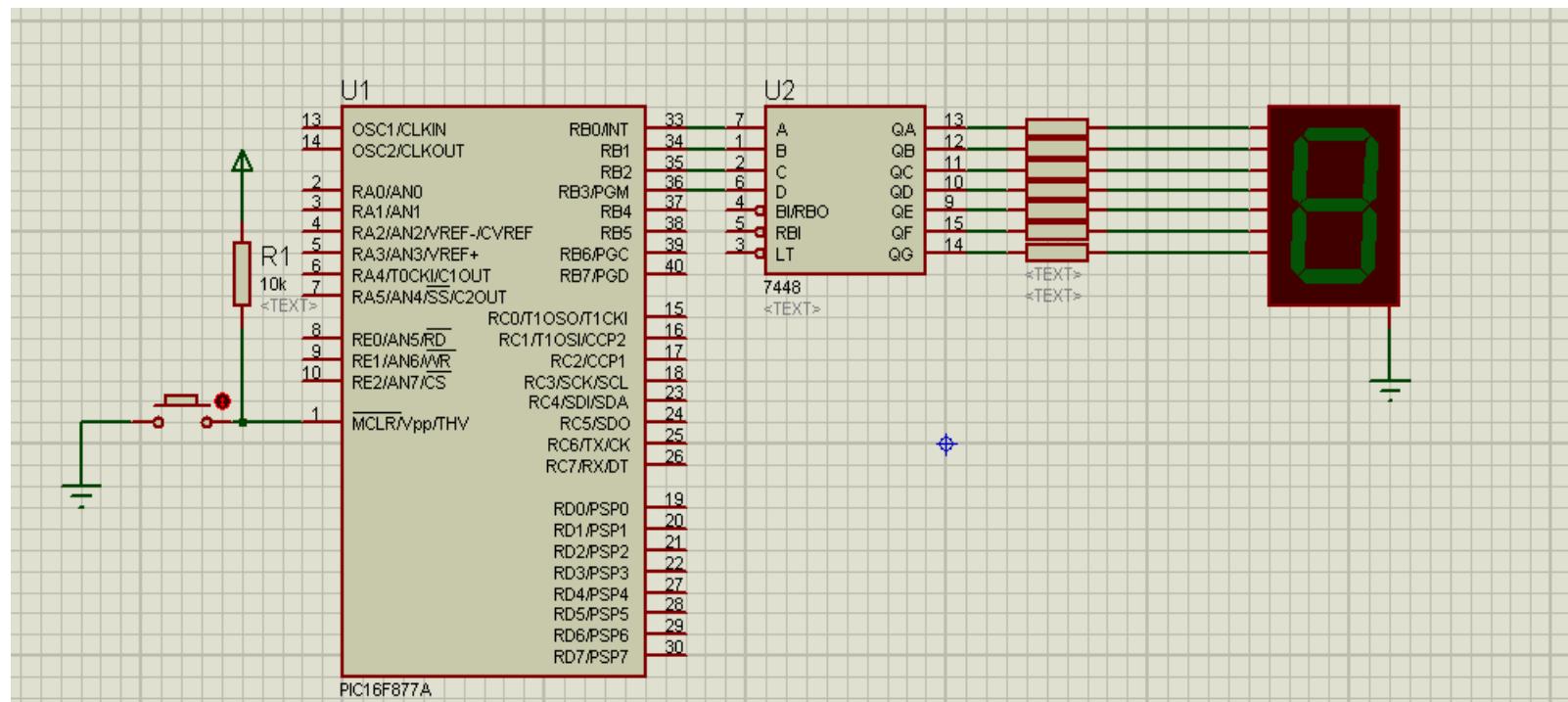
Procedure (Cont.)

Write the code in MikroC program.

Procedure (Cont.)

7seg with decoder:

Connect the circuit as shown in the figure.



❖ Procedure (Cont.)

7seg with decoder:

Connect the circuit as shown in the figure.

```
char counter;

void main()
{
    trisb = 0b00000000; //port B as output
    portb = 0b00000000; //initial values for port B

    for(counter=0; counter < 4; counter++)
    {
        portb = counter;
        delay_ms(1000);
    }
}
```

Discussion

- 1- Write and design the program to display on the 7seg by using protues and mikriC. Connect two 7seg with microcontroller and display (0-99) numbers.

EXPERIMENT 05

Keypad



Experiment Object

The object of this experiment is to learn how to use 7segments with microcontroller.

Theory

Keypad



✿ Theory

□ Keypad (Cont.)

Example of Connecting the keyboard with port D in the microcontroller.

C1 --> D0

C2 --> D1

C3 --> D2

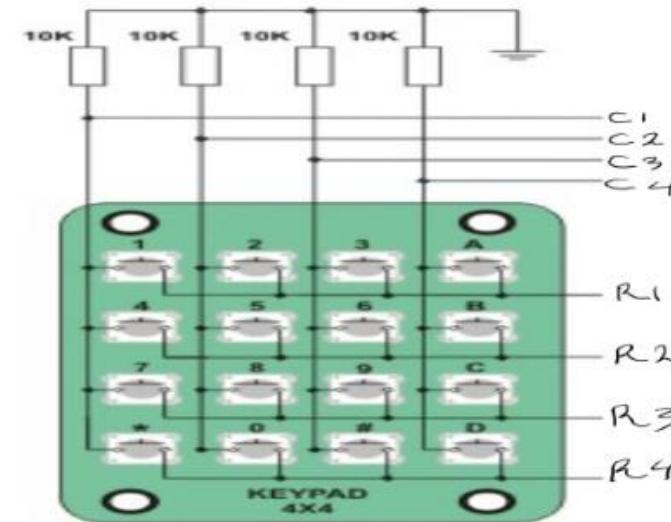
C4 --> D3

R1 --> D4

R2 --> D5

R3 --> D6

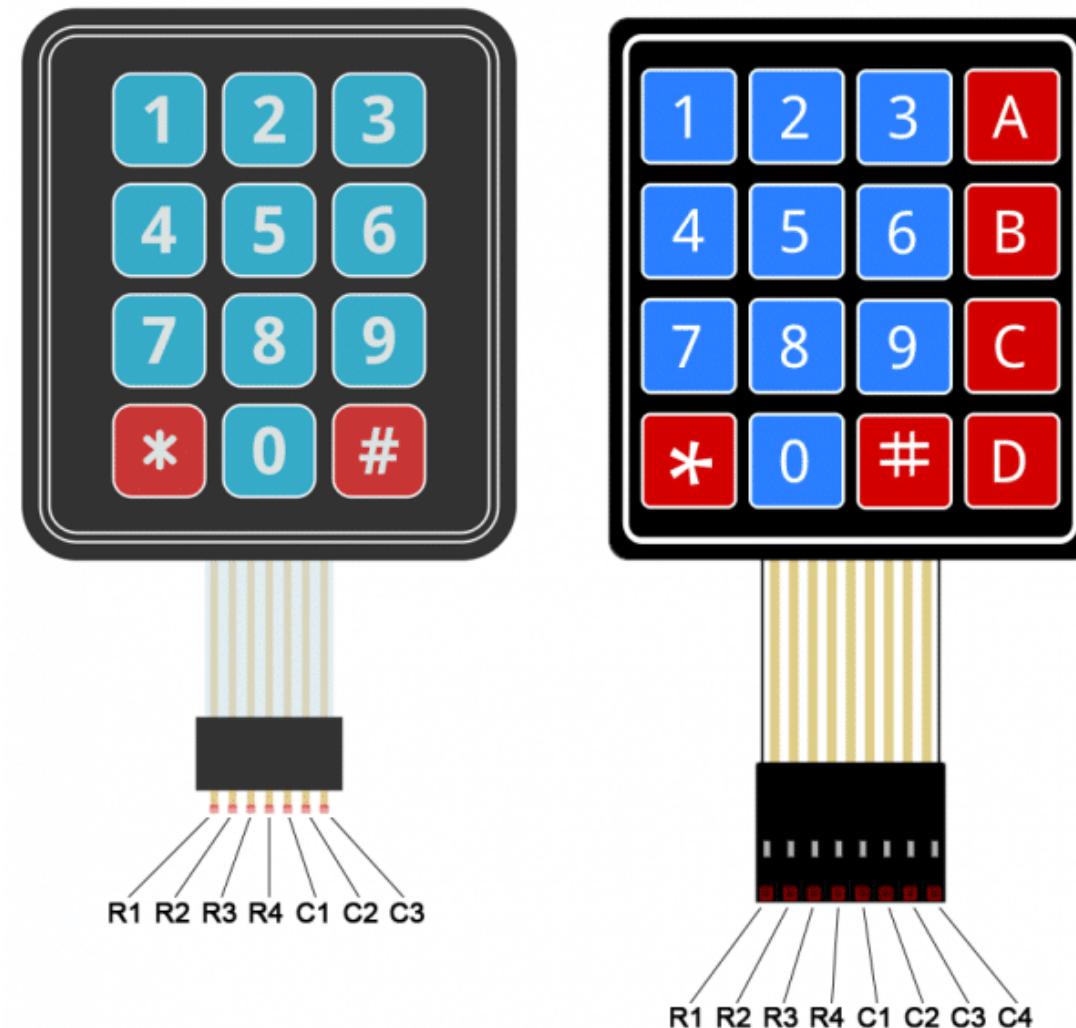
R4 --> D7



Theory

□ Keypad (Cont.)

Keypad Module.

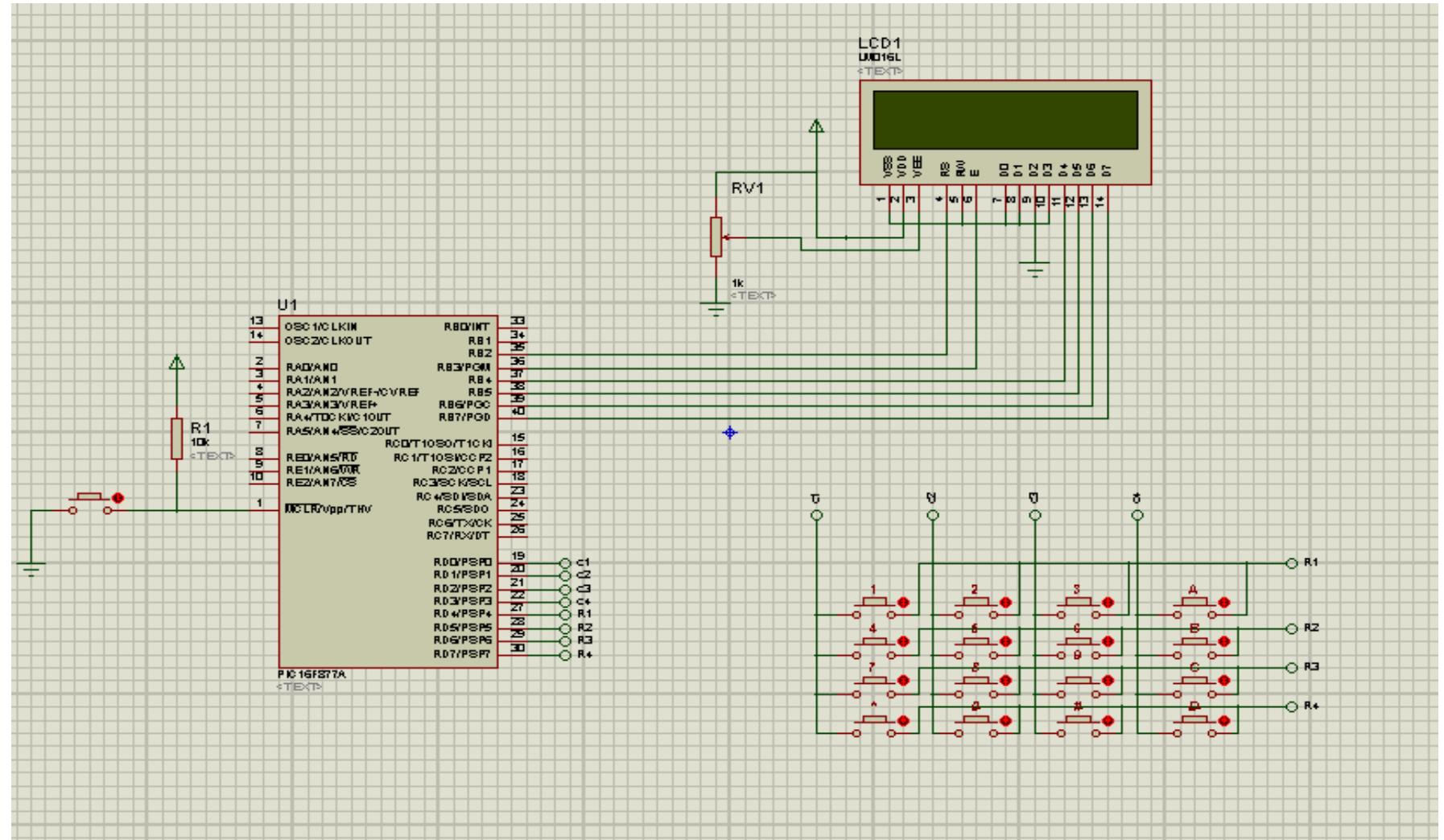


✿ The Experiment Components

- Microcontroller Pic 16F877A.
- LCD 16x2.
- Resistor 150 ohm.
- Push Button.

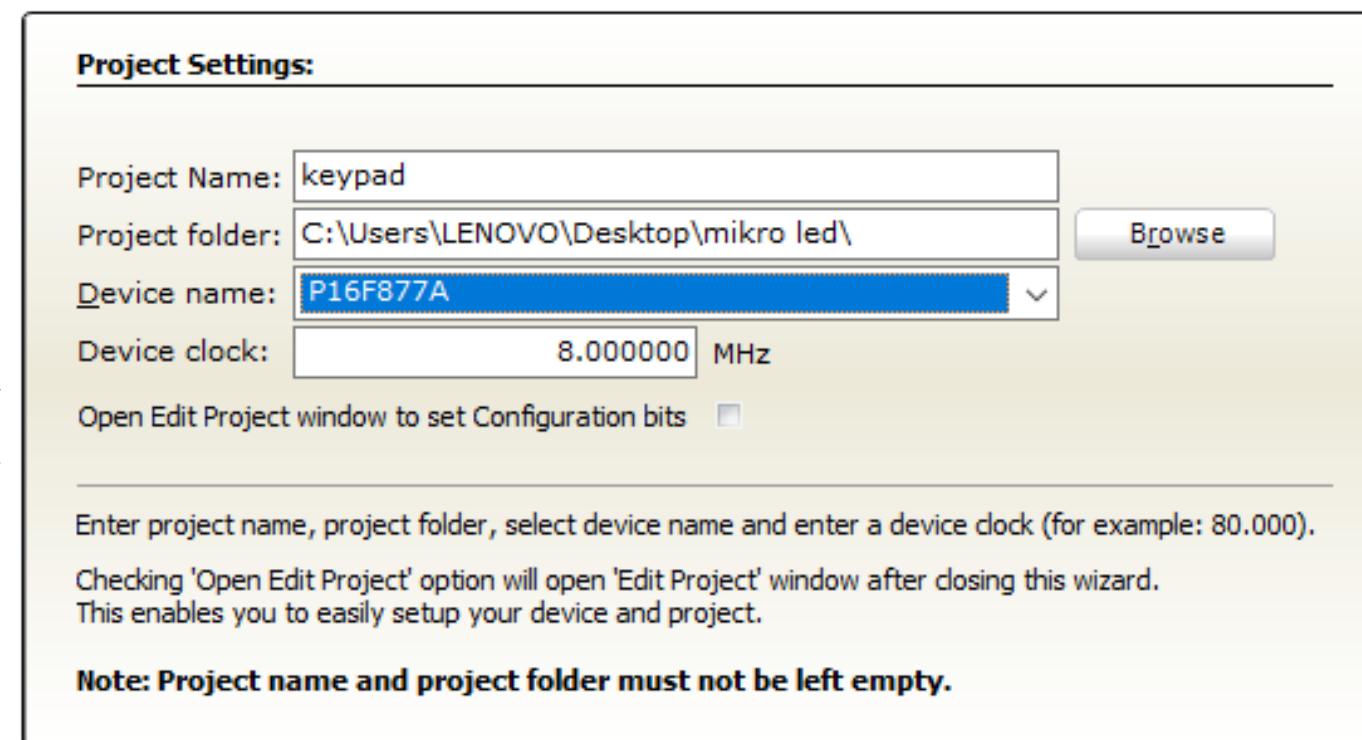
Procedure

Connect the circuit as shown in the figure.



Procedure (Cont.)

Double click on the **pic16F877A**, then the window in the figure show to download the file of extension of (**.hex**) and select the processor clock frequency (in this experiment we select the frequency is **8MHZ** the same frequency of the pic in the mikroC program.



Procedure (Cont.)

Write the code in **MikroC** program.

```
// Keypad module connections
char keypadPort at PORTD;
// End Keypad module connections

// LCD module connections
sbit LCD_RS at RB2_bit;
sbit LCD_EN at RB3_bit;
sbit LCD_D4 at RB4_bit;
sbit LCD_D5 at RB5_bit;
sbit LCD_D6 at RB6_bit;
sbit LCD_D7 at RB7_bit;

sbit LCD_RS_Direction at TRISB2_bit;
sbit LCD_EN_Direction at TRISB3_bit;
sbit LCD_D4_Direction at TRISB4_bit;
sbit LCD_D5_Direction at TRISB5_bit;
sbit LCD_D6_Direction at TRISB6_bit;
sbit LCD_D7_Direction at TRISB7_bit;
// End LCD module connections
```

Procedure (Cont.)

Write the code in **MikroC** program.

```
unsigned short kp;

void main()
{
    Keypad_Init(); // Initialize Keypad

    Lcd_Init(); // Initialize LCD
    Lcd_Cmd(_LCD_CLEAR); // Clear display
    Lcd_Cmd(_LCD_CURSOR_OFF); // Cursor off

    Lcd_Out(1, 1, "Key :"); // Write message text on LCD

    do //enter do loop
    {
        kp = 0; // Reset key code variable

        do // Wait for key to be pressed and released
        {
            kp = Keypad_Key_Click(); // Store key code in kp variable
        }while (!kp); //stay in the loop if kp = 0, (if no button is pressed)

        switch (kp) //transform key to char to be displayed on LCD
        {
```

Procedure (Cont.)

Write the code in **MikroC** program.

```
case 1: kp = '1'; break; // 1
case 2: kp = '2'; break; // 2
case 3: kp = '3'; break; // 3
case 4: kp = 'A'; break; // A
case 5: kp = '4'; break; // 4
case 6: kp = '5'; break; // 5
case 7: kp = '6'; break; // 6
case 8: kp = 'B'; break; // B
case 9: kp = '7'; break; // 7
case 10: kp = '8'; break; // 8
case 11: kp = '9'; break; // 9
case 12: kp = 'C'; break; // C
case 13: kp = '*'; break; // *
case 14: kp = '0'; break; // 0
case 15: kp = '#'; break; // #
case 16: kp = 'D'; break; // D
}

Lcd_Ch(1, 10, kp); // Print pressed key on LCD

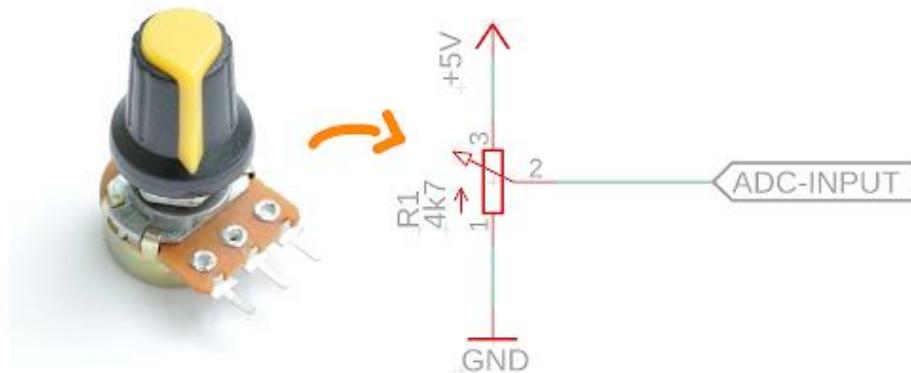
}while (1); //stay in the loop for ever
}
```

Discussion

- 1- Write and design the program to make the simple calculator to do the mathematical operations by using keypad.

EXPERIMENT 06

ADC-Potentiometer



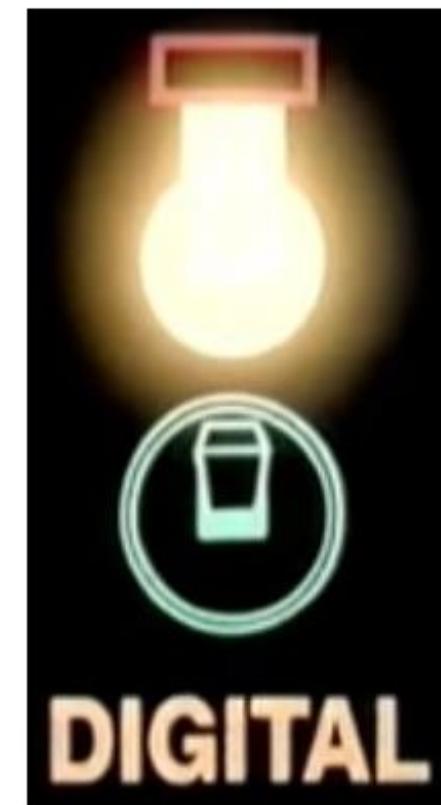
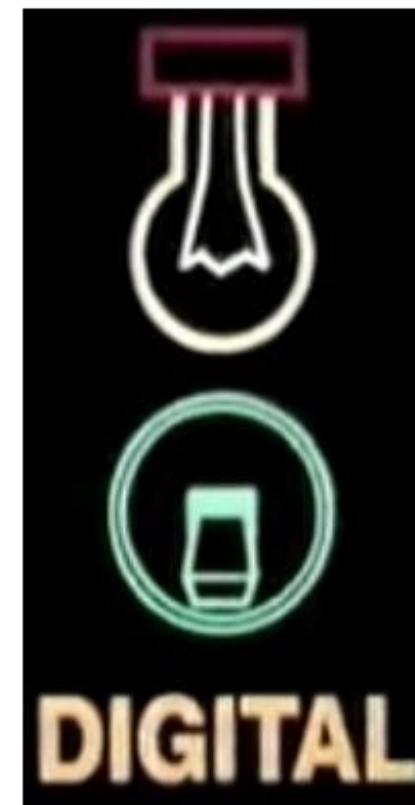
Experiment Object

The object of this experiment is to learn how to use analog signal with microcontroller.

Theory

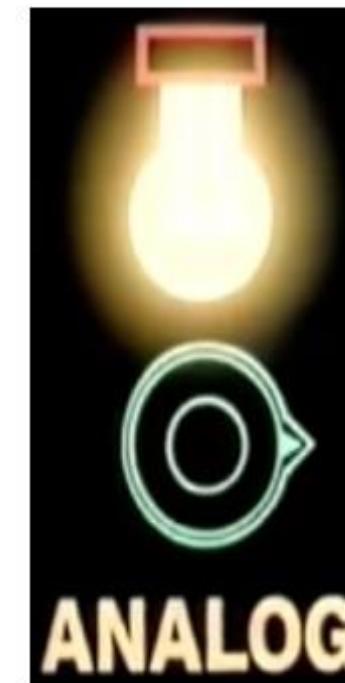
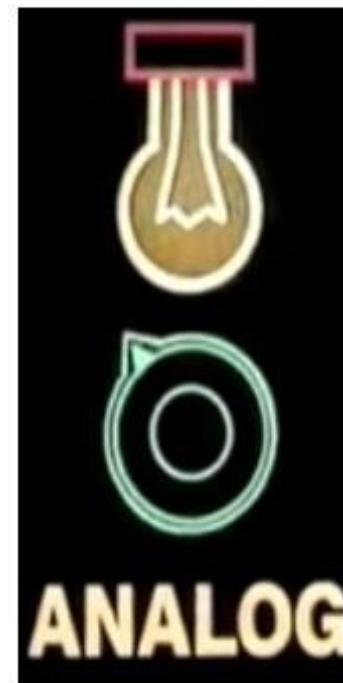
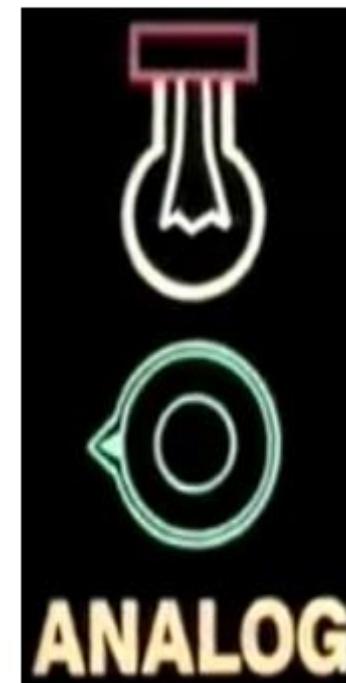
- **Digital:** In digital circuits the voltage is constant at a certain value and is a constant current (DC) and the exchange of data in which only digital is either zero or one.
- **Analog:** In analog circuits the voltage is unstable and the current type is either DC or AC.
- **Note** that in digital circuits we can not control the characteristics of devices only we can turn it on and off like a bulb
- Note the bulb in the event that the connection is digital:

✿ Theory (Cont.)



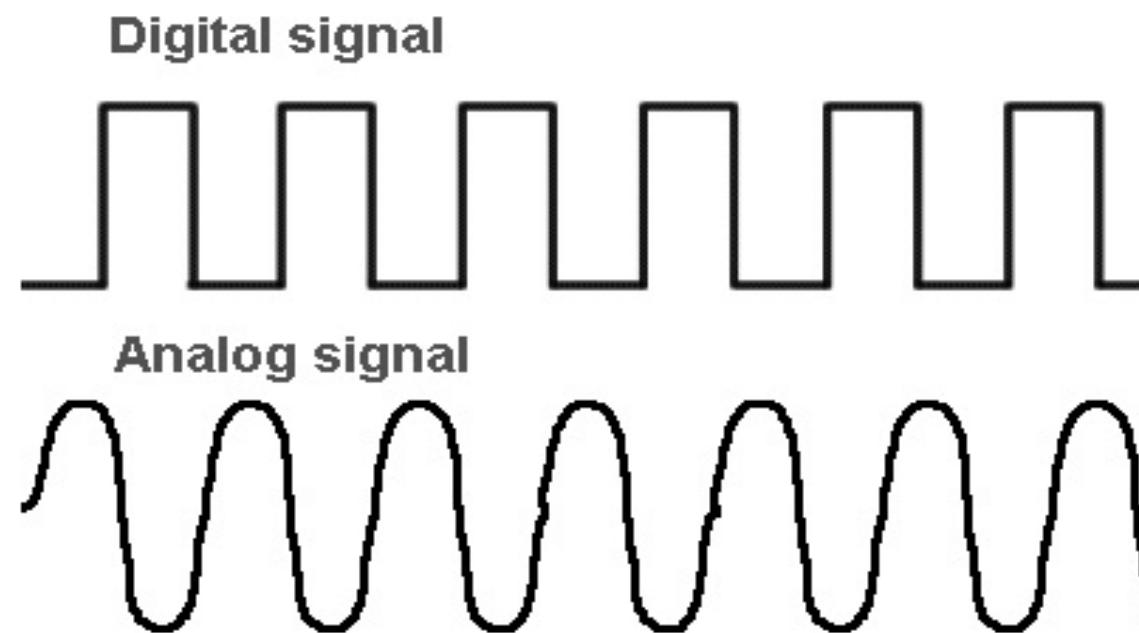
✿ Theory (Cont.)

Note the bulb in the event that the connection is **analog**:



✿ Theory (Cont.)

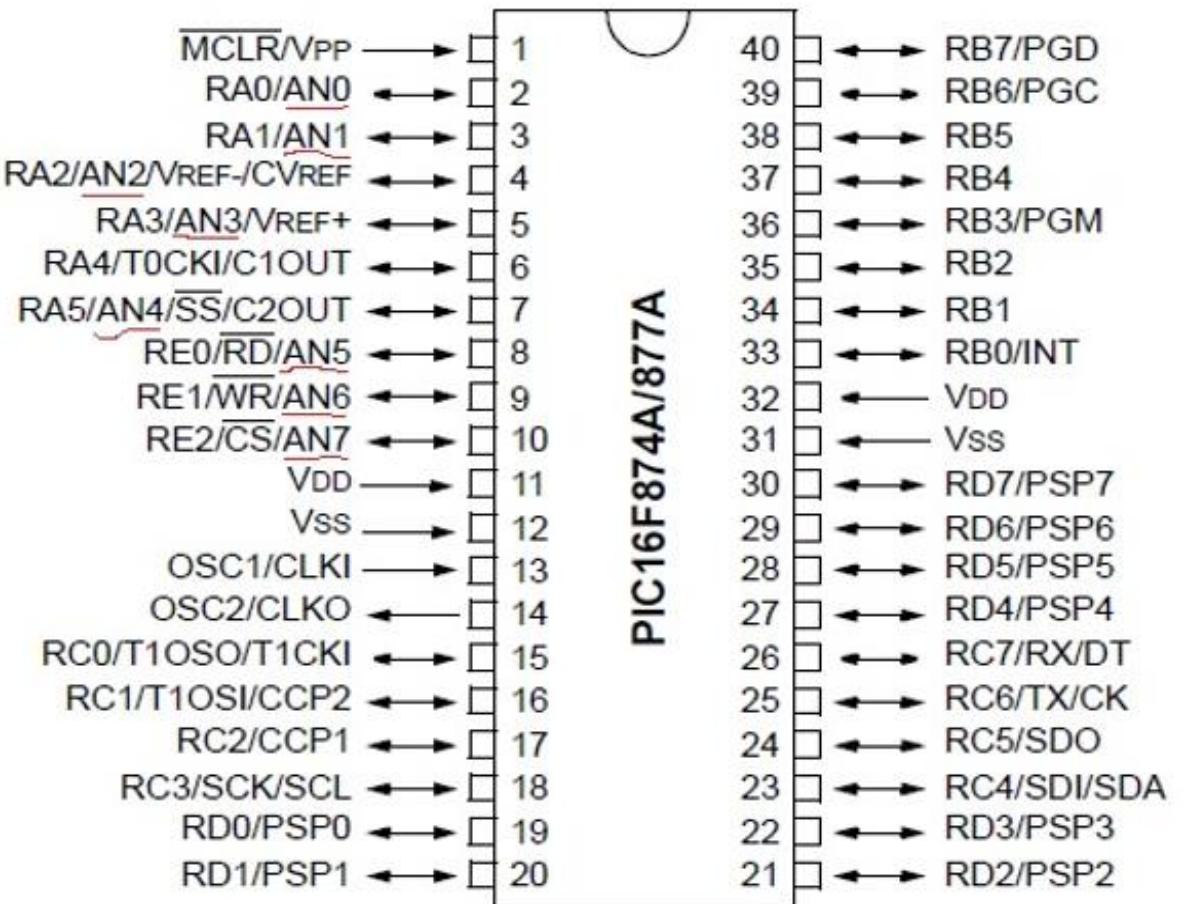
Note the image in the form of digital and analog signal:



❖ Theory (Cont.)

Since the controller is digital, how do you understand the analog devices connected to it?

For this, there are special electrodes in the controller called an abbreviation for **Analog**. Analog devices are connected through these electrodes:

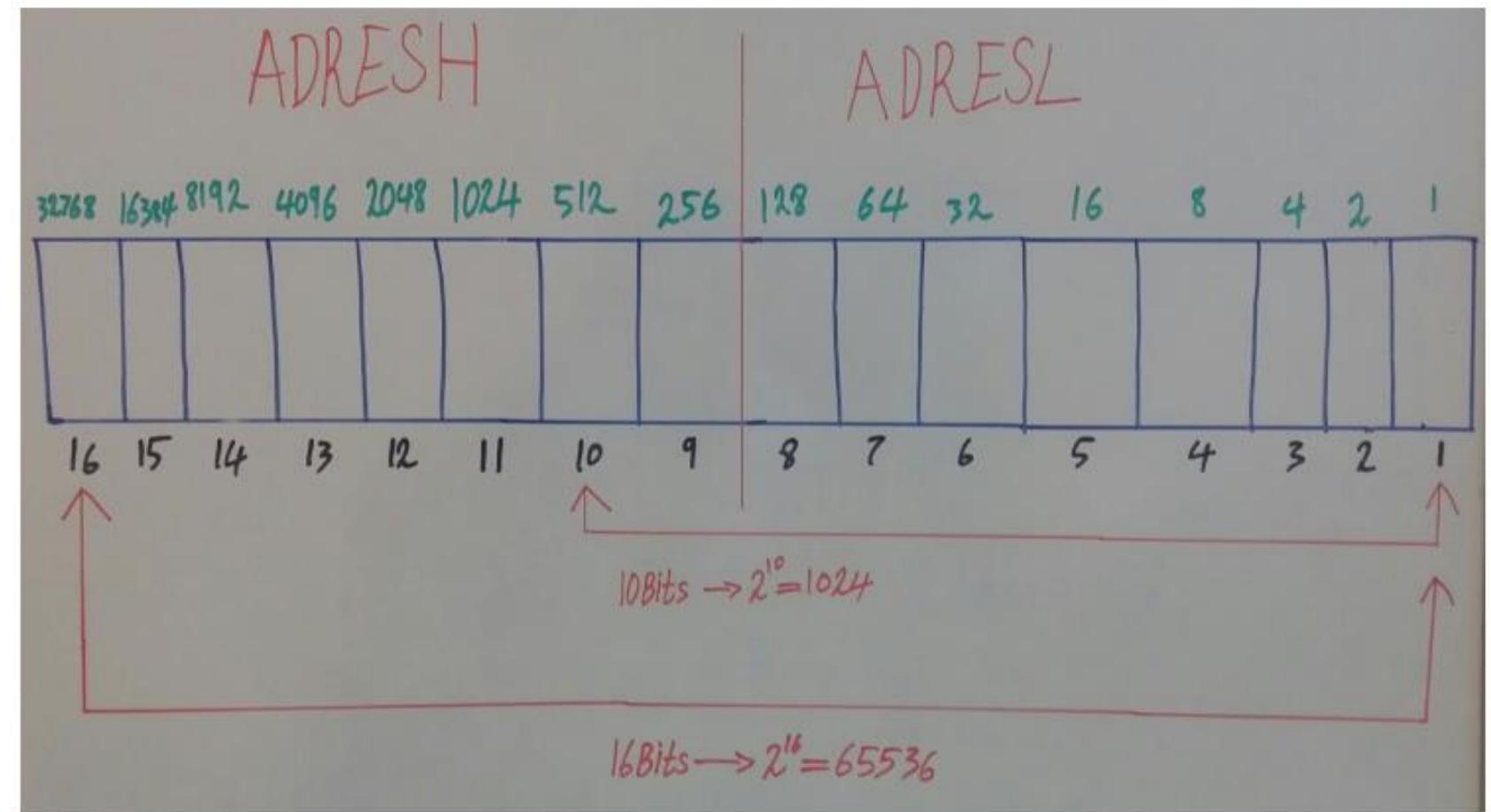


❖ Theory (Cont.)

But before using these electrodes must be defined as analog poles through their own recorders in the controller and also must be defined as an electrode as income.

- There are four recorders in the controller to control the functions of the analog poles:
 - 1- **ADCON0**: to control the functions of digital analog switches.
 - 2- **ADCON1**: to determine which electrodes will be used.
 - 3- **ADRESH**: saves the results of analog to digital conversions.
 - 4- **ADRESL**: saves the results of analog to digital conversions.

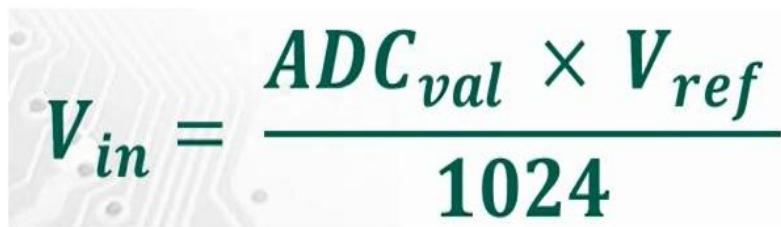
✿ Theory (Cont.)



✿ Theory (Cont.)

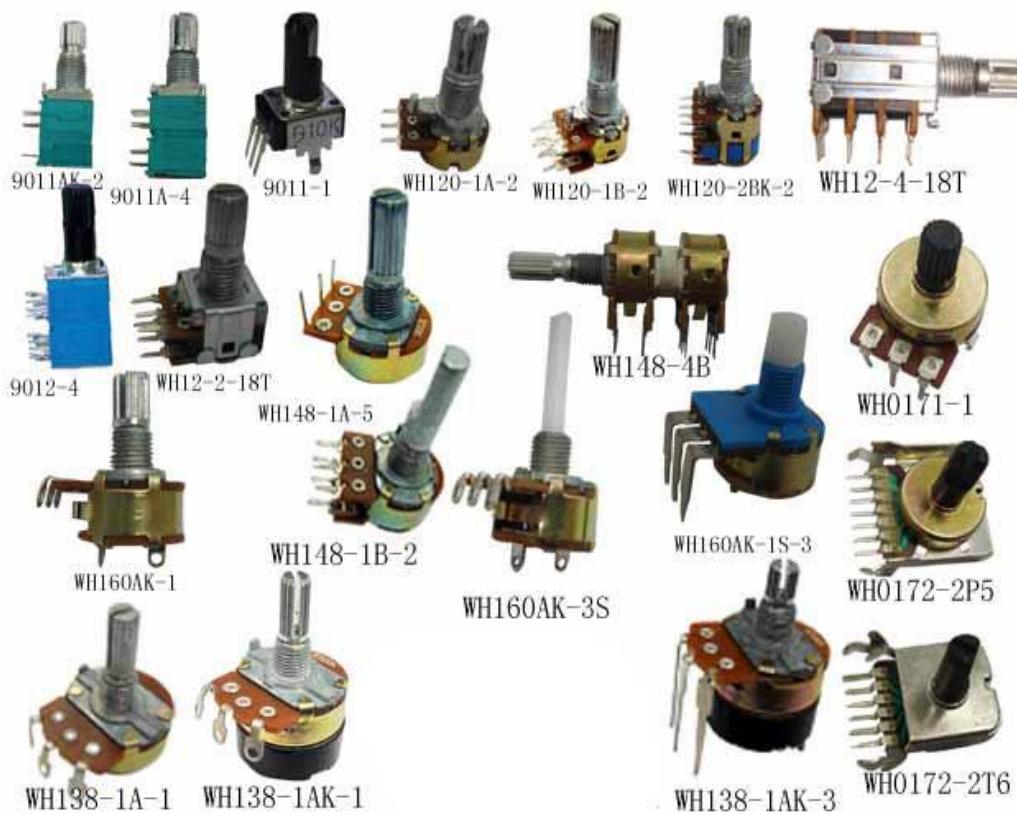
Voltage Reference

The digital analog switch needs a reference voltage to complete the transformations. The reference voltage will be used to compare with the values read. The reference voltage must be equal to or greater than the voltage to be read.


$$V_{in} = \frac{ADC_{val} \times V_{ref}}{1024}$$

$$ADC_{val} = \frac{V_{in} \times 1024}{V_{ref}}$$

Theory (Cont.) Potentiometer

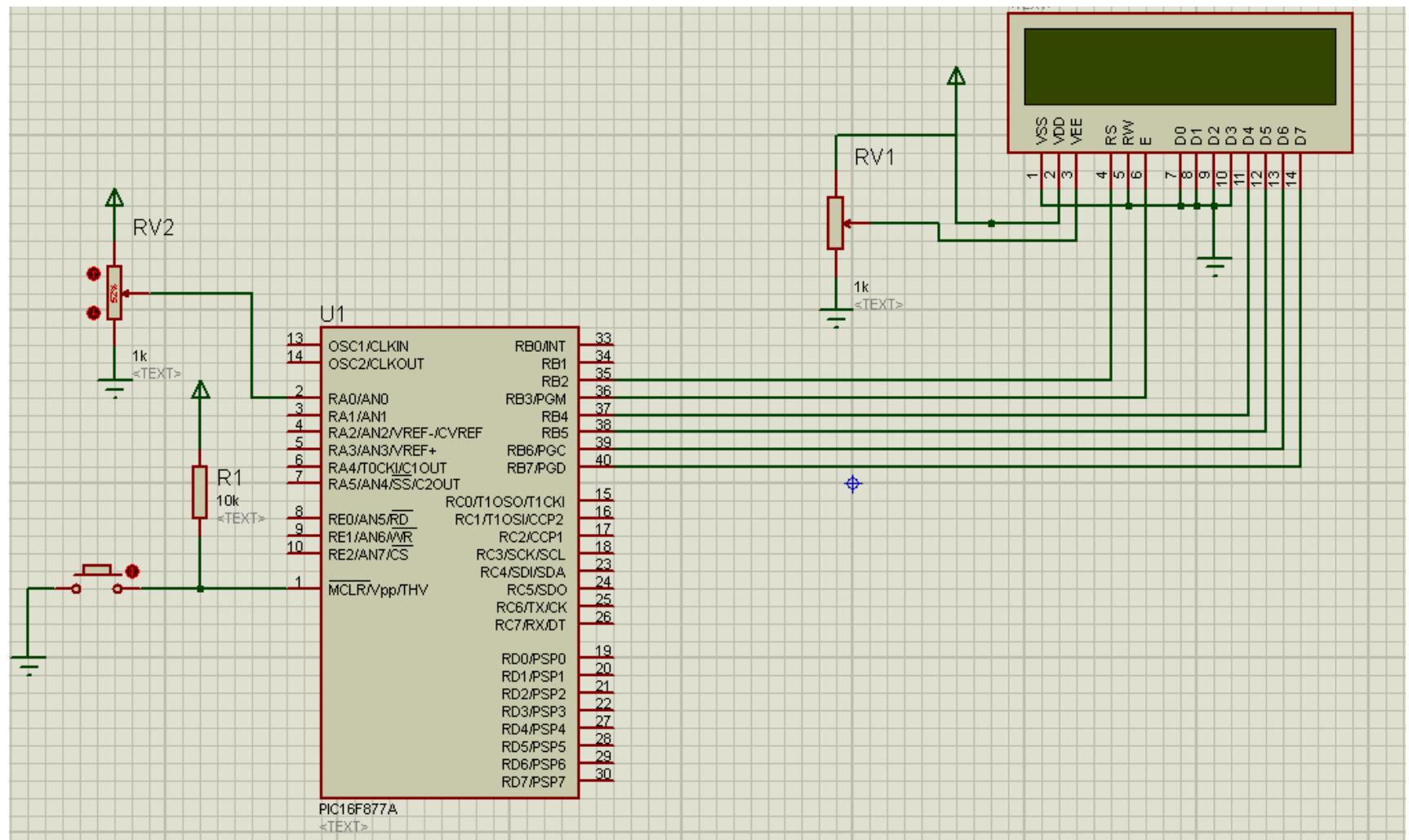


✿ The Experiment Components

- Microcontroller Pic 16F877A.
- 16x2 LCD.
- Resistor 150 ohm.
- Potentiometer.
- Push Button.

Procedure

Connect the circuit as shown in the figure.



Procedure (Cont.)

Write the code in MikroC program to display the image.

```
// LCD module connections
sbit LCD_RS at RB2_bit;
sbit LCD_EN at RB3_bit;
sbit LCD_D4 at RB4_bit;
sbit LCD_D5 at RB5_bit;
sbit LCD_D6 at RB6_bit;
sbit LCD_D7 at RB7_bit;

sbit LCD_RS_Direction at TRISB2_bit;
sbit LCD_EN_Direction at TRISB3_bit;
sbit LCD_D4_Direction at TRISB4_bit;
sbit LCD_D5_Direction at TRISB5_bit;
sbit LCD_D6_Direction at TRISB6_bit;
sbit LCD_D7_Direction at TRISB7_bit;
// End LCD module connections

unsigned int adc_value;
char adc_value_txt[10];

float volt;
char volt_txt[15];
```

Procedure (Cont.)

Write the code in MikroC program to display the image.

```
void main() {  
  
    ADC_Init(); // Initialize ADC module with default settings  
    ADCON1 = 0b00001110;  
  
    TRISA.B0 = 1; // PORT A0 as input  
  
    Lcd_Init(); // Initialize LCD  
    Lcd_Cmd(_LCD_CLEAR); // CLEAR display  
    Lcd_Cmd(_LCD_CURSOR_OFF); // Cursor off  
  
    do  
    {  
        adc_value = ADC_Read(0); //read analog from A0  
  
        //calculate voltage  
        volt = adc_value * 5;  
        volt = volt / 1024;  
  
        wordtostr(adc_value,adc_value_txt); //convert adc value to string  
        floattostr(volt,volt_txt); //convert voltage to string  
  
        Lcd_Out(1,1,adc_value_txt); //display adc value
```

PIC MICROCONTROLLER

06 EXPERIMENTS

ENG. ELAF A.SAEED

Discussion

1-

EXPERIMENT 07

GLCD



Experiment Object

The object of this experiment is to learn how to use 7segments with microcontroller.

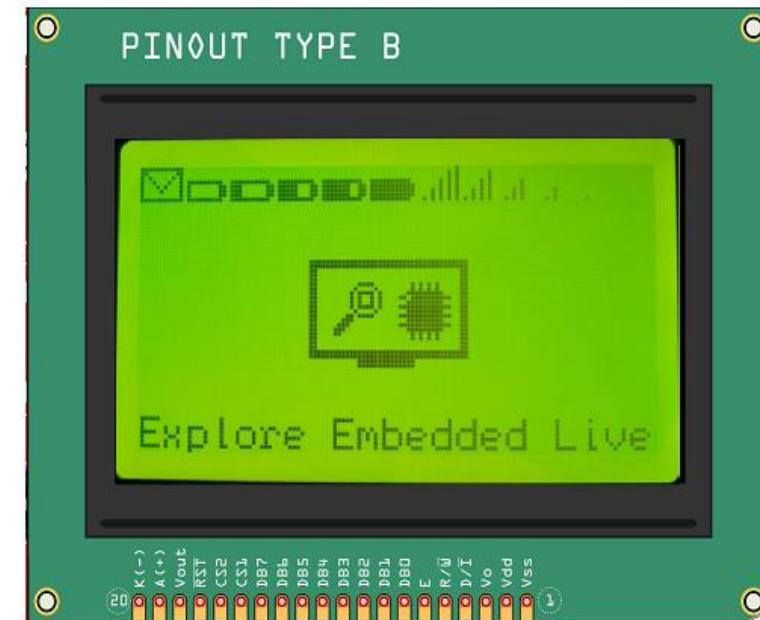
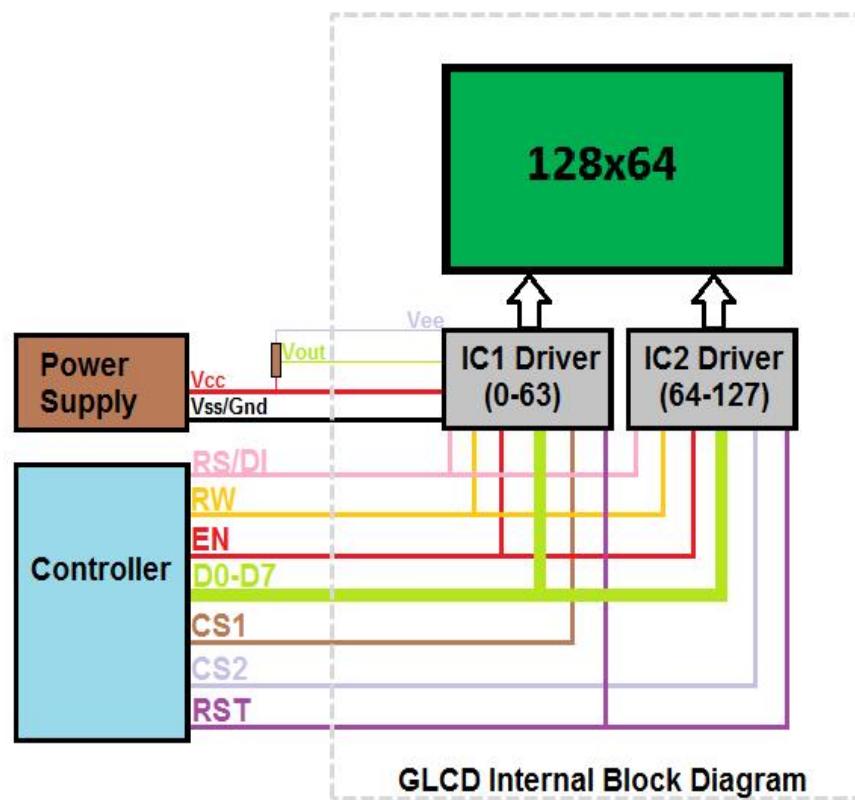
Theory

■ GLCD Internals and Pinout

Below image shows the internal block diagram of 128x64 GLCD along with its pin out.

Theory

■ GLCD Internals and Pinout



Theory (Cont.)

■ GLCD Internals and Pinout

As per the name it has 128pixels on X-axis and 64-pixels on Y-axis. Further the X-axis is divided into two parts of 64 pixels each and controlled by unique controller/driver IC as shown in the above image. Below table provides the detailed info of all the GLCD pins.

Theory (Cont.)

■ GLCD Internals and Pinout

Pin Number	Symbol	Pin Function
1	VSS	Ground
2	VCC	+5v
3	VO	Contrast adjustment (VO)
4	RS/DI	Register Select/Data Instruction. 0:Instruction, 1: Data
5	R/W	Read/Write, R/W=0: Write & R/W=1: Read

Theory (Cont.)

■ GLCD Internals and Pinout

6	EN	Enable. Falling edge triggered
7	D0	Data Bit 0
8	D1	Data Bit 1
9	D2	Data Bit 2
10	D3	Data Bit 3
11	D4	Data Bit 4
12	D5	Data Bit 5
13	D6	Data Bit 6
14	D7	Data Bit 7/Busy Flag
15	CS1	Chip Select for IC1/PAGE0

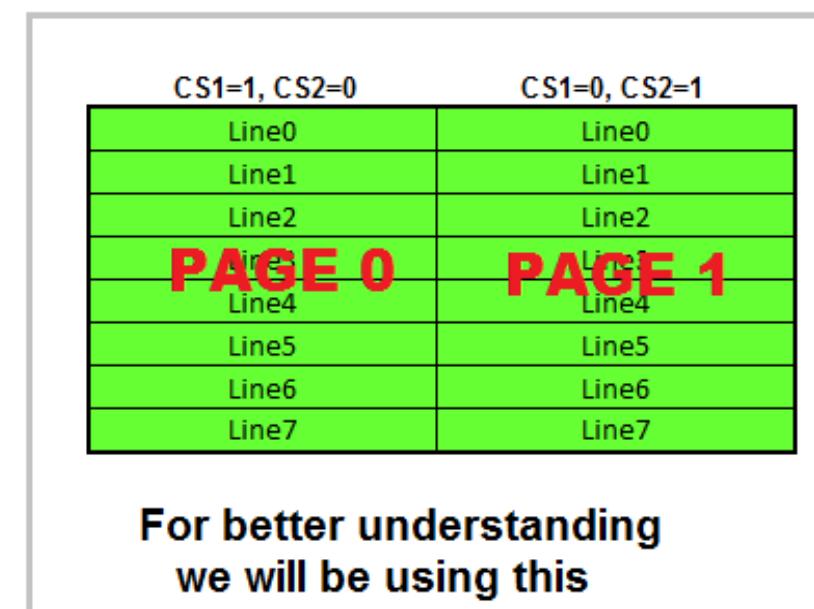
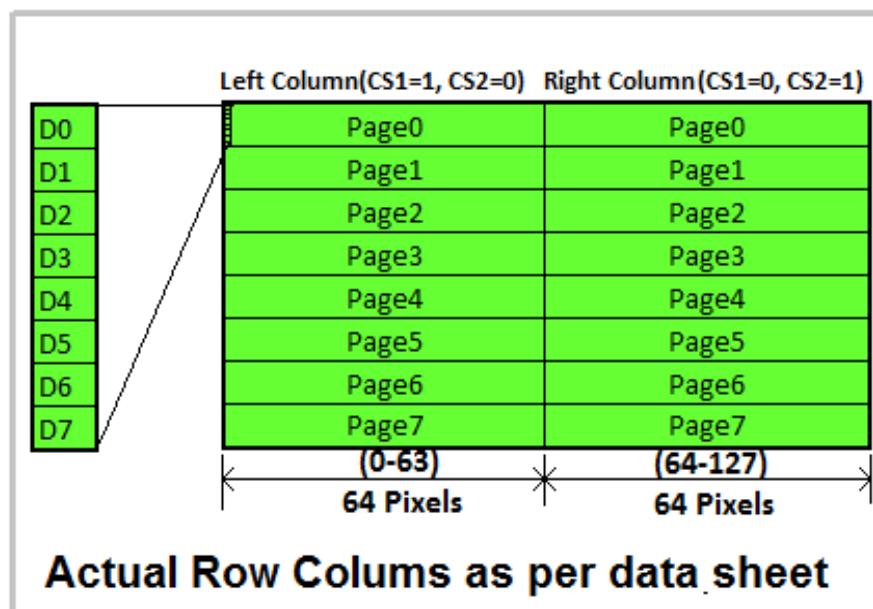
Theory (Cont.)

- GLCD Internals and Pinout

16	CS2	Chip Select for IC2/PAGE1
17	RST	Reset the LCD module
18	VEE	Negative voltage used along with Vcc for brightness control
15	A/LED+	Back-light Anode(+)
16	K/LED-	Back-Light Cathode(-)

Theory (Cont.)

- Page, Line and Cursor Selection



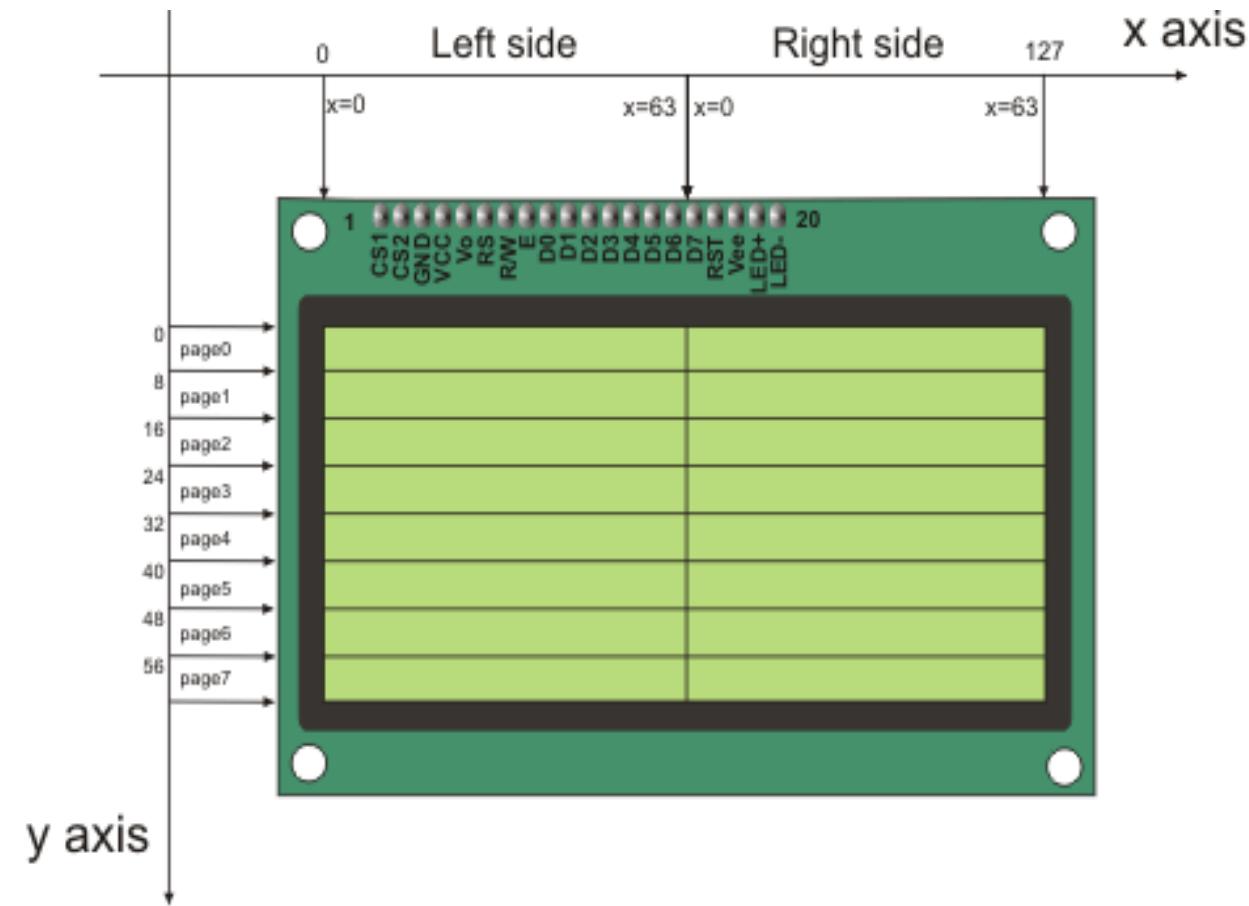
Theory (Cont.)

■ Page, Line and Cursor Selection

- Lets view the GLCD as a open book with **2pages** consisting of **8lines** on each page.
- Each line has 64 cursors positions to display data/images.
- The required page can be selected using **CS1,CS2** pins.
- **CS2-CS1:Chip Select Lines**
 - ✓ 00 = None
 - ✓ 01 = Page 0
 - ✓ 10 = Page 1
 - ✓ 11 = Both Pages

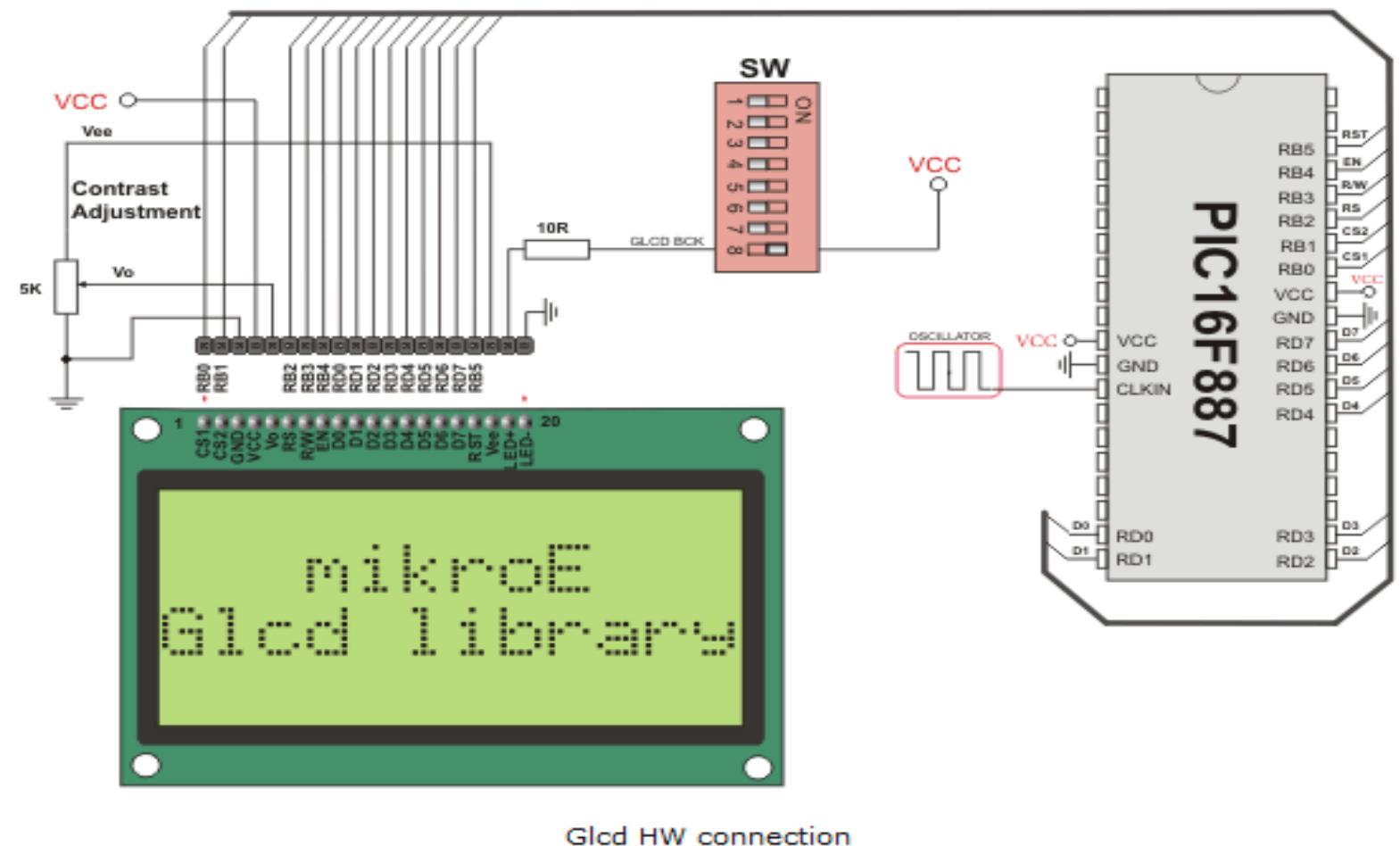
Theory (Cont.)

- HW Connection



Theory (Cont.)

- HW Connection (Cont.)

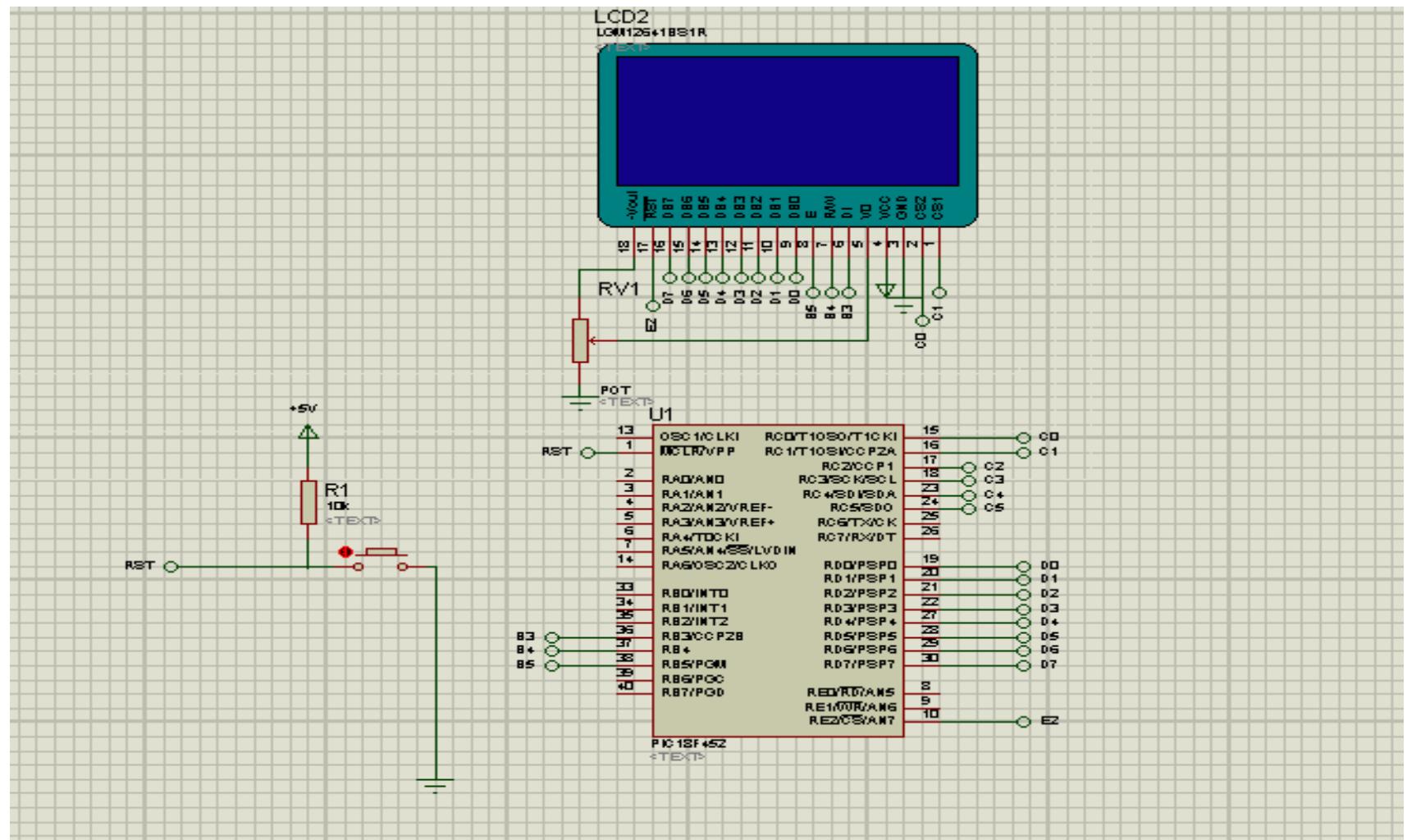


✿ The Experiment Components

- Microcontroller Pic 16F877A.
- LGM12641BS1R.
- Resistor 150 ohm.
- Push Button.

Procedure

Connect the circuit as shown in the figure.

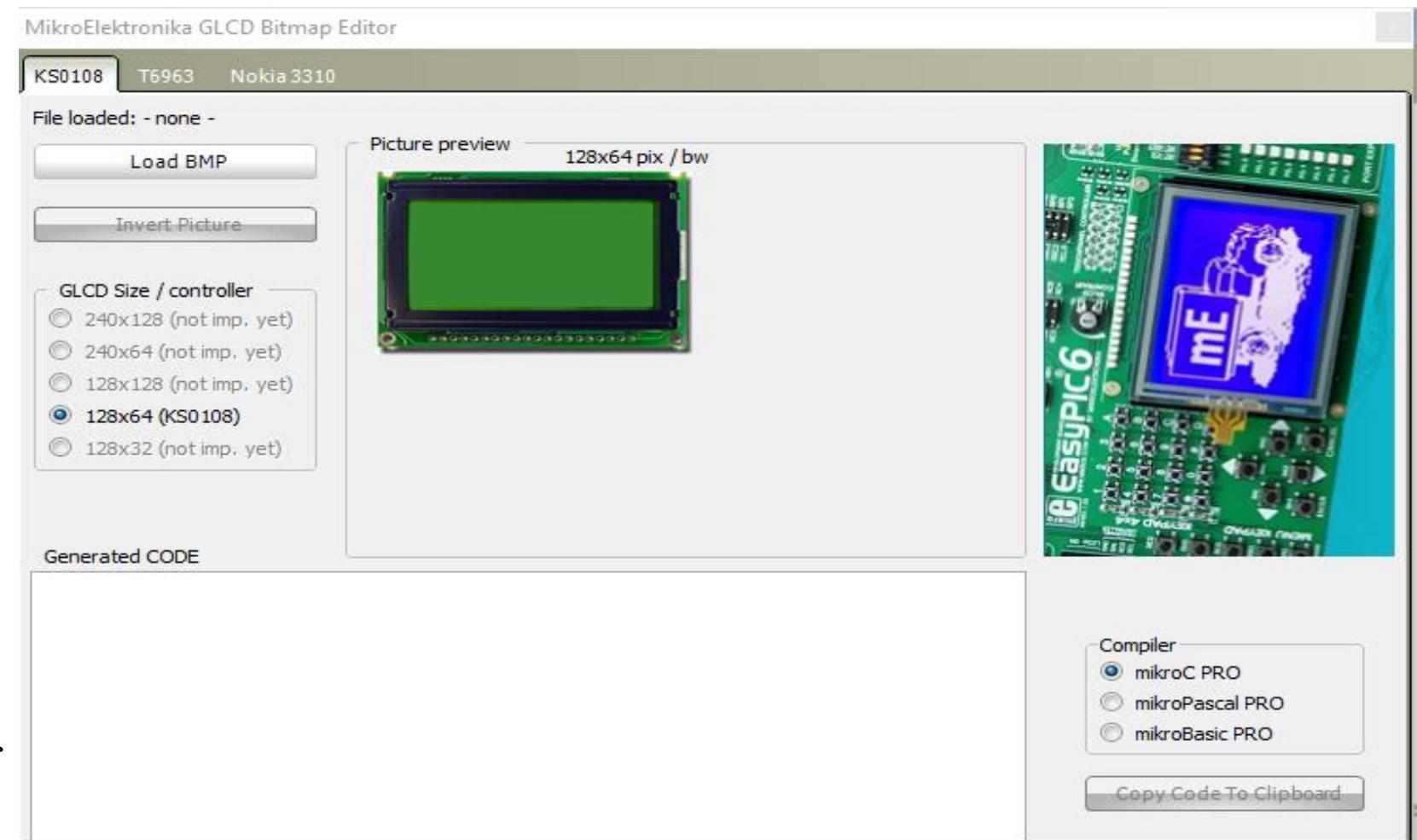


Procedure (Cont.)

From tools → GLCD
Bitmap Editor → the
window Appear as shown
in figure.

From this we can load the
graph from the program to
the GLCD. First select the
size of the GLCD and the
compiler after that load
BMP.

Note: The image must be of
an extension *.bitmap*.



❖ Procedure (Cont.)

Write the code in **MikroC** program to display the image.

```
// Glcd module connections
char GLCD_DataPort at PORTD;

sbit GLCD_CS1 at RC0_bit;
sbit GLCD_CS2 at RC1_bit;
sbit GLCD_RS at RB3_bit;
sbit GLCD_RW at RB4_bit;
sbit GLCD_EN at RB5_bit;
sbit GLCD_RST at RE2_bit;

sbit GLCD_CS1_Direction at TRISCO_bit;
sbit GLCD_CS2_Direction at TRISC1_bit;
sbit GLCD_RS_Direction at TRISB3_bit;
sbit GLCD_RW_Direction at TRISB4_bit;
sbit GLCD_EN_Direction at TRISB5_bit;
sbit GLCD_RST_Direction at TRISE2_bit;
// End Glcd module connections
```

Procedure (Cont.)

Write the code in **MikroC**
program to display the
image.

Procedure (Cont.)

Write the code in **MikroC** program to display the image.

Procedure (Cont.)

Write the code in MikroC program to display the image.

Procedure (Cont.)

Write the code in **MikroC** program to display the image.

❖ Procedure (Cont.)

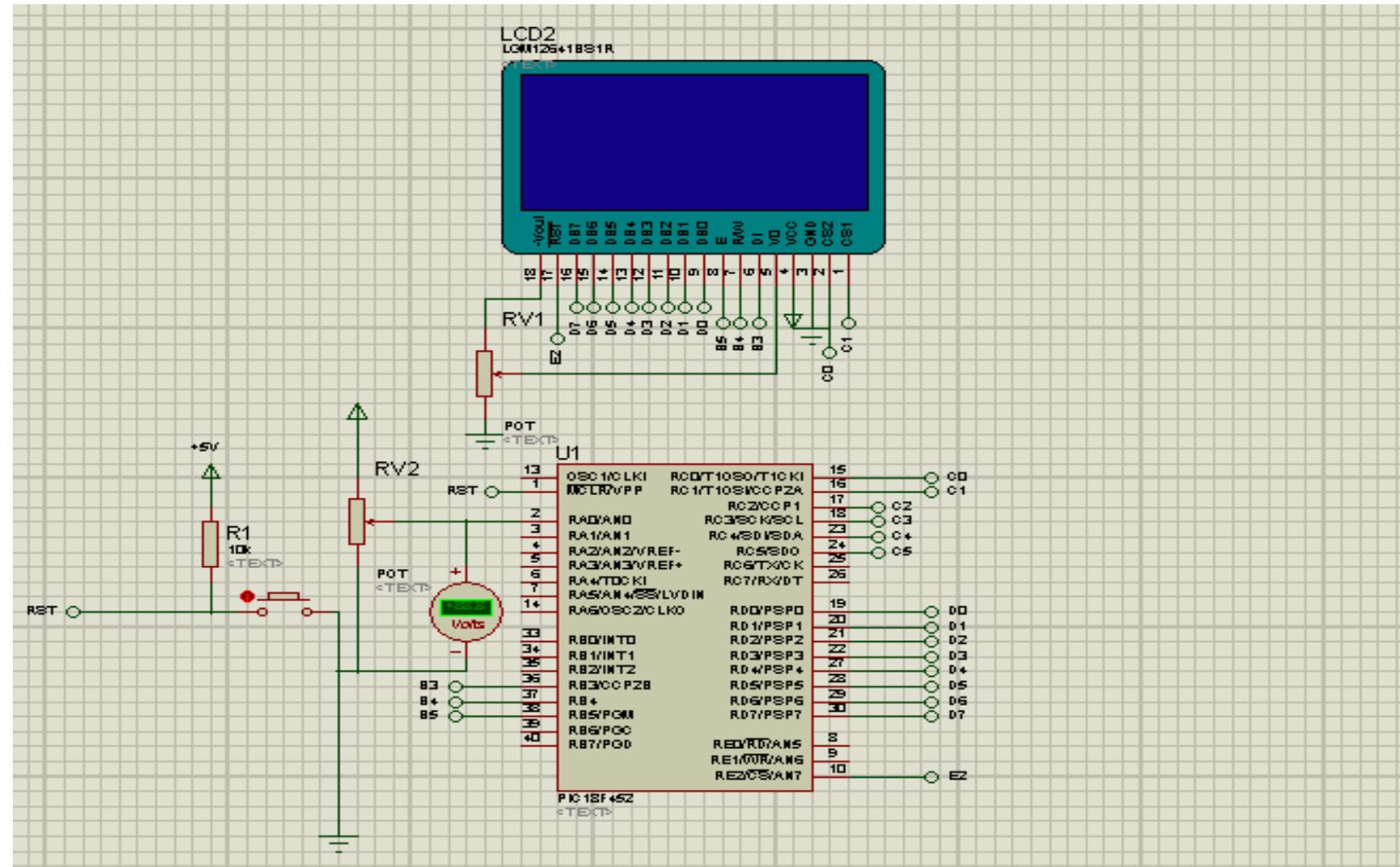
Write the code in **MikroC** program to display the image.

```
void main()
{
    Glcd_Init(); // Initialize GLCD
    Glcd_Fill(0); // Clear GLCD
    Glcd_Image(Allah);
}
```

Procedure

GLCD display the value of potentiometer:

Connect the circuit as shown in the figure.



❖ Procedure

GLCD display the value of potentiometer:

```
// Glcd module connections
char GLCD_DataPort at PORTD;

sbit GLCD_CS1 at RC0_bit;
sbit GLCD_CS2 at RC1_bit;
sbit GLCD_RS  at RB3_bit;
sbit GLCD_RW  at RB4_bit;
sbit GLCD_EN  at RB5_bit;
sbit GLCD_RST at RE2_bit;

sbit GLCD_CS1_Direction at TRISCO_bit;
sbit GLCD_CS2_Direction at TRISC1_bit;
sbit GLCD_RS_Direction  at TRISB3_bit;
sbit GLCD_RW_Direction  at TRISB4_bit;
sbit GLCD_EN_Direction  at TRISB5_bit;
sbit GLCD_RST_Direction at TRISE2_bit;
// End Glcd module connections
```

Procedure

GLCD display the value of potentiometer:

```
unsigned int adc_value;
char adc_value_txt[10];

float volt;
char volt_txt[15];

void _init()
{
    ADC_Init(); // Initialize ADC module with default settings
    ADCON1 = 0b000001110;

    TRISA.B0 = 1; // PORT A0 as input

    Glcd_Init(); // Initialize GLCD
    Glcd_Fill(0x00); // Clear GLCD

    Glcd_Write_Text("<<< ADC & GLCD >>>", 5, 0, 1);
}

void get_adc()
{
    adc_value = ADC_Read(0); //read analog from AN0
```

Procedure

GLCD display the value of potentiometer:

```
//calculate voltage
volt = adc_value * 5;
volt = volt / 1024;

wordtostr(adc_value,adc_value_txt); //convert adc value to string
floattostr(volt,volt_txt); //convert voltage to string

//display adc value
Glcd_Write_Text("ADC Value: ", 0, 2, 1);
Glcd_Write_Text(adc_value_txt, 65, 2, 1);

//display voltage
Glcd_Write_Text("Voltage: ", 0, 3, 1);
Glcd_Write_Text(volt_txt, 65, 3, 1);

Delay_ms(500);
}

void progress_bar()
{
    if(volt > 1){Glcd_Box(10, 39, 20, 50, 1);}
    if(volt < 1){Glcd_Box(10, 39, 20, 50, 0);}
}
```

Procedure

GLCD display the value of potentiometer:

```
if(volt > 1.5){Glcd_Box(20, 39, 30, 50, 1);}
if(volt < 1.5){Glcd_Box(20, 39, 30, 50, 0);}

if(volt > 2){Glcd_Box(30, 39, 40, 50, 1);}
if(volt < 2){Glcd_Box(30, 39, 40, 50, 0);

if(volt > 2.5){Glcd_Box(40, 39, 50, 50, 1);}
if(volt < 2.5){Glcd_Box(40, 39, 50, 50, 0);

if(volt > 3){Glcd_Box(50, 39, 60, 50, 1);}
if(volt < 3){Glcd_Box(50, 39, 60, 50, 0);

if(volt > 3.5){Glcd_Box(60, 39, 70, 50, 1);}
if(volt < 3.5){Glcd_Box(60, 39, 70, 50, 0);

if(volt > 4){Glcd_Box(70, 39, 80, 50, 1);}
if(volt < 4){Glcd_Box(70, 39, 80, 50, 0);

if(volt > 4.5){Glcd_Box(80, 39, 90, 50, 1); Glcd_Write_Text(",
", 0, 7, 1);Glcd_Write_Text("High Voltage ! ", 0, 7, 1);}
if(volt < 4.5){Glcd_Box(80, 39, 90, 50, 0); Glcd_Write_Text(",
", 0, 7, 1);}
}
```

❖ Procedure

GLCD display the value
of potentiometer:

```
void main()
{
    _init();

    do
    {
        get_adc();
        progress_bar();

    }while(1);

}
```

Discussion

1- Write and design the program to display on the GLCD by using protues and mikroC. Display on the GLCD the following graphics:

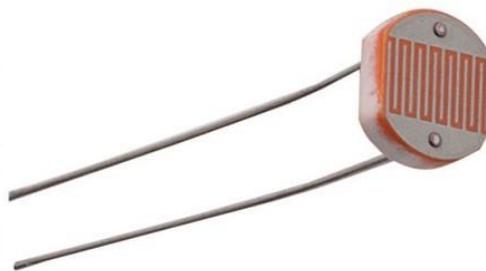
- a. V_line.
- b. H_line.
- c. Rectangle.
- d. Rectangle round edge.
- e. Circle.

These graphs changes on the GLCD every 1s.

2- Write and design the program to display on the GLCD your picture by using protues and mikroC.

EXPERIMENT 08

LDR Sensor



Experiment Object

The object of this experiment is to learn how to use LDR resistor with microcontroller.

Theory

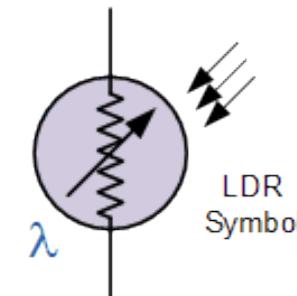
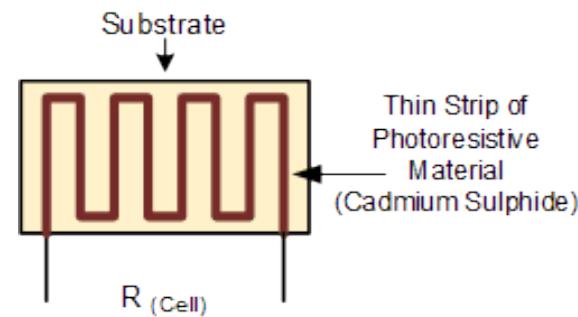
LDR Sensor



❖ Theory

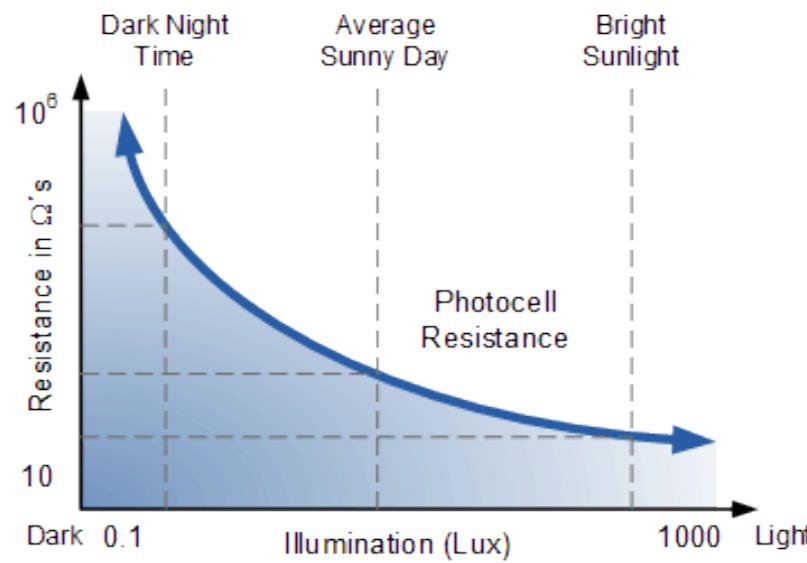
■ LDR Sensor (Cont.)

The **LDR** sensor is the abbreviation of the **Light Dependent Resistor** and as the name implies the resistance of this sensor varies in relation to the intensity of light, greater the intensity of light lower will be resistance of the LDR sensor. The LDR sensors are based on the semiconductor technology which means that they are composed of the Silicon. As it is commonly known fact that resistance of the semiconductors varies due to change in the intensity of light so the Light dependent resistor exploit this property of the semiconductors.



Theory

LDR Sensor (Cont.)

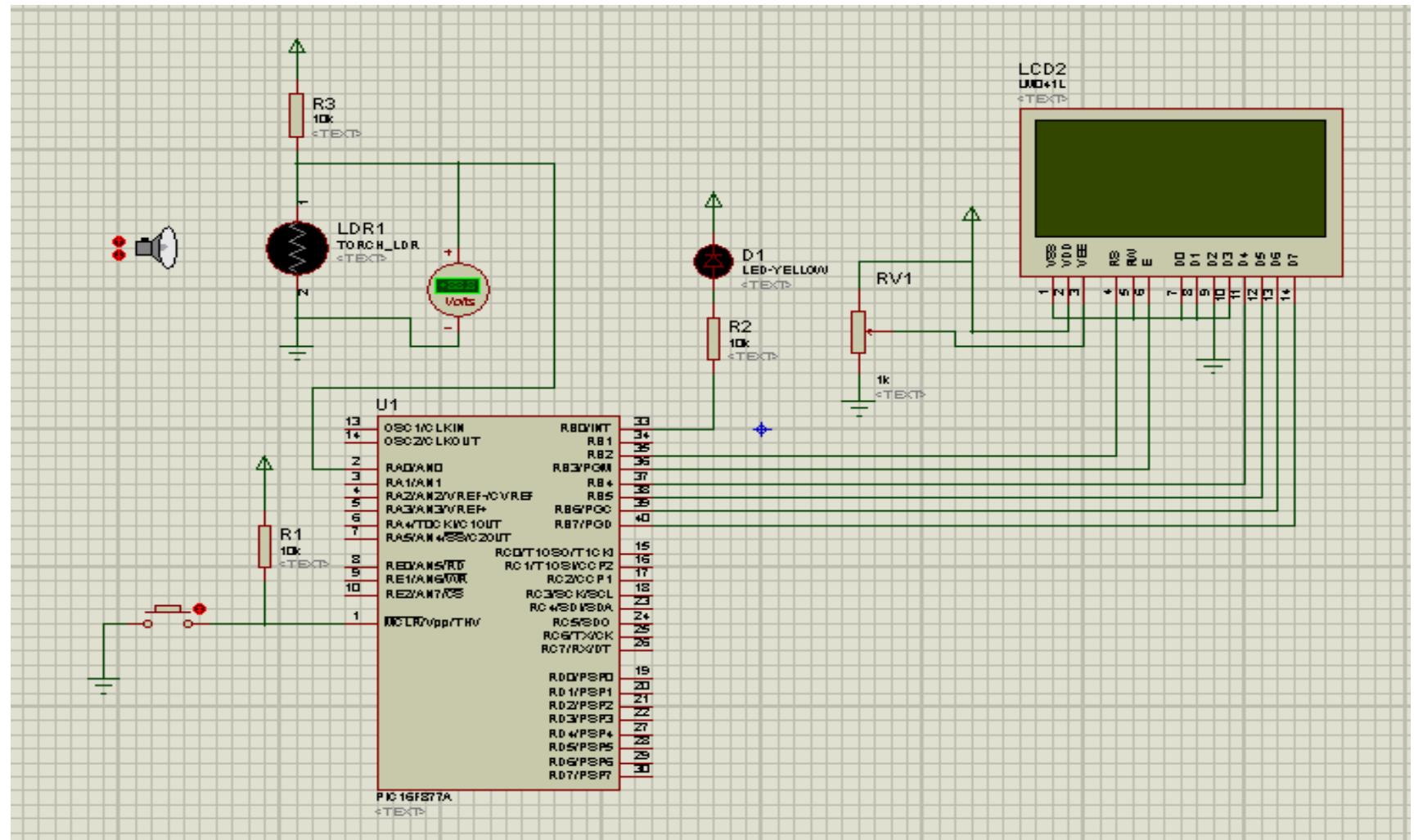


✿ The Experiment Components

- Microcontroller Pic 16F877A.
- LCD 16x4.
- Resistor 150 ohm.
- Push Button.
- Potentiometer.
- LDR resistor.

Procedure

Connect the circuit as shown in the figure.



Procedure

Write the code in MikroC program to display the image.

```
// LCD module connections
sbit LCD_RS at RB2_bit;
sbit LCD_EN at RB3_bit;
sbit LCD_D4 at RB4_bit;
sbit LCD_D5 at RB5_bit;
sbit LCD_D6 at RB6_bit;
sbit LCD_D7 at RB7_bit;

sbit LCD_RS_Direction at TRISB2_bit;
sbit LCD_EN_Direction at TRISB3_bit;
sbit LCD_D4_Direction at TRISB4_bit;
sbit LCD_D5_Direction at TRISB5_bit;
sbit LCD_D6_Direction at TRISB6_bit;
sbit LCD_D7_Direction at TRISB7_bit;
// End LCD module connections

unsigned int adc_value;
char adc_value_txt[10];

float volt;
char volt_text[15];
```

Procedure

Write the code in MikroC program to display the image.

```
void main()
{
    ADC_Init(); // Initialize ADC module with default settings
    ADCON1 = 0b00001110;

    TRISA.B0 = 1; // PORT A0 as input
    TRISB.B0 = 0;
    PORTB.B0 = 0;

    Lcd_Init(); // Initialize LCD
    Lcd_Cmd(_LCD_CLEAR); // CLEAR display
    Lcd_Cmd(_LCD_CURSOR_OFF); // Cursor off

    do
    {
        adc_value = ADC_Read(0); //read analog from A0

        //calculate temperature
        volt = adc_value * 5;

        volt = volt / 1024;

        wordtostr(adc_value,adc_value_txt); //convert adc value to string
        floattostr(volt,volt_text); //convert adc value to string
```

Procedure

Write the code in MikroC program to display the image.

```
Lcd_Out(1,1,adc_value_txt); //display adc value
Lcd_Out(2,1,volt_text);   Lcd_chr_cp('V'); //display voltage

if(volt < 2) //if morning turn off light & display message
{
    portb.b0 = 0;
    lcd_out(4,1, "          ");
    lcd_out(4,1, "Morning");
}

else //if night turn on light & display message
{
    portb.b0 = 1;
    lcd_out(4,1, "          ");
    lcd_out(4,1, "Night");
}

Delay_ms(1000);

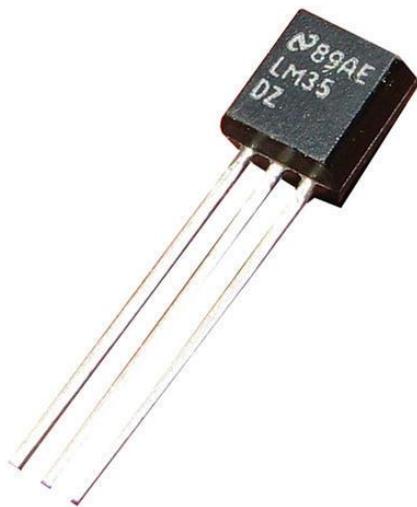
}while(1);
}
```

❖ Discussion

- 1- Write and design the program to display the value of LDR sensor on 7 segments.

EXPERIMENT 09

Temperature Sensor LM35

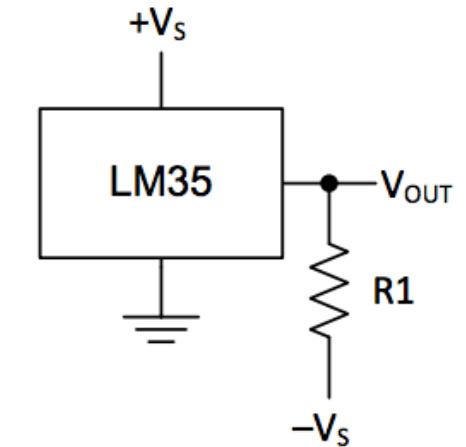
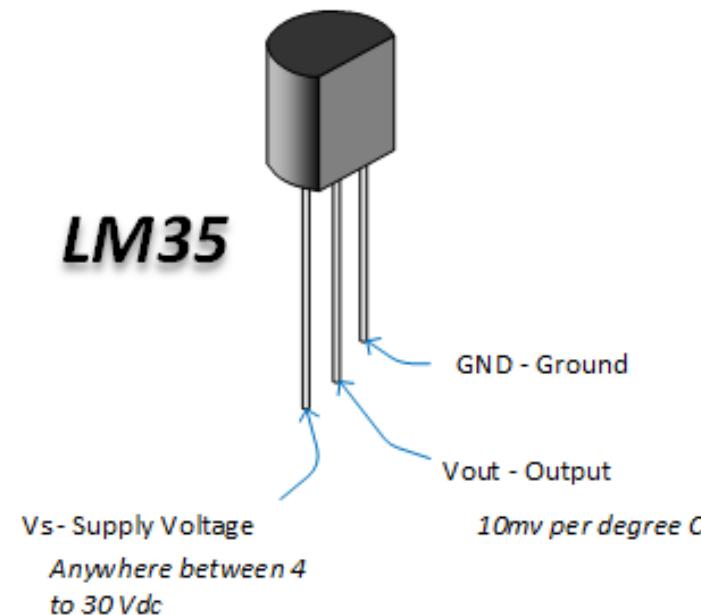


Experiment Object

The object of this experiment is to learn how to use LM35 temperature sensor with microcontroller.

Theory

■ LM35 Temperature Sensor



Choose $R_1 = -V_S / 50 \mu A$
 $V_{OUT} = 1500 \text{ mV at } 150^\circ\text{C}$
 $V_{OUT} = 250 \text{ mV at } 25^\circ\text{C}$
 $V_{OUT} = -550 \text{ mV at } -55^\circ\text{C}$

Figure 2. Full-Range Centigrade Temperature Sensor

✿ Theory

■ LM35 Temperature Sensor

LM35 is a commonly used temperature sensor, It shows values in the form of output voltages instead of degree Celsius.

LM35 shows high voltage values than thermocouples and may not need that the output voltage is amplified.

The output voltage of LM35 is proportional to the Celsius temperature. The scale factor is $.01 \text{ V}^{\circ}\text{C}$.

One most important characteristic is that it draws just 60 microamps from its supply and acquires a low self-heating capacity.

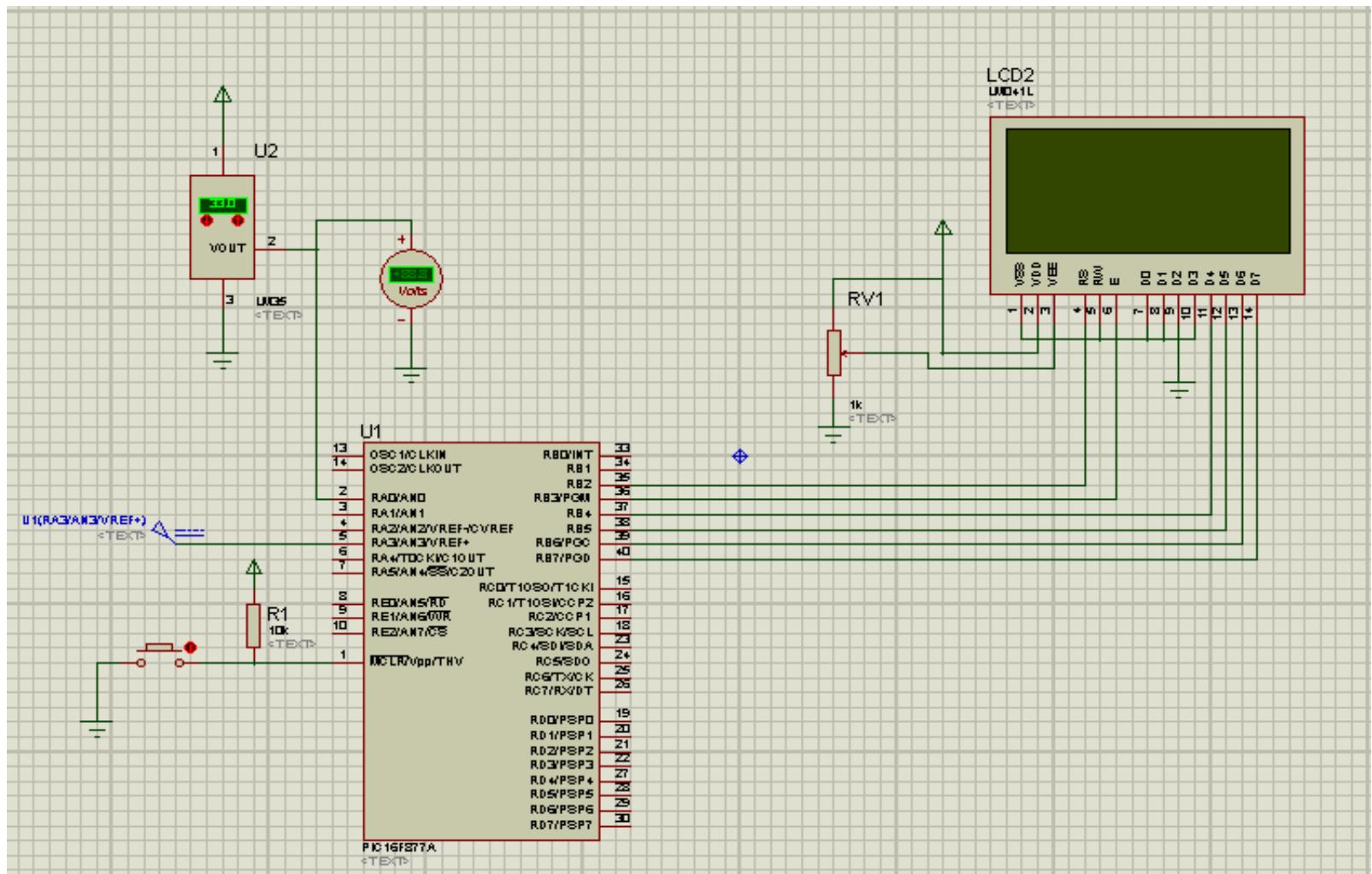
LM35 temperature sensor available in many different packages like T0-46 metal transistor-like package, TO-92 plastic transistor-like package, 8-lead surface mount SO-8 small outline package.

✿ The Experiment Components

- Microcontroller Pic 16F877A.
- LCD 16x4.
- Resistor 150 ohm.
- Push Button.
- Potentiometer.
- LM35 Temperature sensor.

Procedure

Connect the circuit as shown in the figure.



Procedure

Write the code in MikroC program to display the image.

```
// LCD module connections
sbit LCD_RS at RB2_bit;
sbit LCD_EN at RB3_bit;
sbit LCD_D4 at RB4_bit;
sbit LCD_D5 at RB5_bit;
sbit LCD_D6 at RB6_bit;
sbit LCD_D7 at RB7_bit;

sbit LCD_RS_Direction at TRISB2_bit;
sbit LCD_EN_Direction at TRISB3_bit;
sbit LCD_D4_Direction at TRISB4_bit;
sbit LCD_D5_Direction at TRISB5_bit;
sbit LCD_D6_Direction at TRISB6_bit;
sbit LCD_D7_Direction at TRISB7_bit;
// End LCD module connections

unsigned int adc_value;
char adc_value_txt[10];

float TempC;
char TempC_text[15];
```

Procedure

Write the code in MikroC program to display the image.

```
void main()
{
    ADC_Init(); // Initialize ADC module with default settings

    ADCON1 = 0b00000101;

    TRISA.B0 = 1; // PORT A0 as input

    Lcd_Init(); // Initialize LCD
    Lcd_Cmd(_LCD_CLEAR); // CLEAR display
    Lcd_Cmd(_LCD_CURSOR_OFF); // Cursor off

    do
    {
        adc_value = ADC_Read(0); //read analog from A0

        //calculate temperature
        TempC = adc_value * 1.5;

        TempC = TempC / 1024;

        TempC = TempC * 100;

        wordtostr(adc_value,adc_value_txt); //convert adc value to string
    }
}
```

Procedure

Write the code in MikroC program to display the image.

```
    floattosstr(TempC,TempC_text); //convert temerature to string  
  
    Lcd_Out(1,1,adc_value_txt); //display adc value  
    Lcd_Out(2,1,TempC_text);   Lcd_chr_cp('C');//display temperature  
  
    Delay_ms(1000);  
}while(1);  
}
```

Discussion

- 1- Write and design the program to LED light control by using LM35 sensor. If the temperature is greater than a certain value the led (on) else the led (off).

EXPERIMENT 10

Relay Switch

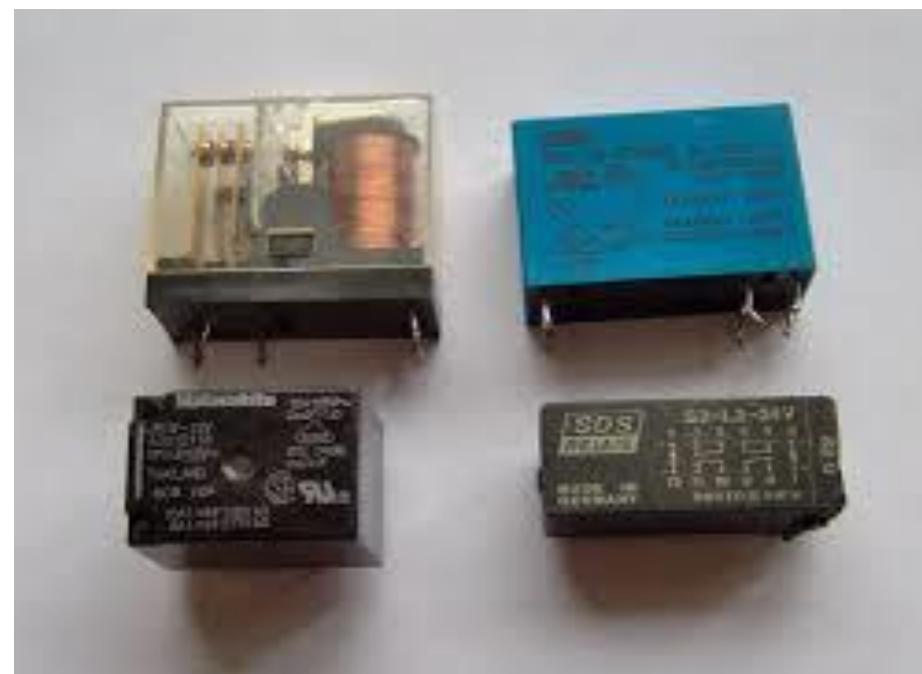


Experiment Object

The object of this experiment is to learn how to use relay switch with microcontroller.

Theory

■ Relay Switch



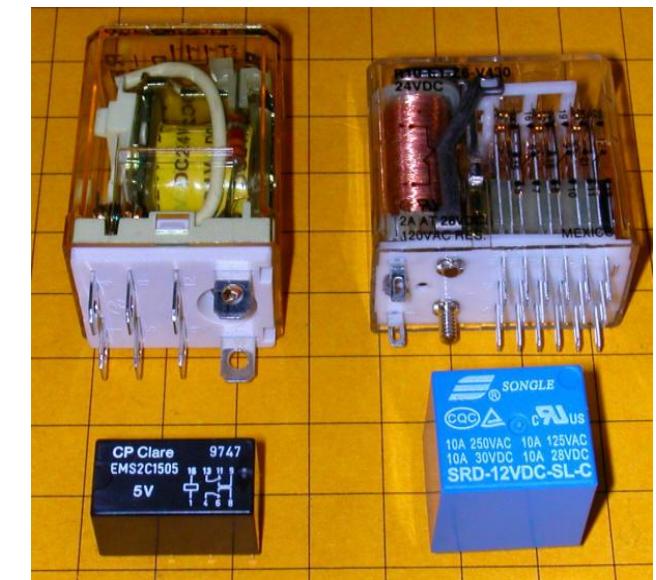
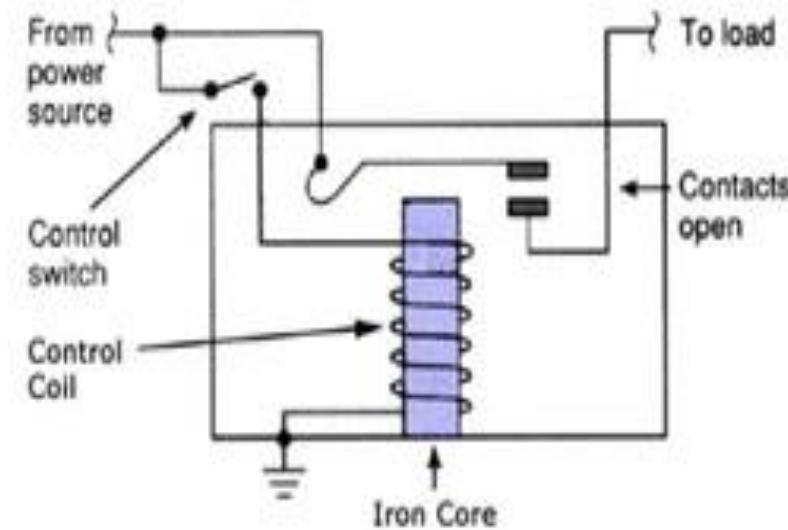
❖ Theory

■ Relay Switch

Relays are electromagnetic switches that are operated by a rather small electrical current. The current is able to turn off or on another, larger electrical current when signaled. A relay is sort of like a lever that operates using electricity. When it is switched on using a small current, then it in turn can switch on other appliances or applications by using a stronger current. These are very useful electronic parts. There are many types of sensors that are used in various types of electrical equipment but they are only capable of producing very small currents. Sometimes these smaller currents are necessary in order to operate a bigger apparatus that needs larger currents. Relays are used to fill in the gap so that a small current can be used to activate a larger one. What this means is that relays can be used as a switch which turns items off or on; or as an amplifier which converts a small current into a larger one.

Theory

■ Relay Switch

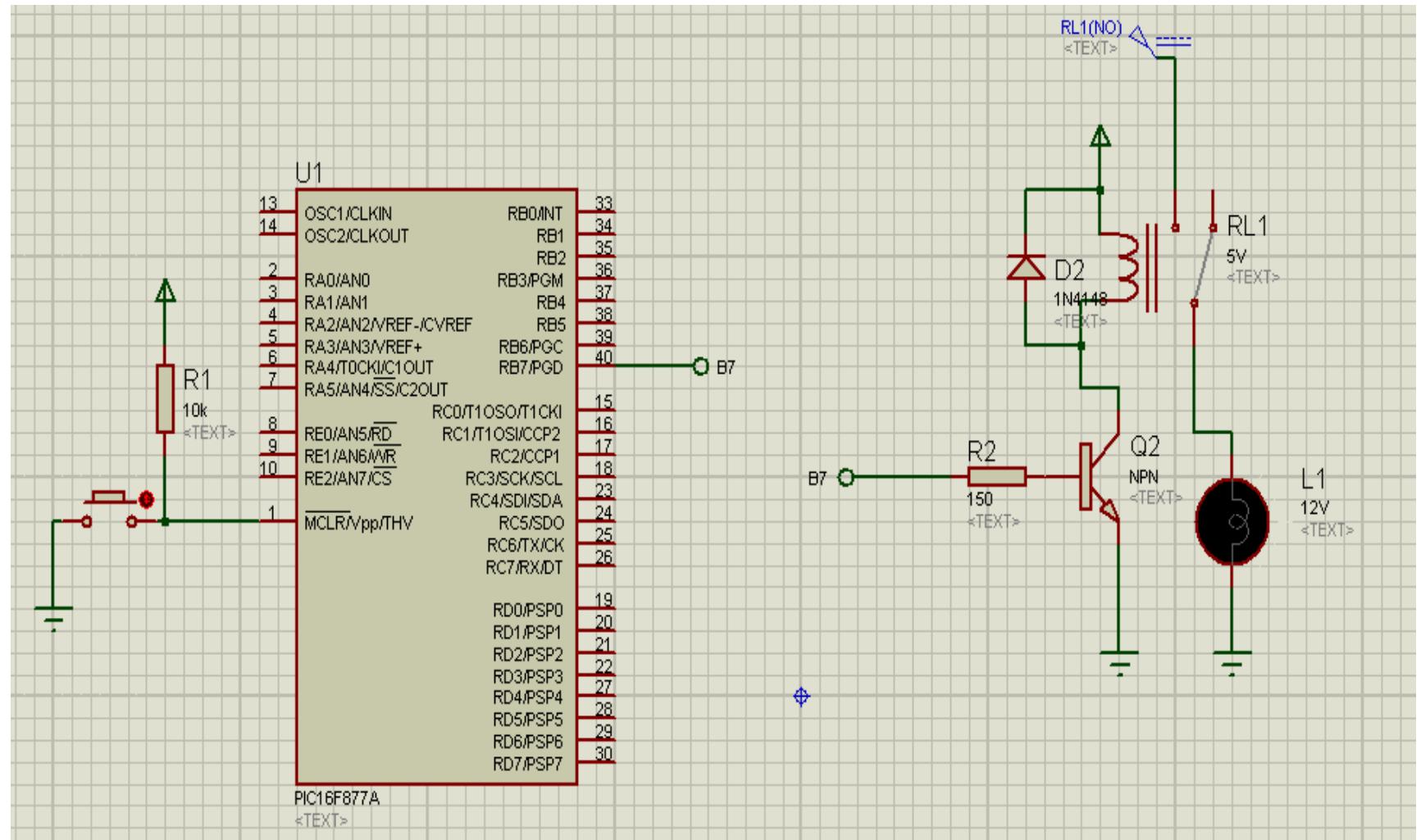


✿ The Experiment Components

- Microcontroller Pic 16F877A.
- Resistor 150 ohm.
- Push Button.
- Transistor 2N2222 NPN.
- Relay.
- Diode.

Procedure

Connect the circuit as shown in the figure.



❖ Procedure

Write the code in MikroC program to display the image.

```
//main program-----  
void main()  
{  
  
    TRISb.b7 = 0; PORTb.b7 = 0;//light as output  
  
    portb.b7 = 1;  
    delay_ms(4000);  
    portb.b7 = 0;  
  
}  
//end main program-----
```

❖ Discussion

- 1- What are the types of relay and what are the benefits of relay?

EXPERIMENT 11

DC Motor



❖ Experiment Object

The object of this experiment is to learn how to use dc motor with microcontroller.

❖ Theory

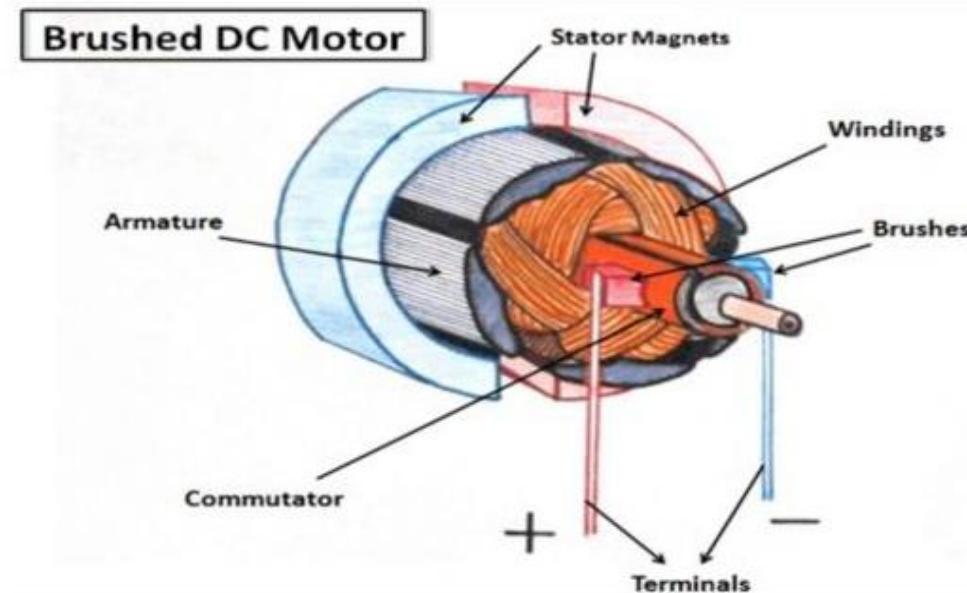
■ What is a DC Motor?



✿ Theory

■ What is a DC Motor?

A motor that converts electrical energy to mechanical energy. When the motor is connected to a source, the motor continuously rotates against the step motor.



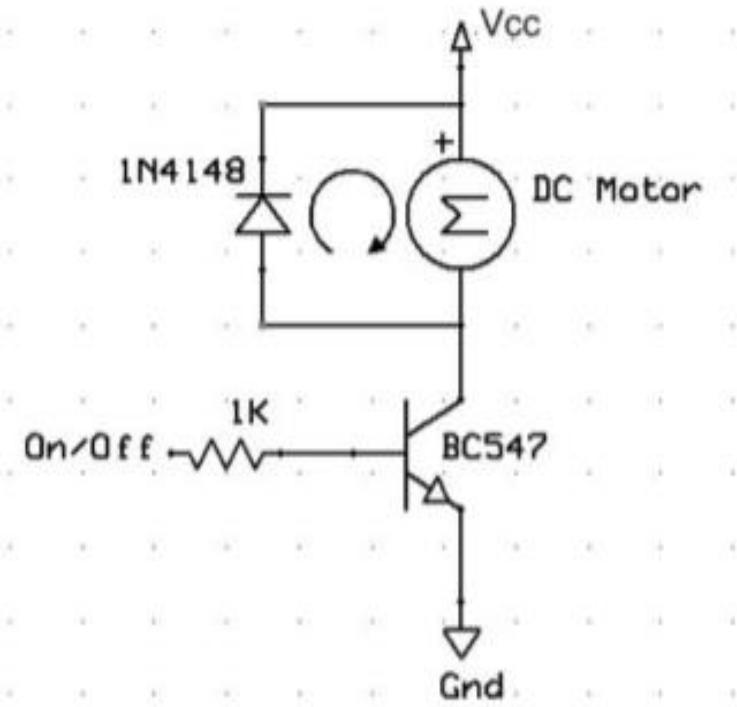
✿ Theory

■ DC Motor Control

A motor that converts electrical energy to mechanical energy. When the motor is connected to a source, the motor continuously rotates against the step motor.

When the engine is plugged in, the engine will rotate in one direction.

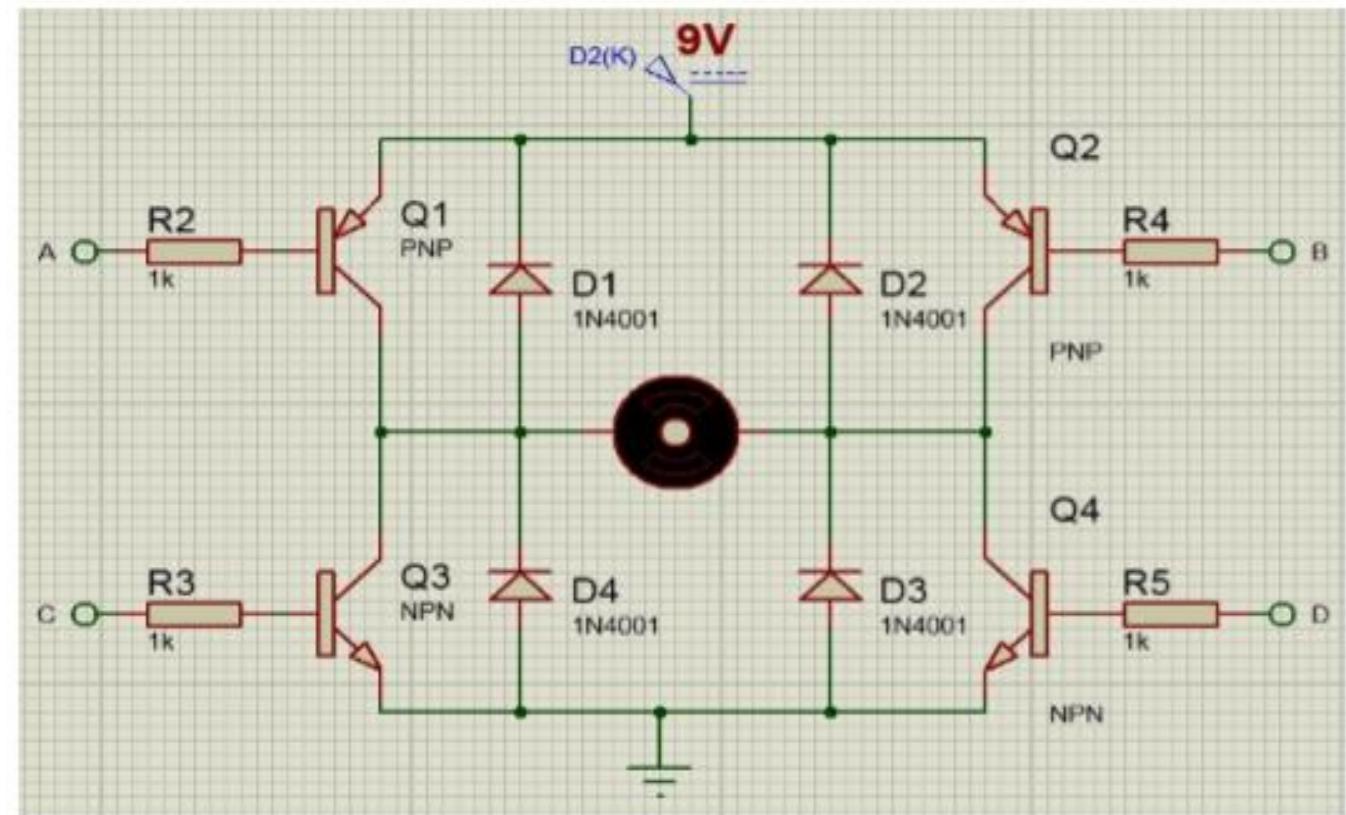
But when you need to move the engine to rotate in two directions you can use H-Bridge:



❖ Theory

■ DC Motor Control (Cont.)

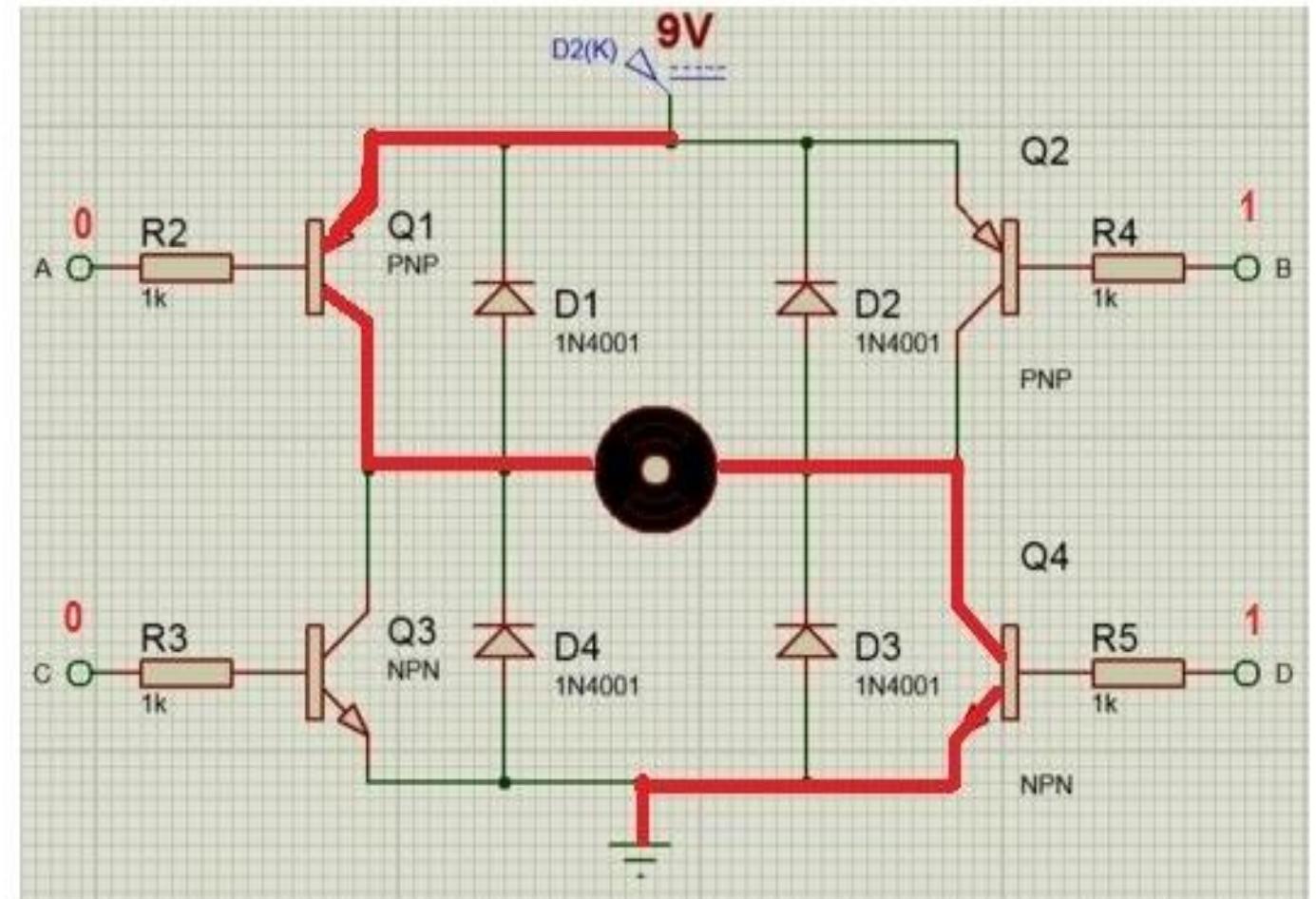
As we can see in the h-Bridge circuit we have four transistors to control the engine. When we want to move the engine clockwise, the (PNP)A transistor is activated, the transistor is activated (NPN)D, and the motor moves counter clockwise, the transistor (PNP)B is activated, and the transistor is activated (NPN)C.



✿ Theory

■ DC Motor Control (Cont.)

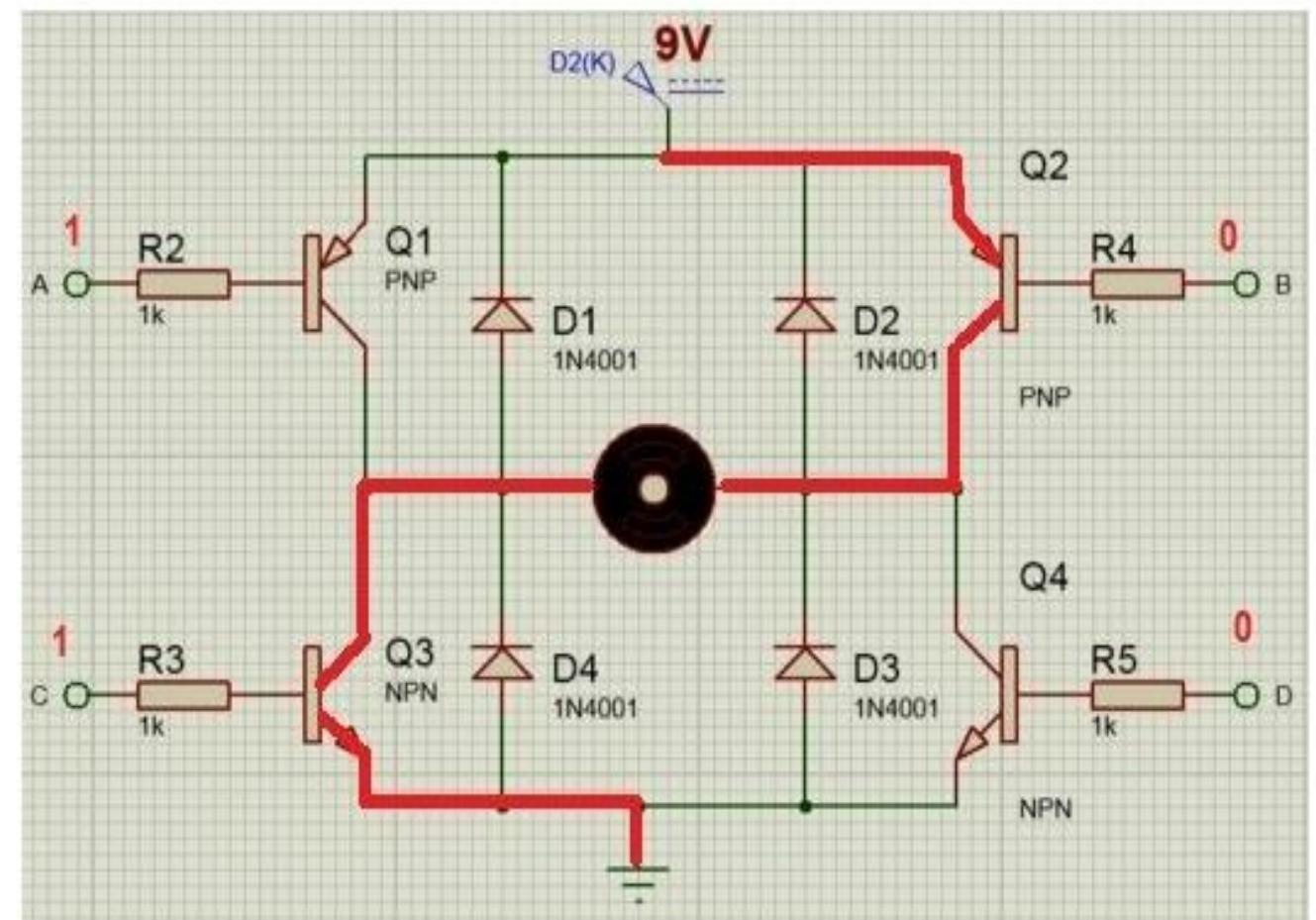
Note the current path in the case of the engine clockwise:



Theory

■ DC Motor Control (Cont.)

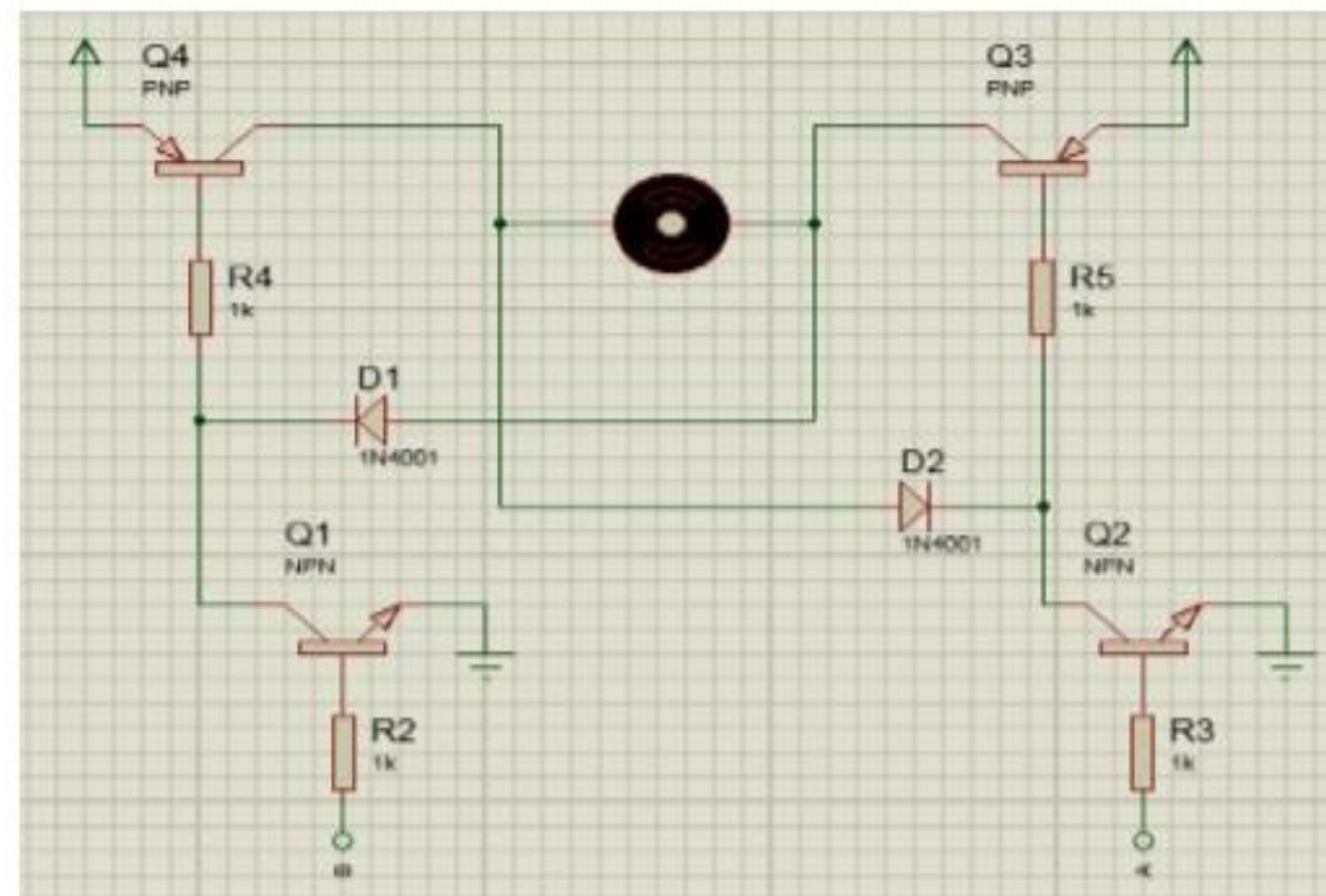
Note the current path in the case of the engine counter clockwise:



✿ Theory

■ DC Motor Control (Cont.)

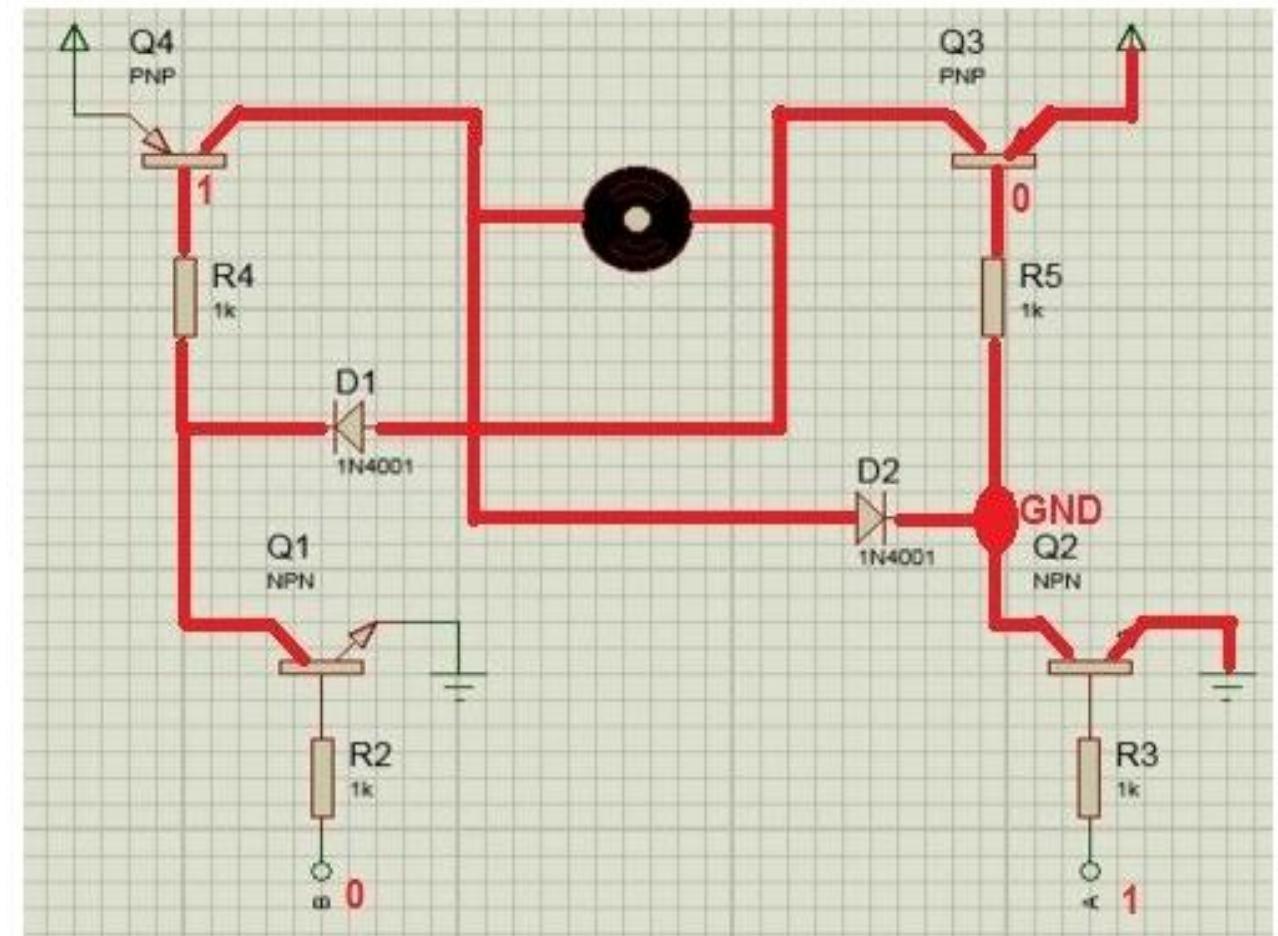
As seen in the previous circuit H-Bridge to move the engine is controlled by four transistors, which is complicated in control. For this reason, the circuit can be modified and the transistor is used only for control.



✿ Theory

■ DC Motor Control (Cont.)

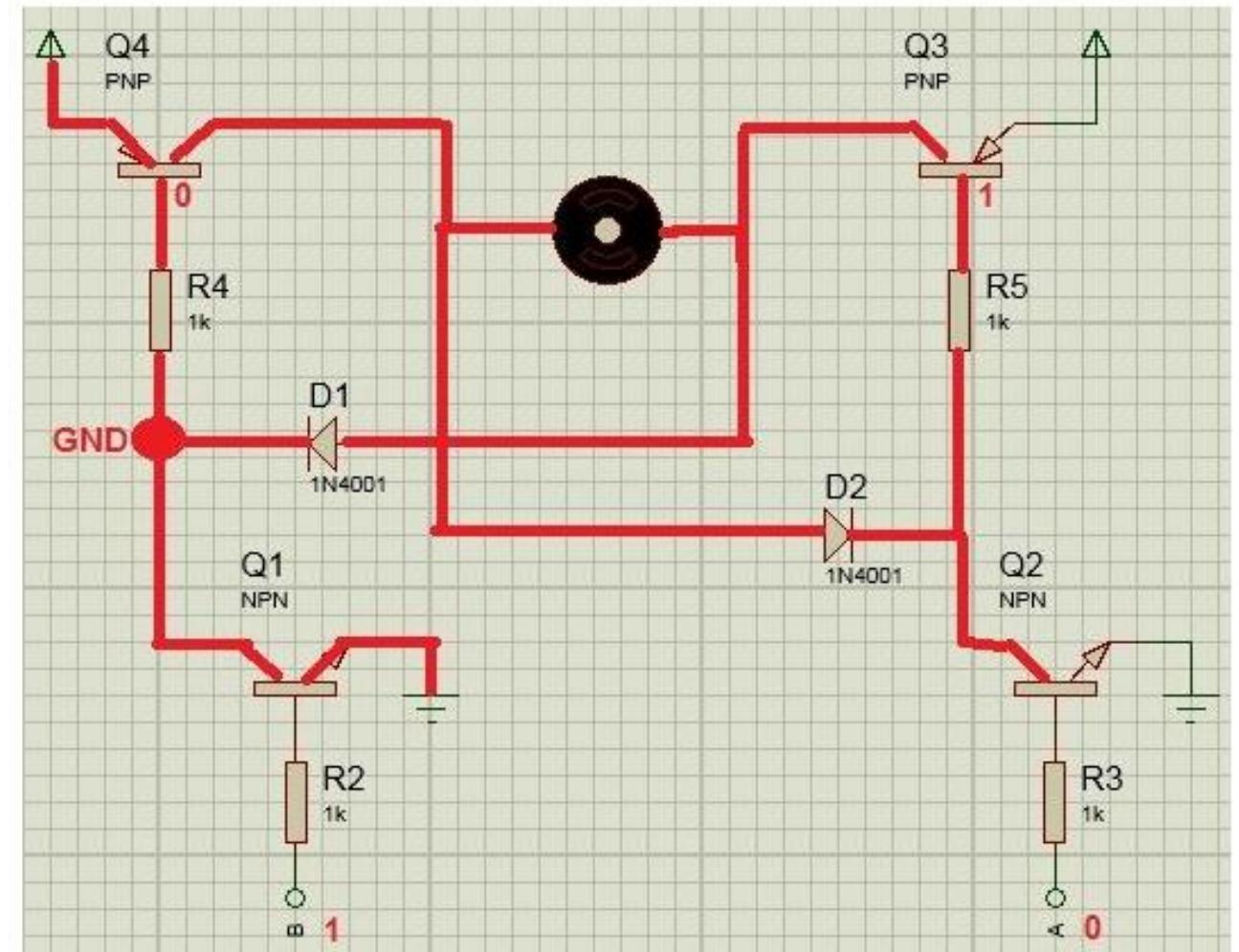
Note the current path in the case of the engine clockwise:



Theory

■ DC Motor Control (Cont.)

Note the current path in the case of the engine counter clockwise:

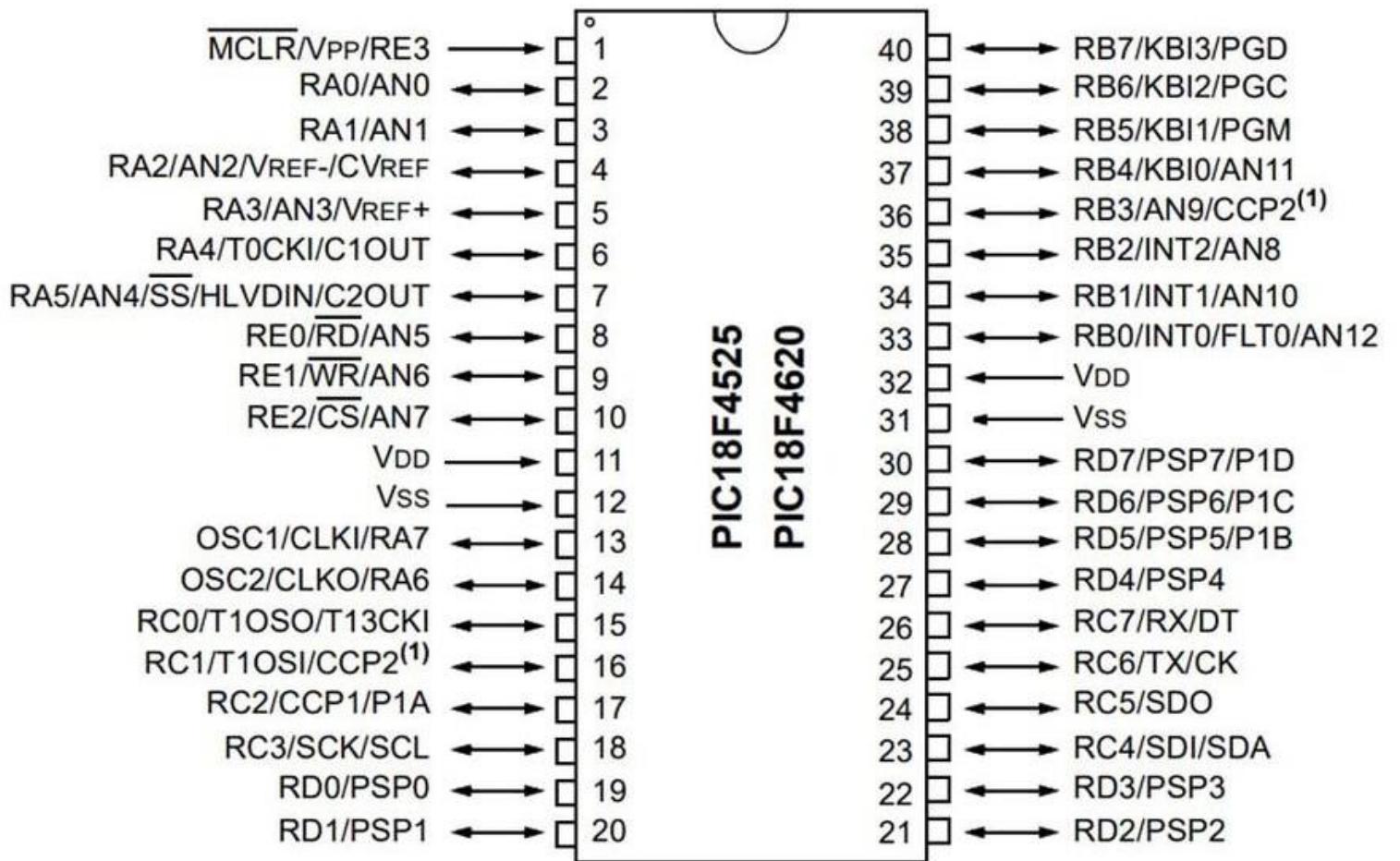


✿ The Experiment Components

- Microcontroller Pic 18F4525.
- Resistor 150 ohm.
- Push Button.
- Crystal 8MHZ.
- Capacitor 10 μ F.
- Transistor 2N2222.
- DC motor.
- Diode.

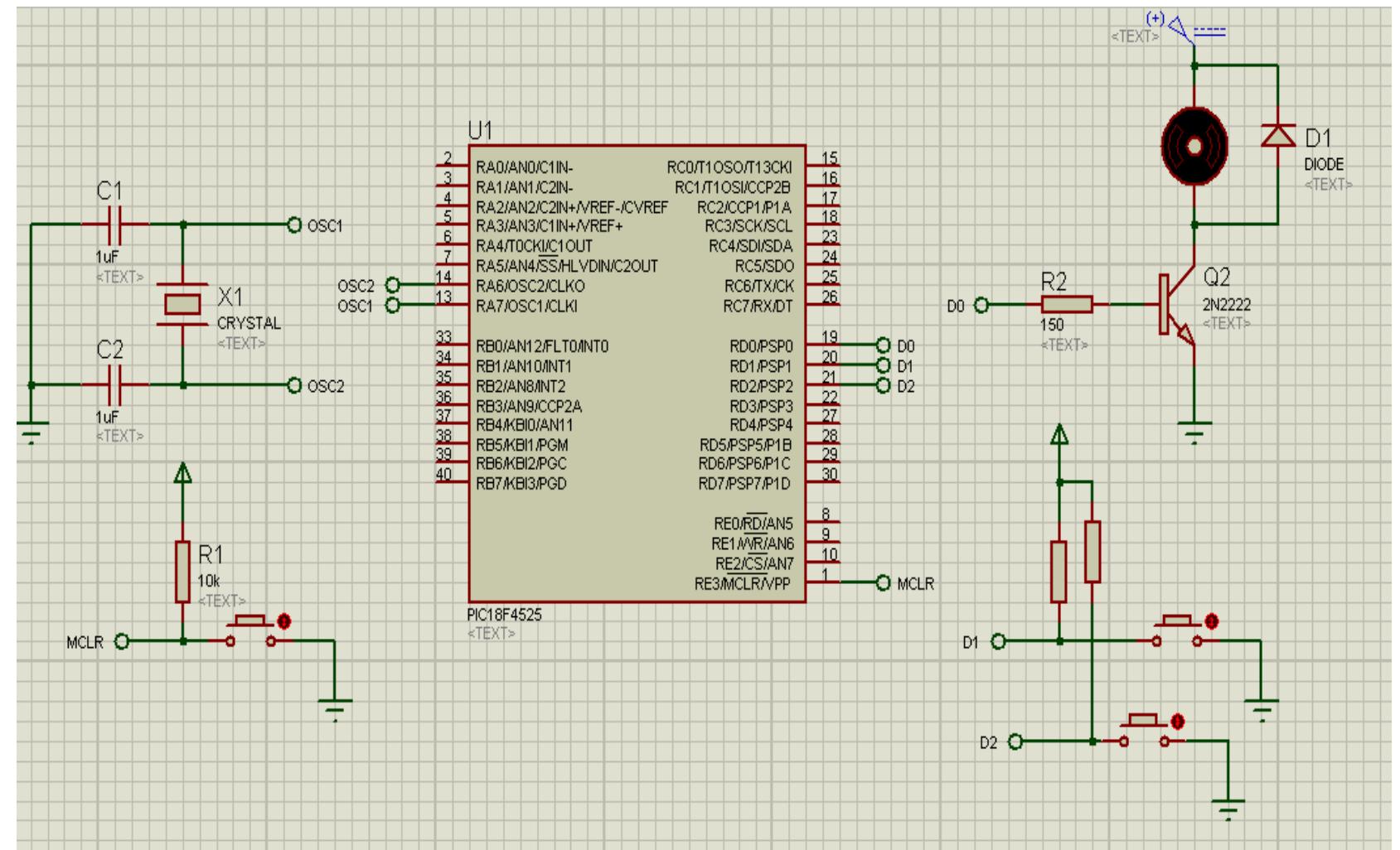
The Experiment Components

- In this experiment we use the Microcontroller Pic 18F4525.



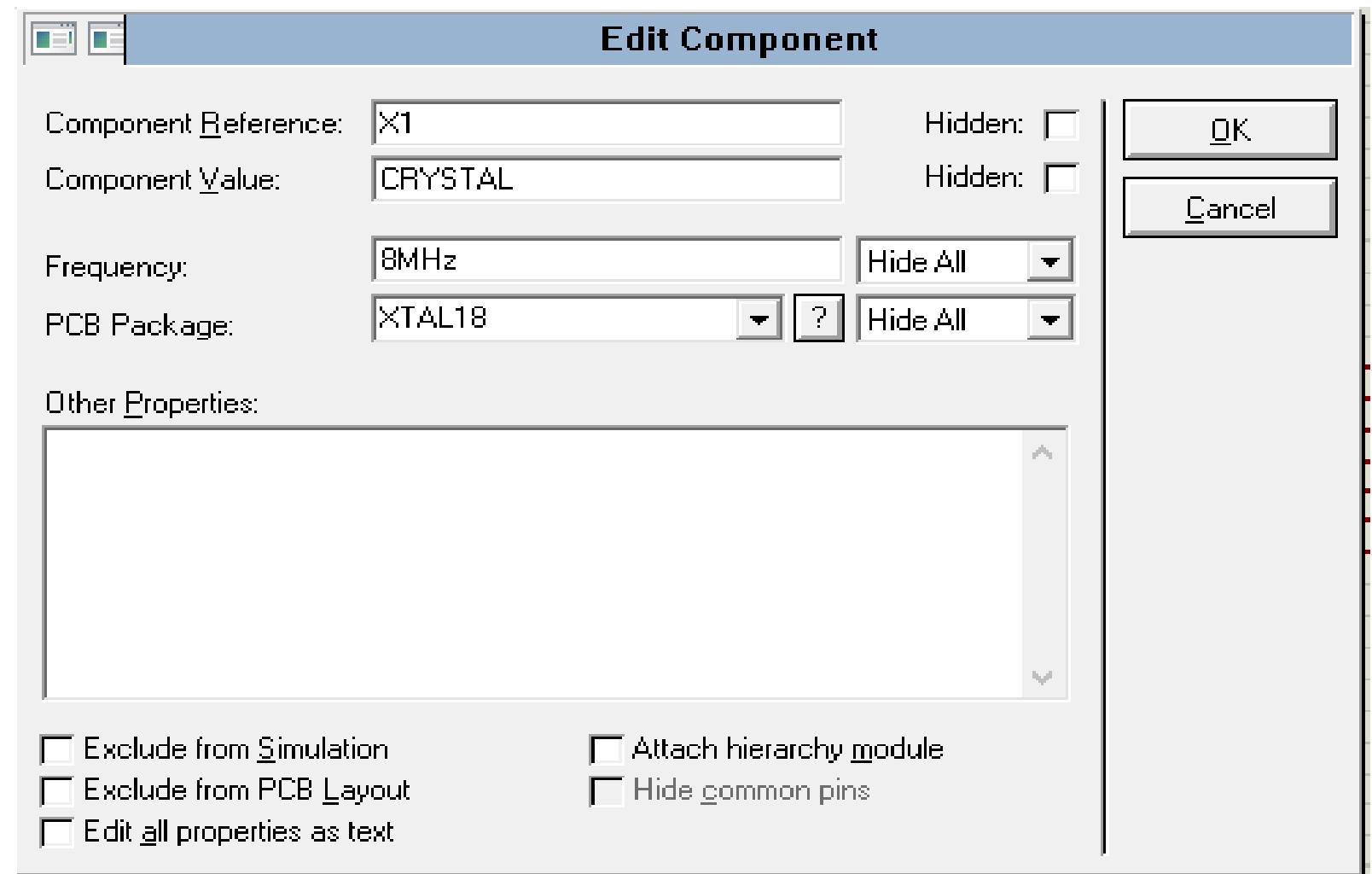
Procedure

Connect the circuit as shown in the figure.



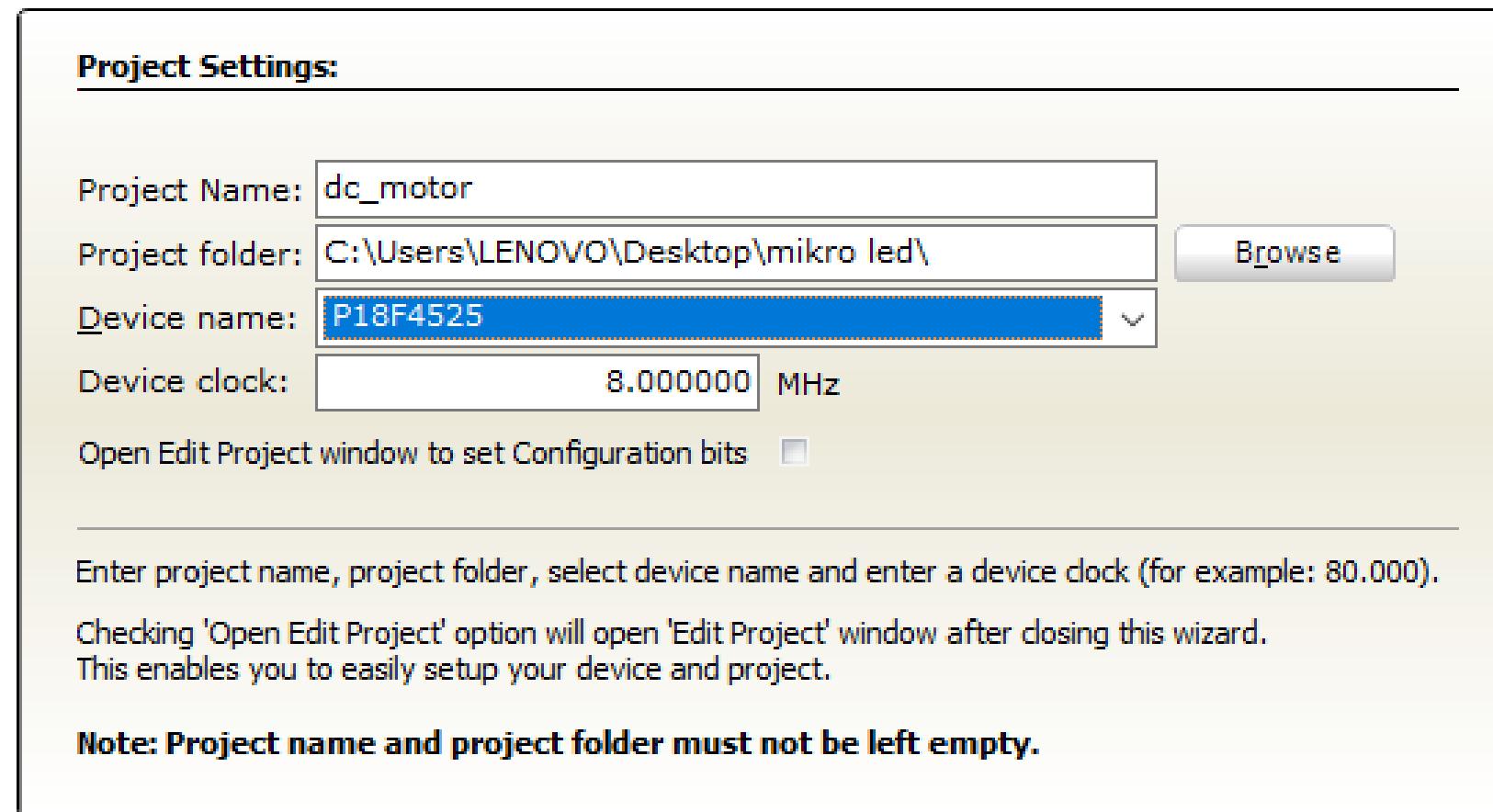
Procedure (Cont.)

Double click on the crystal and change the frequency to 8MHZ as shown in figure:



Procedure (Cont.)

Double click on the **pic18F4525A**, then the window in figure show to download the file of extension of (.hex) and select the processor clock frequency (in this experiment we select the frequency is 8MHZ the same frequency of the pic in the mikroC program.



❖ Procedure (Cont.)

Write the code in MikroC program:

```
// motor pin connections
sbit motor at RD0_bit;
sbit motor_Direction at TRISD0_bit;

// button on pin connections
sbit btn_on at RD1_bit;
sbit btn_on_Direction at TRISD1_bit;

// button off pin connections
sbit btn_off at RD2_bit;
sbit btn_off_Direction at TRISD2_bit;

void config()
{
    ADCON1 = 0b00001111; //all analog pins as digital

    motor_Direction = 0; //motor as output
    motor = 0; //motor initial value
```

❖ Procedure (Cont.)

Write the code in MikroC program:

```
//buttons as input
btn_on_Direction = 1;
btn_off_Direction = 1;
}

void main()
{
    config();

loop:

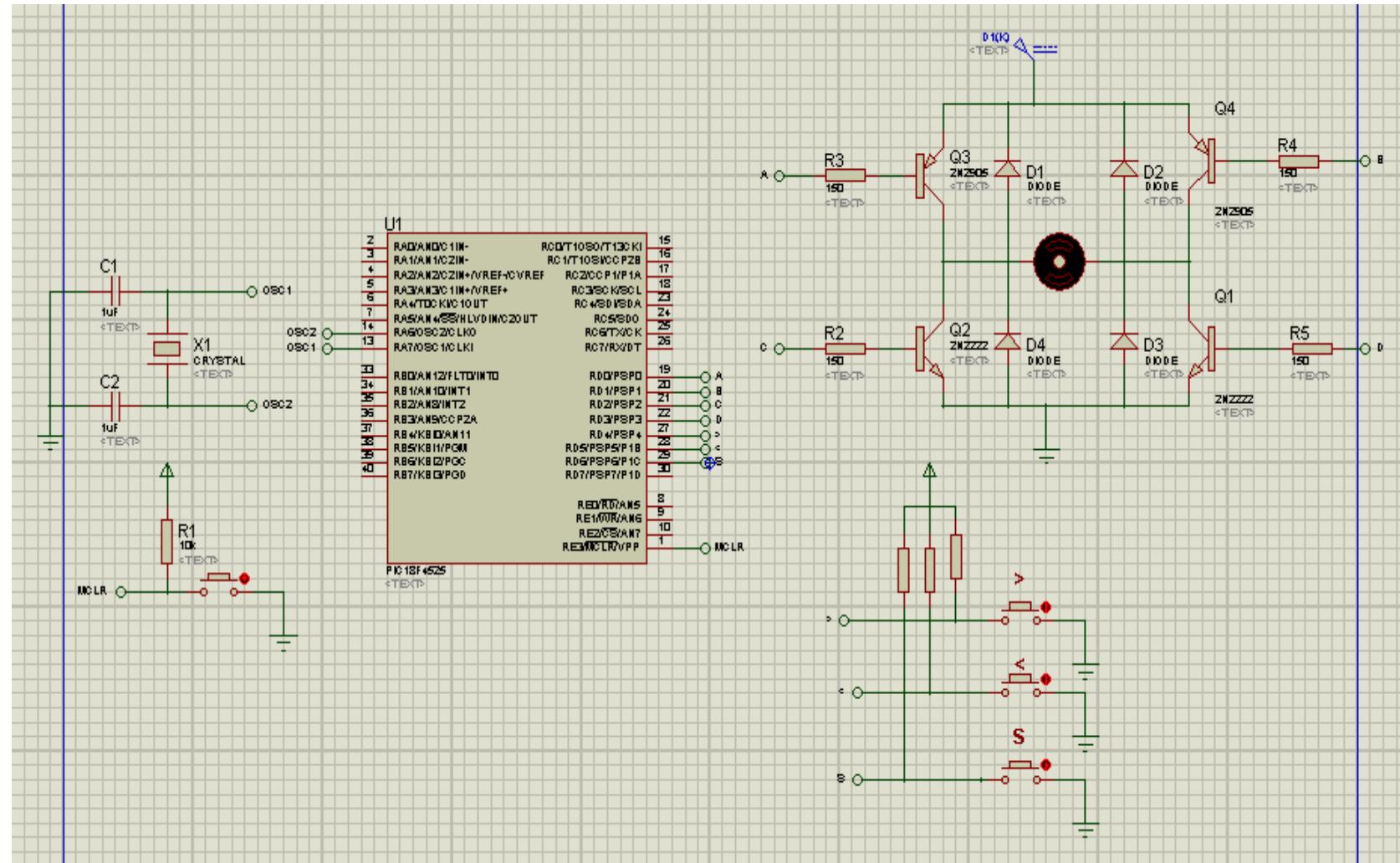
    if(btn_on == 0){motor=1; delay_ms(300);} //turn on motor if button clicked
    if(btn_off == 0){motor=0; delay_ms(300);} //turn on off motor if button clicked

    goto loop;
}
```

Procedure (Cont.)

To control the direction of DC motor that need to use the H-bridge as explain in theory. We connect the circuit 1 by using four transistors.

Connect the circuit as shown in the figure.



❖ Procedure (Cont.)

Write the code in MikroC
program:

```
// A Transistor motor pin connections
sbit _A at RD0_bit;
sbit _A_Direction at TRISD0_bit;

// B Transistor motor pin connections
sbit _B at RD1_bit;
sbit _B_Direction at TRISD1_bit;

// C Transistor motor pin connections
sbit _C at RD2_bit;
sbit _C_Direction at TRISD2_bit;

// D Transistor motor pin connections
sbit _D at RD3_bit;
sbit _D_Direction at TRISD3_bit;

// right button pin connections
sbit _right at RD4_bit;
sbit _right_Direction at TRISD4_bit;

// left button pin connections
sbit _left at RD5_bit;
sbit _left_Direction at TRISD5_bit;
```

Procedure (Cont.)

Write the code in MikroC
program:

```
// stop button Transistor pin connections
sbit _stop at RD6_bit;
sbit _stop_Direction at TRISD6_bit;

void config()
{
    ADCON1 = 0b00001111; //all analog pins as digital

    //control transistors as output
    _A_Direction = 0;
    _B_Direction = 0;
    _C_Direction = 0;
    _D_Direction = 0;

    _A = _B = 1; _C = _D = 0; //initial values for transistors

    _right_Direction = _left_Direction = _stop_Direction = 1; // buttons as input
}
```

Procedure (Cont.)

Write the code in MikroC
program:

```
void main()
{
    config();

loop:

    if(_stop == 0){_A =1; _B =1; _C =0; _D = 0; delay_ms(300);} //if stop cliked turn of transistors

    if(_right == 0) //if right button clicked
    {
        _B=1;_C=0; //turn off B(PNP) & C(NPN)
        _A=0;_D=1; //turn on A(PNP) & D(NPN)
        delay_ms(300);
    }
    if(_left == 0) //if left button clicked
    {
        _A=1;_D=0; //turn off A(PNP) & D(NPN)
        _B=0;_C=1; //turn on B(PNP) & C(NPN)
        delay_ms(300);
    }

    goto loop;
}
```

Discussion

1- Write and design the program by using mikroC and protues to control the dc motor by using the second circuit that shown in theory.

EXPERIMENT 12

Stepper Motor



❖ Experiment Object

The object of this experiment is to learn how to use stepper motor with microcontroller.

❖ Theory

■ What is a Stepper Motor?

Stepper Motor is a brushless electromechanical device which converts the train of electric pulses applied at their excitation windings into precisely defined step-by-step mechanical shaft rotation. The shaft of the motor rotates through a fixed angle for each discrete pulse. This rotation can be linear or angular. It gets one step movement for a single pulse input.

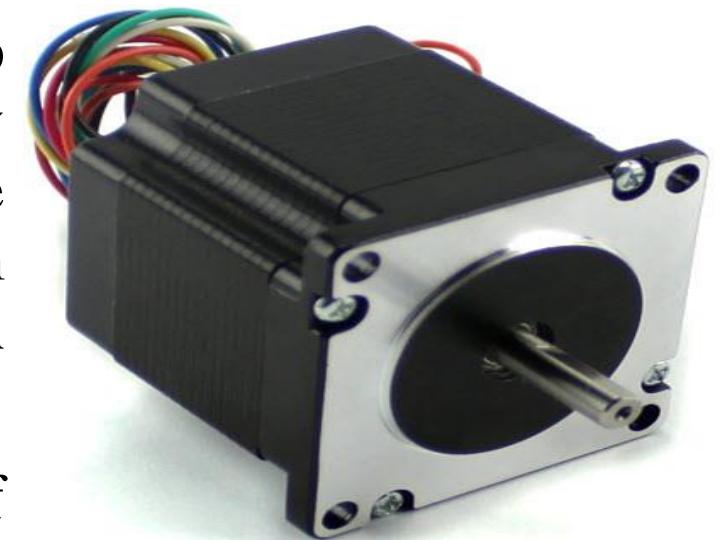
When a train of pulses is applied, it gets turned through a certain angle. The angle through which the stepper motor shaft turns for each pulse is referred as the step angle, which is generally expressed in degrees.

❖ Theory (Cont.)

■ What is a Stepper Motor?

The number of input pulses given to the motor decides the step angle and hence the position of motor shaft is controlled by controlling the number of pulses. This unique feature makes the stepper motor to be well suitable for open-loop control system wherein the precise position of the shaft is maintained with exact number of pulses without using a feedback sensor.

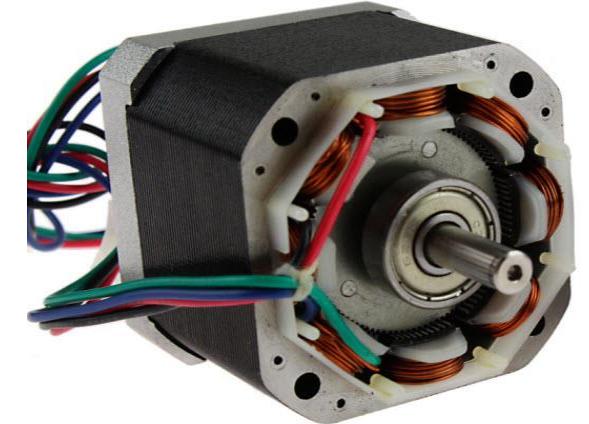
If the step angle is smaller, the greater will be the number of steps per revolutions and higher will be the accuracy of the position obtained. The step angles can be as large as 90 degrees and as small as 0.72 degrees, however, the commonly used step angles are 1.8 degrees, 2.5 degrees, 7.5 degrees and 15 degrees.



✿ Theory (Cont.)

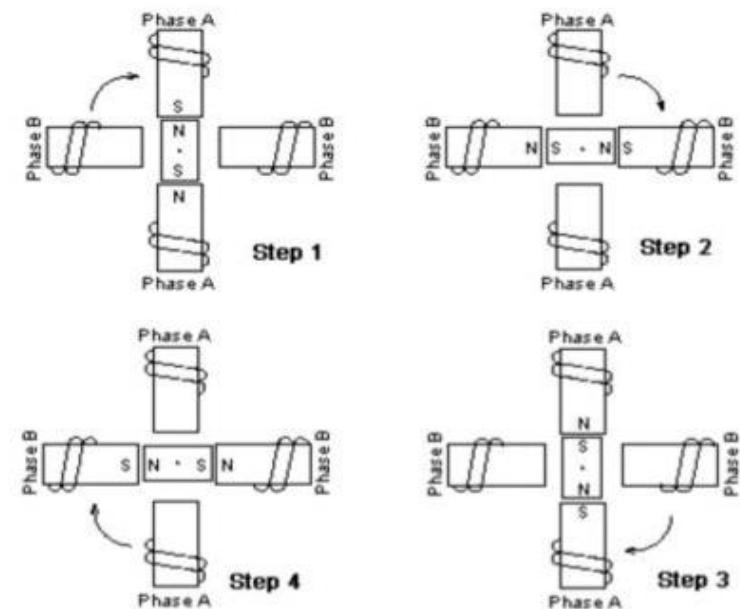
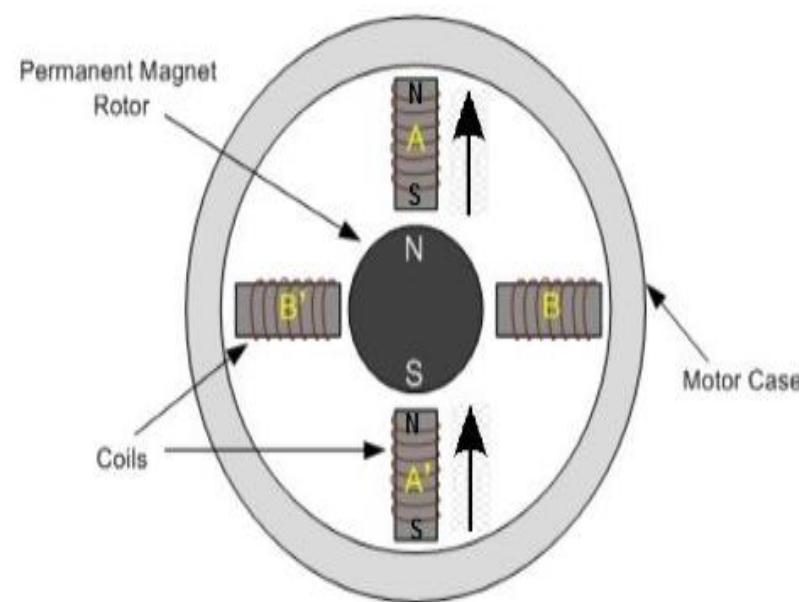
■ What is a Stepper Motor?

The direction of the shaft rotation depends on the sequence of pulses applied to the stator. The speed of the shaft or the average motor speed is directly proportional to the frequency (the rate of input pulses) of input pulses being applied at excitation windings. Therefore, if the frequency is low, the stepper motor rotates in steps and for high frequency, it continuously rotates like a DC motor due to inertia.



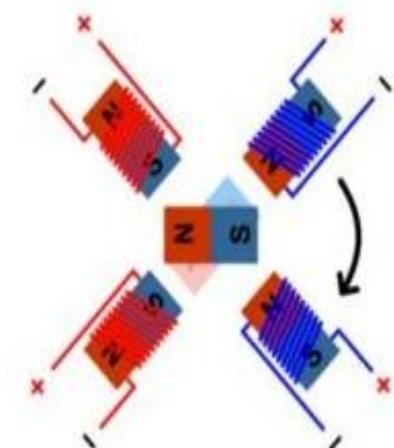
✿ Theory (Cont.)

- Bipolar Stepper Motor Full Step 90 Degree:

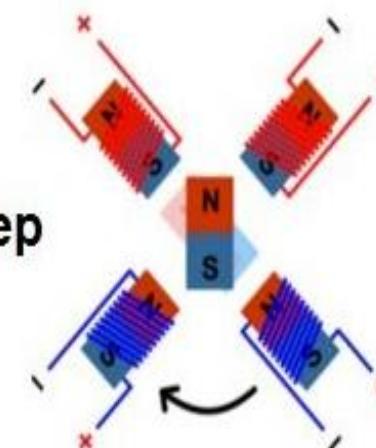


✿ Theory (Cont.)

- Bipolar Stepper Motor Full Step 90 Degree:

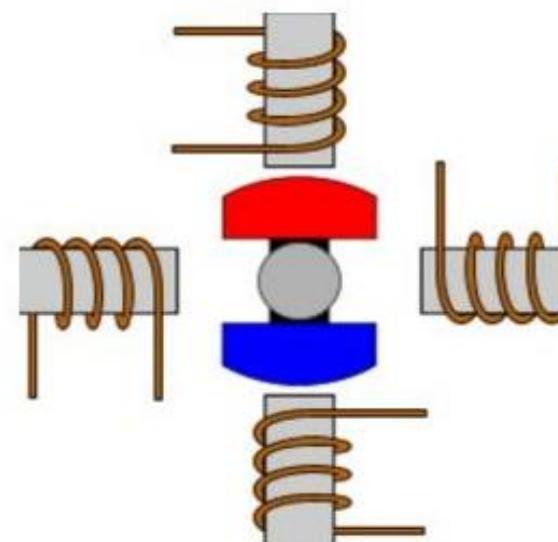


Bipolar Half Step



✿ Theory (Cont.)

- Unipolar Stepper Motor:

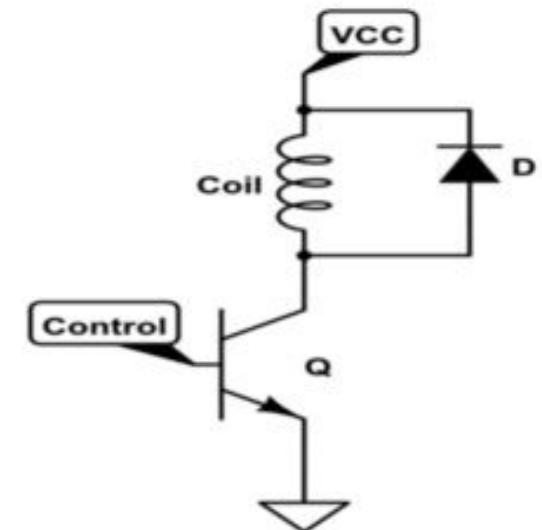
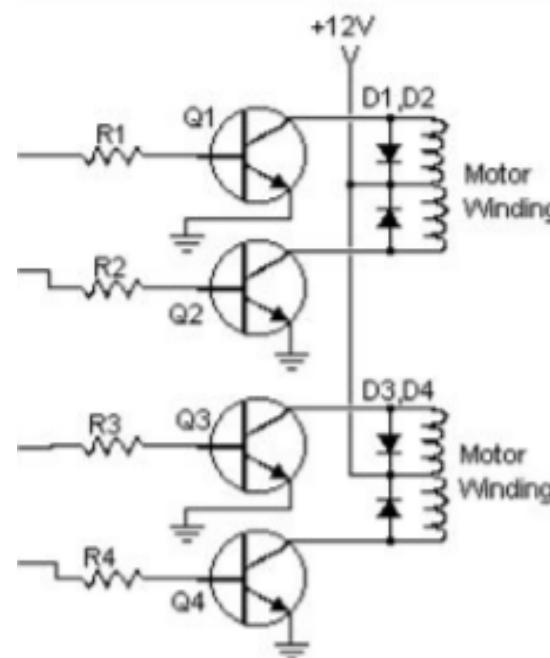


❖ Theory (Cont.)

■ Unipolar Stepper Motor Driver:

Using Transistor:

Due to the need for the engine to run a large current can not be connected with the engine directly. It will draw more current than the engine can take out. This may damage the controller so transistors are used for this purpose.

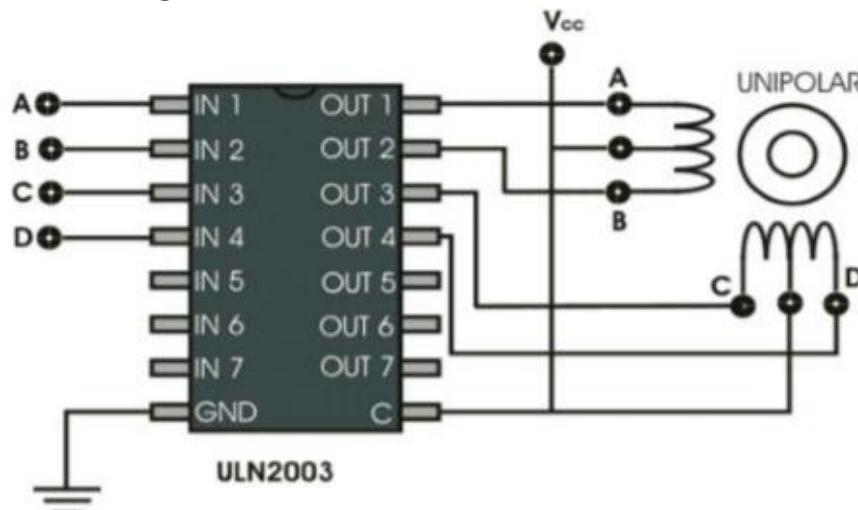


❖ Theory (Cont.)

■ Unipolar Stepper Motor Driver:

Using Transistor:

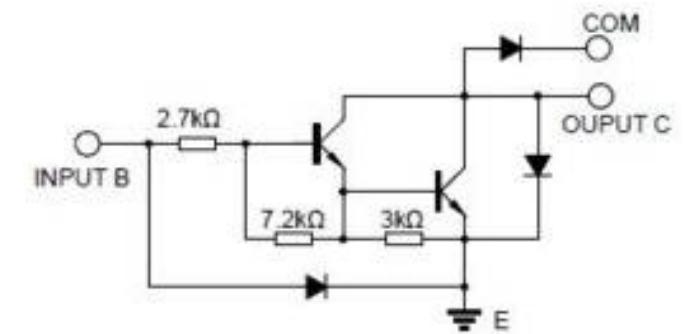
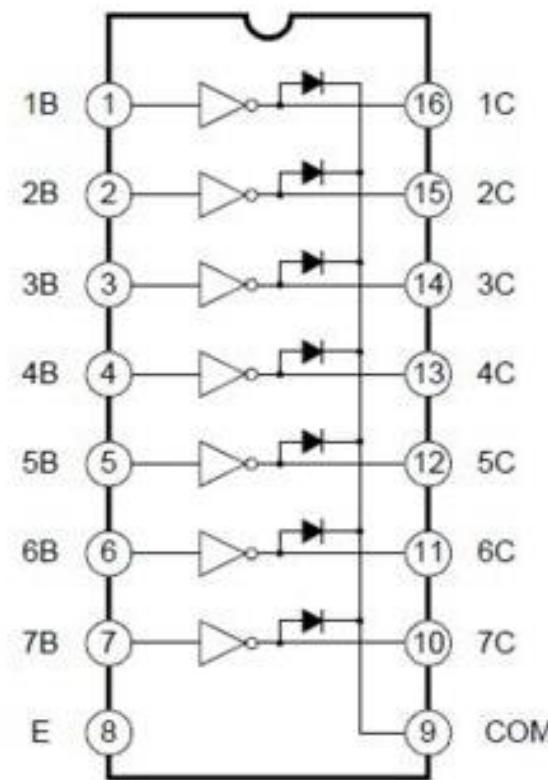
Note when using transistors the connection of the circuit may be complex or the components of the drive may be many. An integrated circuit (ULN2003) containing all these elements can be used internally.



✿ Theory (Cont.)

■ Unipolar Stepper Motor Driver:

Using Transistor:

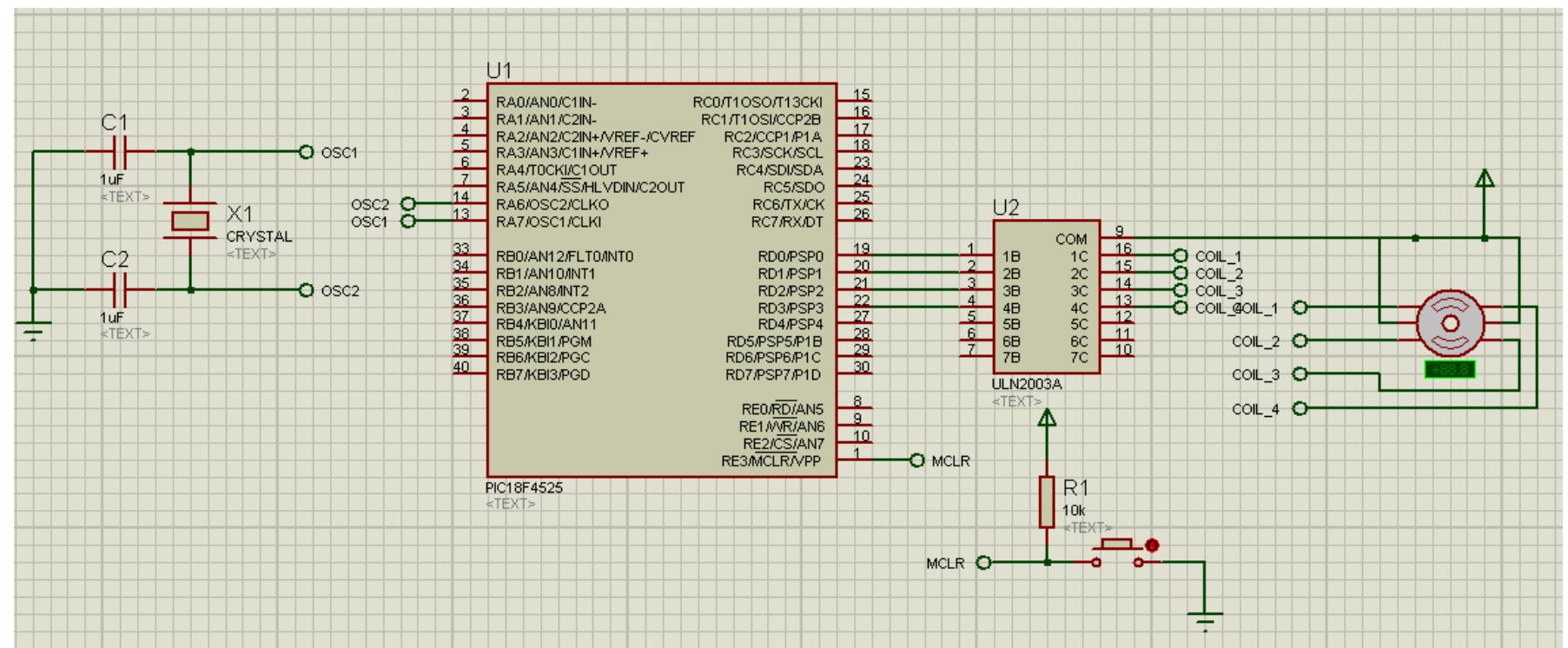


✿ The Experiment Components

- Microcontroller Pic 18F4525.
- Resistor 150 ohm.
- Push Button.
- Crystal 8MHZ.
- Capacitor 10 μ F.
- Transistor 2N2222.
- Driver ULN2003A.
- Bipolar stepper motor.

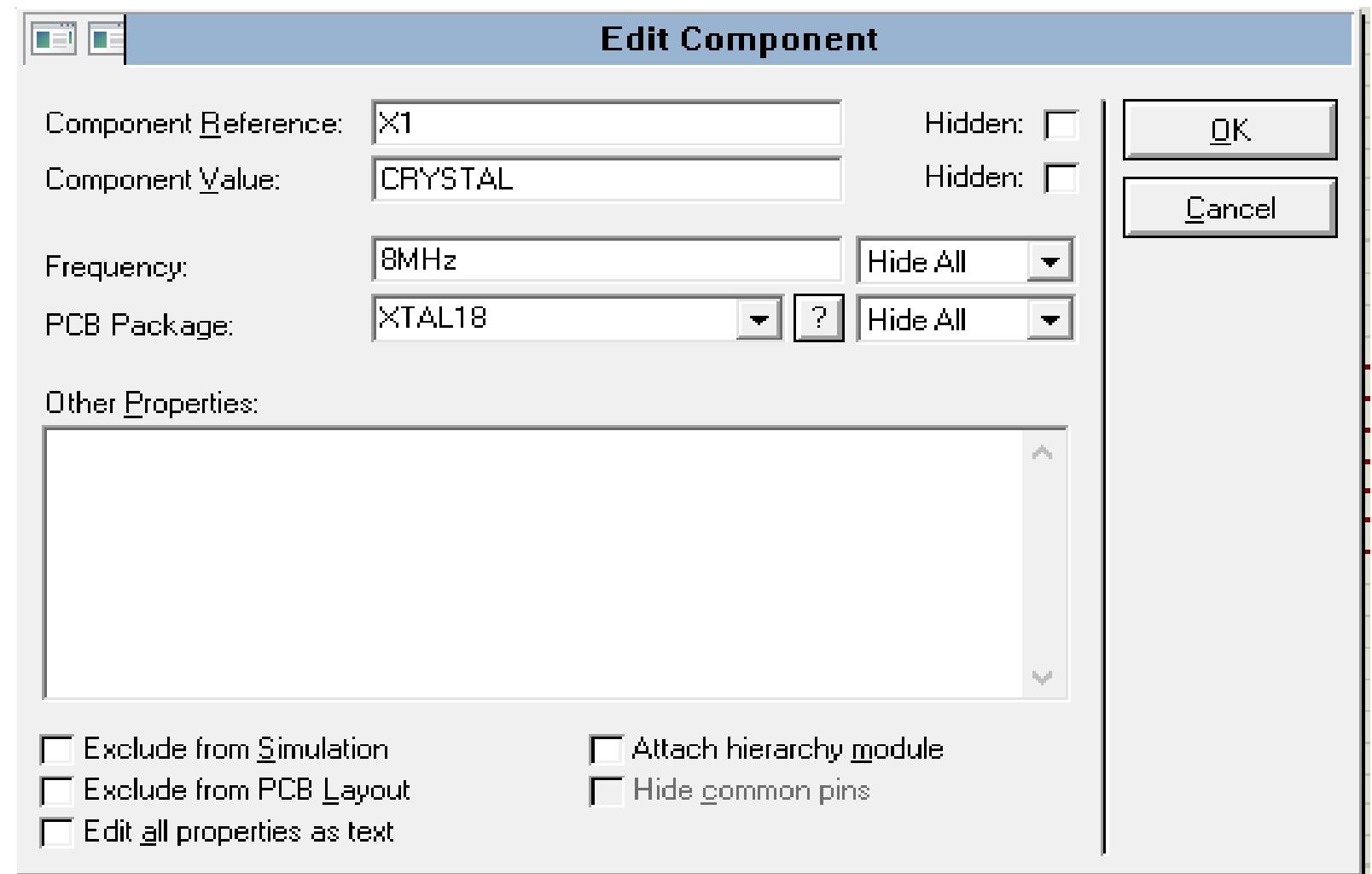
Procedure

Connect the circuit as shown in the figure.



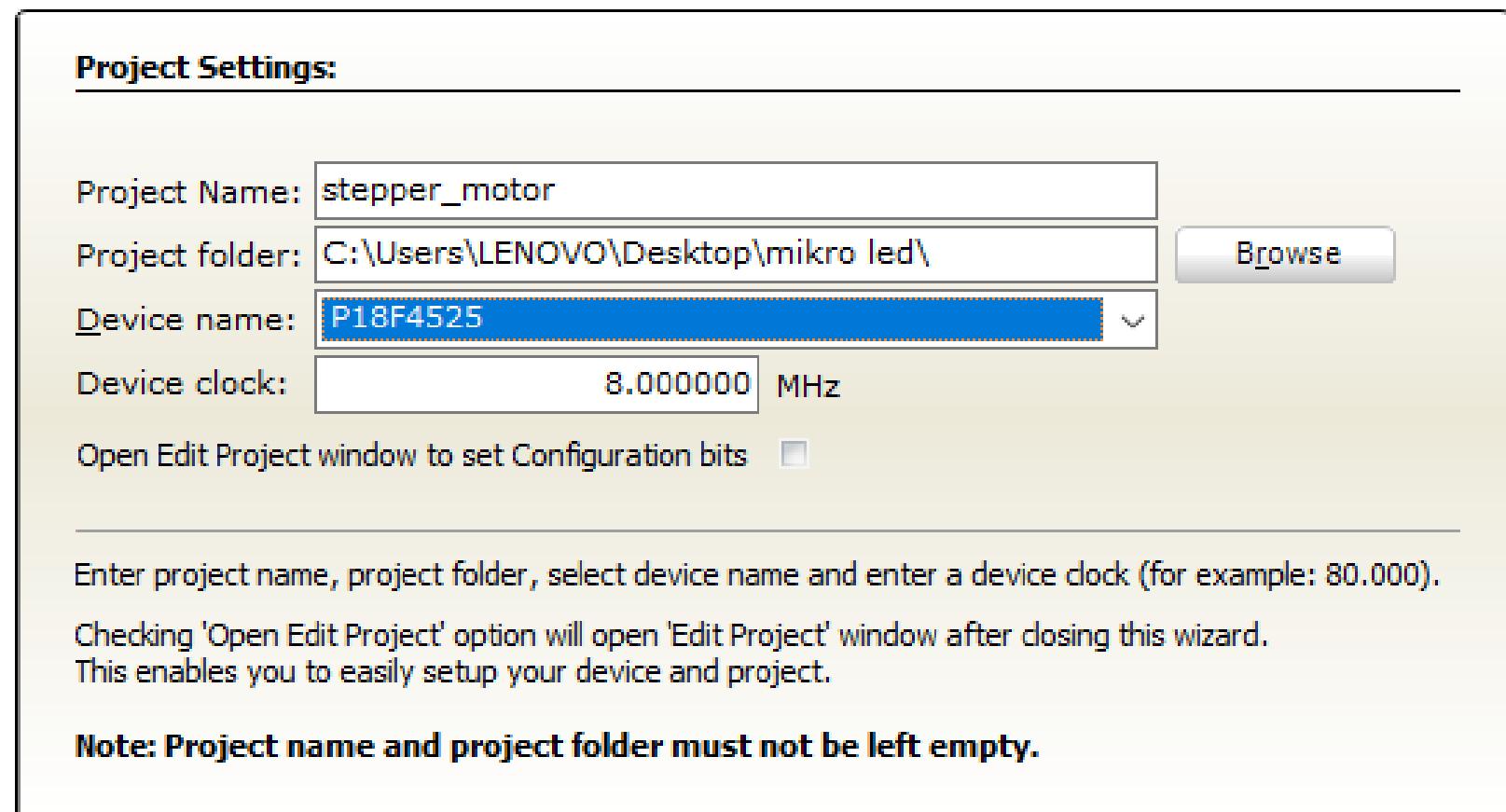
Procedure (Cont.)

Double click on the crystal and change the frequency to 8MHZ as shown in figure:



❖ Procedure (Cont.)

Double click on the **pic16F4525A**, then the window in figure show to download the file of extension of (**.hex**) and select the processor clock frequency (in this experiment we select the frequency is **8MHZ** the same frequency of the pic in the mikroC program.



❖ Procedure (Cont.)

Write the code in MikroC
program:

```
// coil_1 pin connections
sbit coil_1 at RD0_bit;
sbit coil_1_Direction at TRISD0_bit;

// coil_2 pin connections
sbit coil_2 at RD1_bit;
sbit coil_2_Direction at TRISD1_bit;

// coil_3 pin connections
sbit coil_3 at RD2_bit;
sbit coil_3_Direction at TRISD2_bit;

// coil_4 pin connections
sbit coil_4 at RD3_bit;
sbit coil_4_Direction at TRISD3_bit;

unsigned int step_number;

void config()
{
    ADCON1 = 0b00001111; //all analog pins as digital
    //pins of mcu that controls the motor, as outputs
```

❖ Procedure (Cont.)

Write the code in MikroC
program:

```
coil_1_Direction = 0;
coil_2_Direction = 0;
coil_3_Direction = 0;
coil_4_Direction = 0;

//turn off coils of motor at the beginning
coil_1 = 0;
coil_2 = 0;
coil_3 = 0;
coil_4 = 0;

}

void motor_off()
{
    coil_1 = 0;
    coil_2 = 0;
    coil_3 = 0;
    coil_4 = 0;
}
```

❖ Procedure (Cont.)

Write the code in MikroC
program:

```
void step_1()
{
    coil_1 = 1;
    coil_2 = 0;
    coil_3 = 0;
    coil_4 = 0;
}

void step_2()
{
    coil_1 = 0;
    coil_2 = 1;
    coil_3 = 0;
    coil_4 = 0;
}

void step_3()
{
    coil_1 = 0;
    coil_2 = 0;
    coil_3 = 1;
    coil_4 = 0;
}
```

❖ Procedure (Cont.)

Write the code in MikroC
program:

```
void step_4()
{
    coil_1 = 0;
    coil_2 = 0;
    coil_3 = 0;
    coil_4 = 1;
}

void step_to_the_left(unsigned int step_number)
{
    while(step_number > 0) //loop until steps 0
    {
        //move to the left
        step_1(); //30
        delay_ms(200);
        step_2(); //30
        delay_ms(200);
        step_3(); //30
        delay_ms(200);
        step_4(); //30
        delay_ms(200);

        step_number--; //decreas after each step
    }
}
```

❖ Procedure (Cont.)

Write the code in MikroC
program:

```
    motor_off(); //turn off the motor when done
}

void step_to_the_right(unsigned int step_number)
{
    while(step_number > 0) //loop until steps 0
    {
        //move to the right
        step_4();
        delay_ms(200);
        step_3();
        delay_ms(200);
        step_2();
        delay_ms(200);
        step_1();
        delay_ms(200);

        step_number--; //decreases after each step
    }
    motor_off(); //turn off the motor when done
}

void main()
{
    config();
}
```

❖ Procedure (Cont.)

Write the code in MikroC
program:

```
step_to_the_right(50); //send Loop value
delay_ms(200);
step_to_the_left(50);
delay_ms(200);

}
```

❖ Discussion

- 1- Write and design the program by using mikroC and protues to control the stepper motor by two buttons. The first that turn the motor to the right and the second button turn the motor to the left.
- 2- Write and design the program by using mikroC and protues to control the stepper motor by four buttons. Using two buttons to control the directions of the motor and the another two buttons to specify the number of steps(when press the button the step increase and when press another button the steps decrease).every thing that display on the LCD.

EXPERIMENT 13

Ultrasonic Sensor



❖ Experiment Object

The object of this experiment is to learn how to use ultrasonic sensor with microcontroller.

❖ Theory

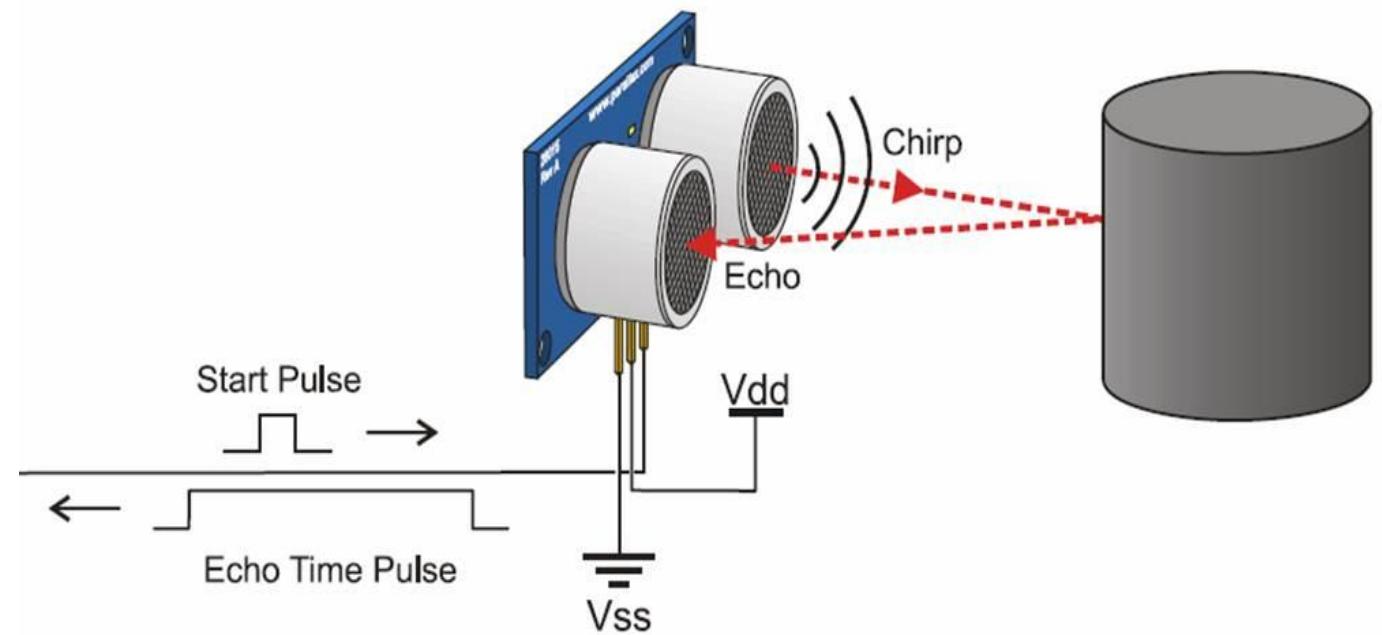
■ Ultrasonic Sensor

ultrasonic Distance sensor. It generates sound waves beyond the scope of human hearing and measures distance by calculating the time required by these waves to hit an obstacle and travel back.

The principle of an ultrasonic sensor like sonar is as follows. A short ultrasonic sound (typically 40 kHz frequency) is emitted by the sensor. The duration time is then measured for the sound to propagate forward, reflect (i.e. echo) off the object of interest, and return to the sensor. This process is indicated in the diagram below.

Theory

■ Ultrasonic Sensor (Cont.)



❖ Theory

■ Ultrasonic Sensor (Cont.)

The following equation then holds between the distance of the sensor to the object, the speed of sound c , and the duration.

$$2 * \text{distance} = c * \text{duration}$$

$$\text{distance} = c * \text{duration} / 2$$

The distance in **cm**: Formula: $\text{Timer(uS)} / 58 = \text{centimeters}$

The distance in **inch**: Formula: $\text{Timer(uS)} / 148 = \text{inch}$

❖ Theory

■ Ultrasonic Sensor (Cont.)

Distance calculation: $D = t / (58*2)$

D : Distance

t : Timer (echo high to low)

58 : centimeter

2 : calibration

If Timer1=4035:

$$D = t / (58*2)$$

$$D = 4035 / 116$$

$$D = 34.78 \text{ cm} \rightarrow 34\text{cm}$$

❖ Theory

- Ultrasonic Sensor (Cont.)

Timer1?

$$D = t / (58 * 2)$$

$$34 = t / 116$$

$$t = 34.78 * 116$$

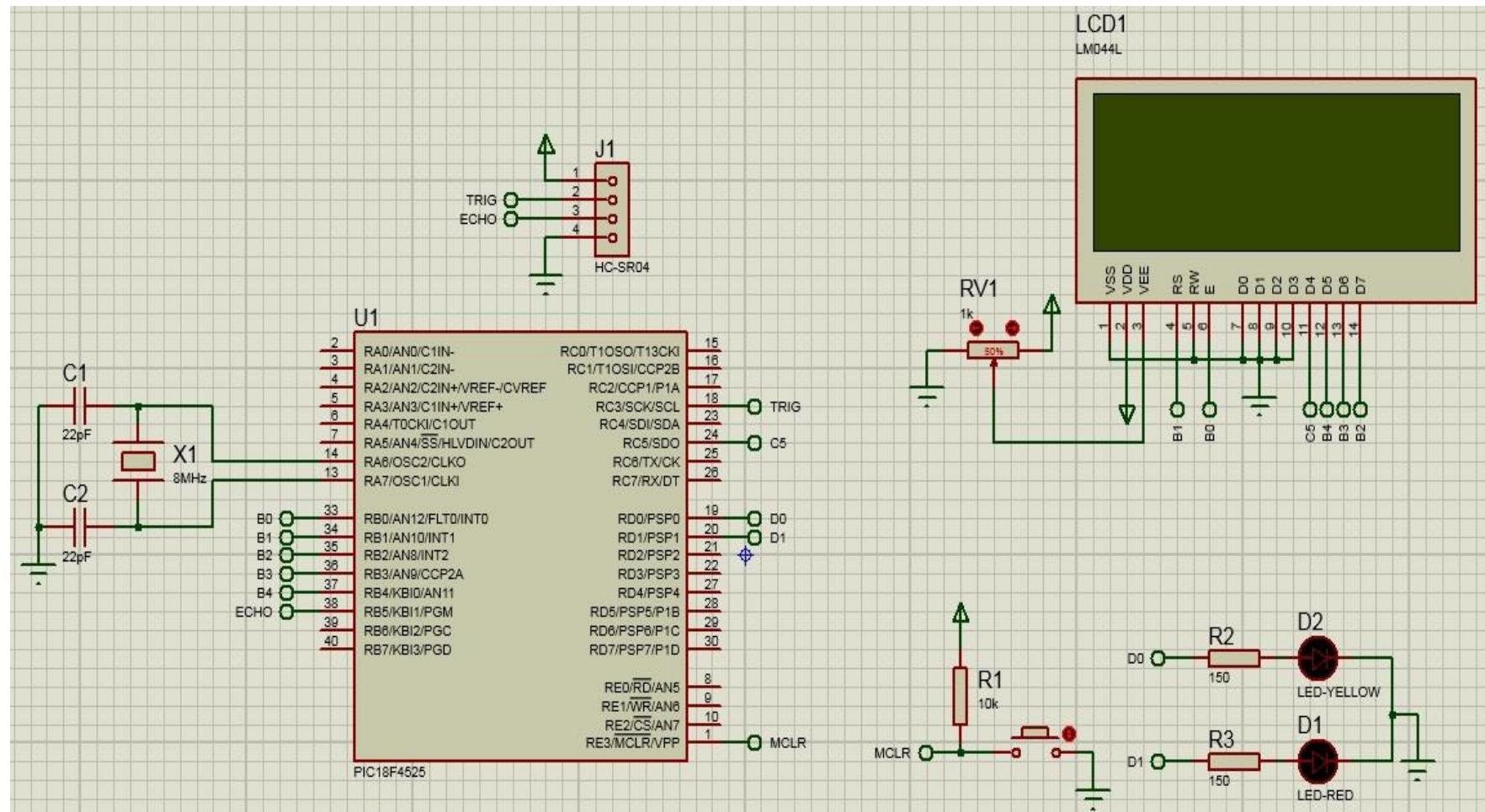
$$t = 4034.48 \mu\text{s} \rightarrow 4035 \mu\text{s}$$

✿ The Experiment Components

- Microcontroller Pic 18F4525.
- Resistor 150 ohm.
- Push Button.
- LCD 16x4.
- Ultrasonic.
- Potentiometer 5kohm.

Procedure

Connect the circuit as shown in the figure.



❖ Procedure (Cont.)

Write the code in MikroC
program:

```
// LCD module connections
sbit LCD_RS at RB1_bit;
sbit LCD_EN at RB0_bit;
sbit LCD_D4 at RC5_bit;
sbit LCD_D5 at RB4_bit;
sbit LCD_D6 at RB3_bit;
sbit LCD_D7 at RB2_bit;

sbit LCD_RS_Direction at TRISB1_bit;
sbit LCD_EN_Direction at TRISB0_bit;
sbit LCD_D4_Direction at TRISC5_bit;
sbit LCD_D5_Direction at TRISB4_bit;
sbit LCD_D6_Direction at TRISB3_bit;
sbit LCD_D7_Direction at TRISB2_bit;
// End LCD module connections

// Ultrasonic Sensor Trigger pin connections
sbit trig at RC3_bit;
sbit trig_Direction at TRISC3_bit;

// Ultrasonic Sensor Echo pin connections
sbit echo at RB5_bit;
sbit echo_Direction at TRISB5_bit;
```

Procedure (Cont.)

Write the code in MikroC
program:

```
// LED(Yellow) pin D0 connections
sbit led1 at RD0_bit;
sbit led1_Direction at TRISD0_bit;

// LED(Red) pin D1 connections
sbit led2 at RD1_bit;
sbit led2_Direction at TRISD1_bit;

unsigned int time; //stores timer1 value
char time_txt[8]; //stores time in uS converted to stirng
unsigned int distance; //stores the calculated distance
char distance_txt[8]; //stores distance converted to stirng

void config()
{
    ADCON1 = 0b00001111; //all analog pins as digital

    trig_Direction = 0; //Ultrasonic Sensor Trigger pin as output
    echo_Direction = 1; //Ultrasonic Sensor Echo pin as input
    led1_Direction = 0; led1 = 0; // LED1 as output
    led2_Direction = 0; led2 = 0; // LED2 as output
```

Procedure (Cont.)

Write the code in MikroC
program:

```
UART1_Init(9600); // Initialize hardware UART1 and establish communication at 9600 bps
UART1_Write_text("Welcome");
UART1_Write(10);
UART1_Write(13);

Lcd_Init(); // Initialize LCD
Lcd_Cmd(_LCD_CLEAR); // Clear display
Lcd_Cmd(_LCD_CURSOR_OFF); // Cursor off
Lcd_Out(1, 1, "Hello!");

led1 = 1; led2 = 1; //turn on leds 1 and 2 for 1second
delay_ms(1000);
Lcd_Cmd(_LCD_CLEAR); // Clear display
led1 = 0; led2 = 0; //turn off leds 1 and 2
lcd_out(1,1,"Ultrasonic Sensor");

T1CON = 0b00000000; //timer1 default configurations
}

void get_distance()
{
    //reset timer1 registers
    TMR1H = 0;
    TMR1L = 0;
```

Procedure (Cont.)

Write the code in MikroC
program:

```
//trigger for 10us
trig = 1;
Delay_us(10);
trig = 0;

while(echo == 0); //wait if echo is low
TMR1ON_bit = 1; //start counting, when echo is high
while(echo == 1); //wait if echo is high(returned signal is not received yet)
TMR1ON_bit = 0; //stop counting, when echo is low(returned signal is received)
//so now we have the time of echo high to echo low

time = ((TMR1H << 8) + TMR1L); //get high and low bytes of timer1, time in uS

//Convert Time to Distance
distance = (time / 58) / 2; //divide time by 58 to get distance in cm, divide by 2 to fix result(calibration)

if(distance >= 10 && distance <= 400) //if distance in range
{
    //convert distance and time to string
    wordToStr(distance,distance_txt);
    wordToStr(time,time_txt);
```

❖ Procedure (Cont.)

Write the code in MikroC
program:

```
//Trim the leading spaces of distance and time
Ltrim(distance_txt);
Ltrim(time_txt);

//display data on LCD
lcd_out(2,8,"");
lcd_out(3,11,""); //clear position before displaying data
lcd_out(2,1,"Time = ");
lcd_out(3,1,"Distance =");
lcd_out(2,8,time_txt);
lcd_out(3,12,distance_txt);
lcd_out(2,14,"uS");
lcd_out(3,16,"cm");

//send data to computer
uart1_write_text(time_txt);
uart1_write_text("uS\n");
uart1_write_text(distance_txt);
uart1_write_text("cm\n");

led2 = 0; //turn off led2
led1 = ~led1; //toggle led1 every new value
}
```

Procedure (Cont.)

Write the code in MikroC
program:

```
else //otherwise distance not in range
{
    lcd_out(4,1,"          ");
    lcd_out(4,1,"Out of Range !");
    delay_ms(500);
    lcd_out(4,1,"          ");

    uart1_write_text("Out of Range !\n");

    led1 = 0; //turn off led1
    led2 = ~led2; //toggle led2
}

Delay_ms(500);

}

void main()
{
    config();

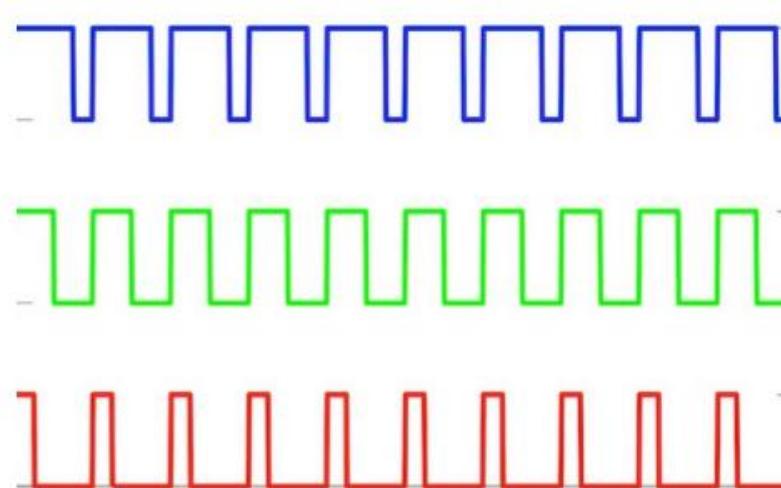
    while(1)
    {
        get_distance();
    }
}
```

❖ Discussion

- 1- Write and design the program by using mikroC and protues to measure the distance of the object and display the values and explained the approach and away the object on the GLCD display(draw simple thing on the GLCD).

EXPERIMENT 14

PWM

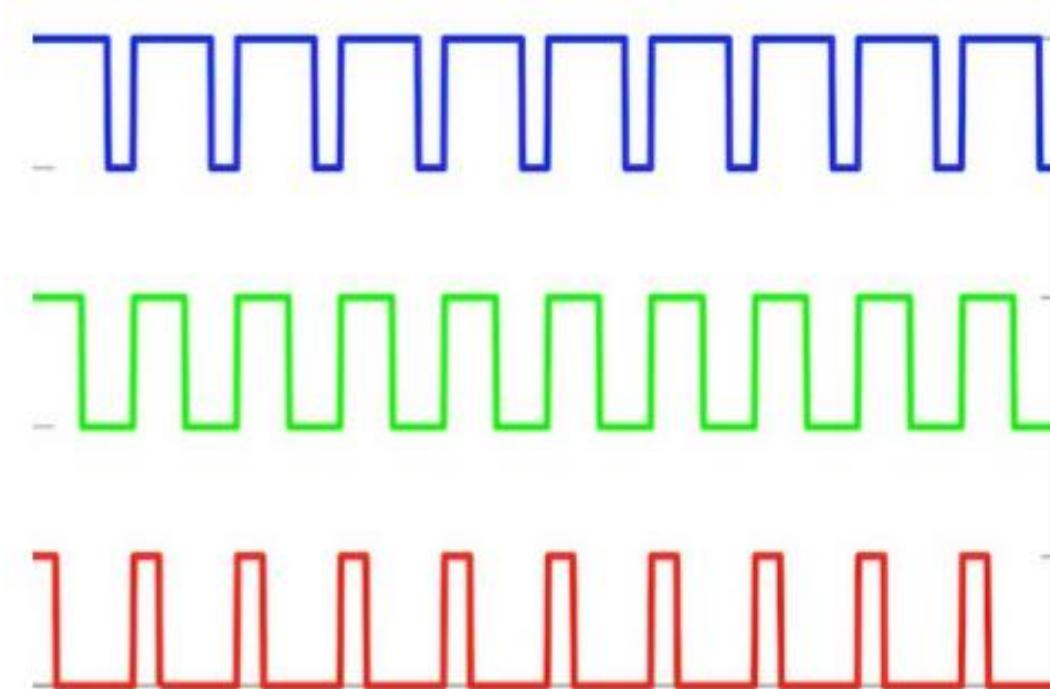


Experiment Object

The object of this experiment is to learn how to use PWM signal with microcontroller.

Theory

- Pulse Width Modulation Signal

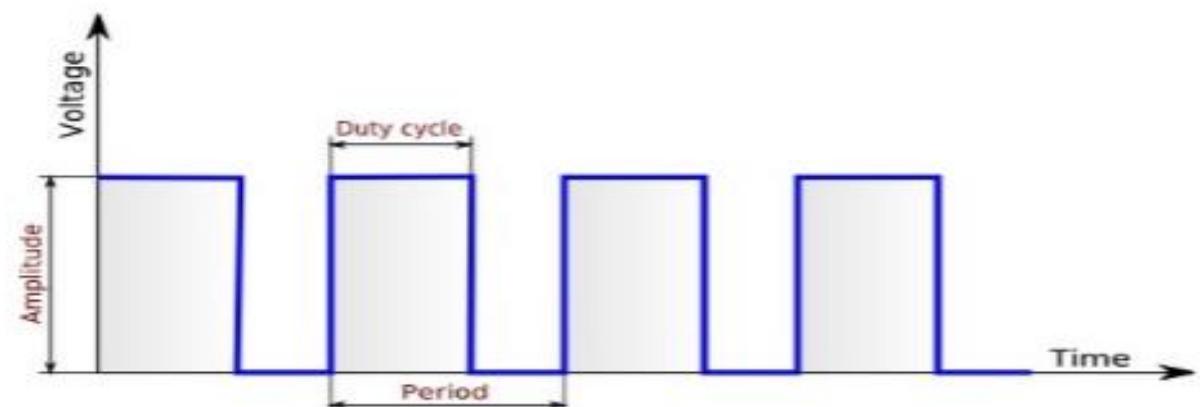
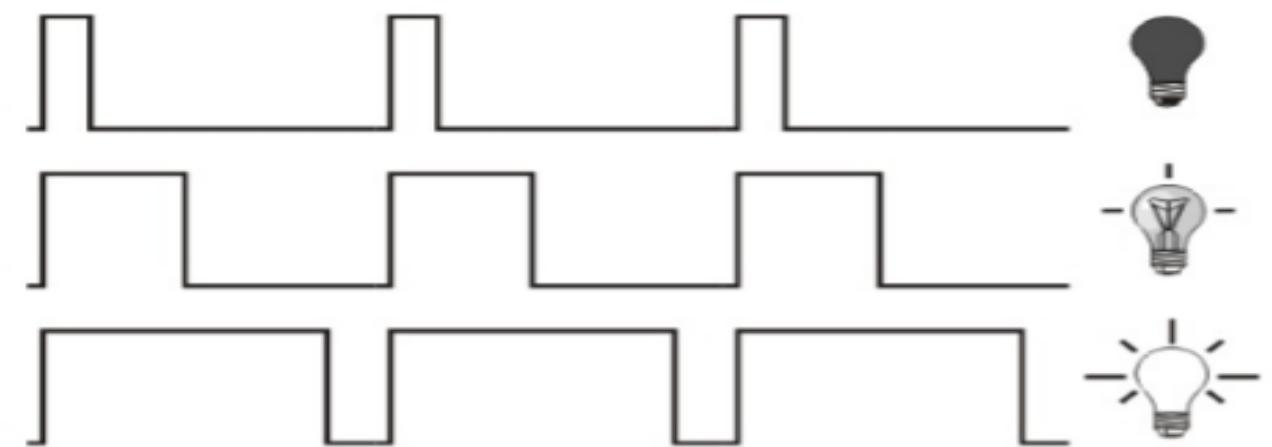


✿ Theory

■ Pulse Width Modulation Signal

Use PWM:

- Light intensity control
- Motor speed control
- Digital to analog conversion



❖ Theory

■ Pulse Width Modulation Signal

What is duty cycle:

The time period in which the pulse is high, i.e. the percentage of time that the element or device was made. The work cycle is expressed in percentage.

$$\text{duty cycle} = \frac{t_{\text{ON}}}{t_{\text{ON}} + t_{\text{OFF}}} \times 100\%$$

t_{ON} = ON time

t_{OFF} = OFF time

$t_{\text{ON}} + t_{\text{OFF}}$ = Time period

$$f = \frac{1}{T}$$

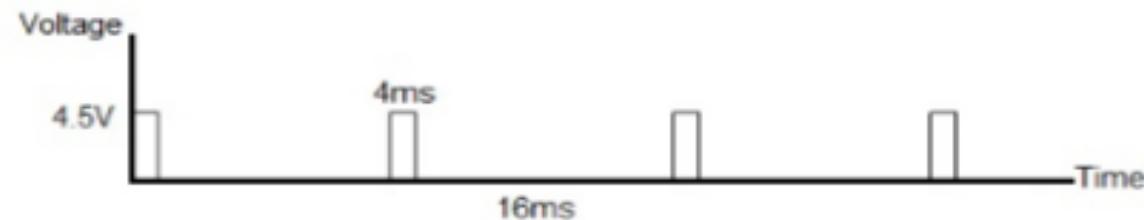
$$V_{\text{out}} = \text{duty cycle} * \text{voltage}$$

✿ Theory

■ Pulse Width Modulation Signal

Calculate:

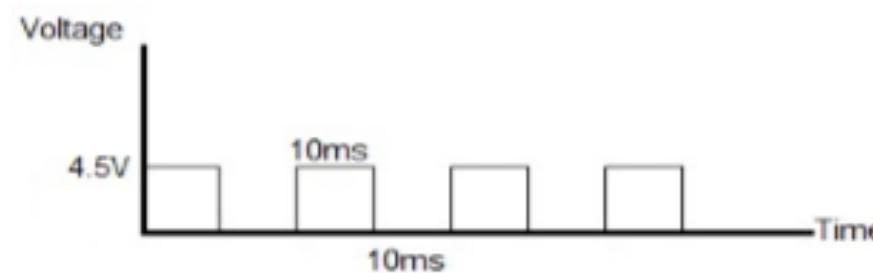
- T.
- Vout.
- Duty cycle.



$$T = T_{on} + T_{off} = 4\text{ms} + 16\text{ms} = 20\text{ms}$$

$$\text{duty cycle} = (T_{on} / T) * 100\% = (4 / 20) = 0.2 \rightarrow 0.2 * 100 = 20\%$$

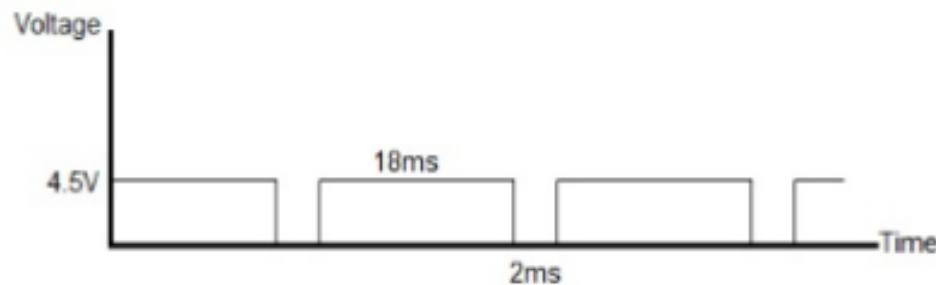
$$V_{out} = \text{duty cycle} * \text{Voltage} = 0.2 * 4.5 = 0.9\text{V}$$



$$T = T_{on} + T_{off} = 10\text{ms} + 10\text{ms} = 20\text{ms}$$

$$\text{duty cycle} = (T_{on} / T) * 100\% = (10 / 20) = 0.5 \rightarrow 0.5 * 100 = 50\%$$

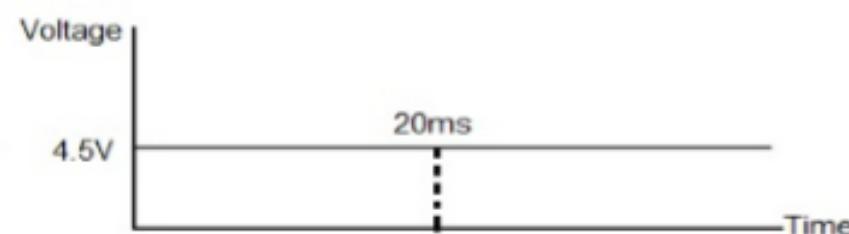
$$V_{out} = \text{duty cycle} * \text{Voltage} = 0.5 * 4.5 = 2.25\text{V}$$



$$T = T_{on} + T_{off} = 18\text{ms} + 2\text{ms} = 20\text{ms}$$

$$\text{duty cycle} = (T_{on} / T) * 100\% = (18 / 20) = 0.9 \rightarrow 0.9 * 100 = 90\%$$

$$V_{out} = \text{duty cycle} * \text{Voltage} = 0.9 * 4.5 = 4.05\text{V}$$



$$T = T_{on} + T_{off} = 20\text{ms}$$

$$\text{duty cycle} = (T_{on} / T) * 100\% = (20 / 20) = 1 \rightarrow 1 * 100 = 100\%$$

$$V_{out} = \text{duty cycle} * \text{Voltage} = 1 * 4.5 = 4.5\text{V}$$

Theory

Pulse Width Modulation Signal

Theory

■ Pulse Width Modulation Signal

How to adjust Duty Cycle?

$$\text{Duty Cycle} = (\text{Value}\%) * (255)$$

If duty cycle=50%.

$$\text{Value}\% = 50 / 100 = 0.5$$

$$\text{Duty Cycle} = (0.5) * (255) = 127.5 \rightarrow 128$$

If duty cycle=100%.

$$\text{Value}\% = 100 / 100 = 1$$

$$\text{Duty Cycle} = (1) * (255) = 255$$

If duty cycle=75%.

$$\text{Value}\% = 75 / 100 = 0.75$$

$$\text{Duty Cycle} = (0.75) * (255) = 191.25 \rightarrow 191$$

✿ Theory

■ Pulse Width Modulation Signal

If duty cycle=25%.

$$\text{Value\%} = 25 / 100 = 0.25$$

$$\text{Duty Cycle} = (0.25) * (255) = 63.75 \rightarrow 64$$

Use PWM to Generate Frequencies: Period Register = $((1 / \text{Frequency}) / (1 / \text{Fosc} * 4 * \text{prescaler})) - 1$

Calculate the periodic time and frequency of the PWM:

$$\text{PWM Period} = (\text{PR2} + 1) * (4) * (1/\text{Fosc}) * (\text{Prescaler})$$

$$\text{PWM Frequency} = 1 / \text{PWM Period}$$

❖ Theory

■ Pulse Width Modulation Signal

Generate a frequency of 4KHz using PWM:

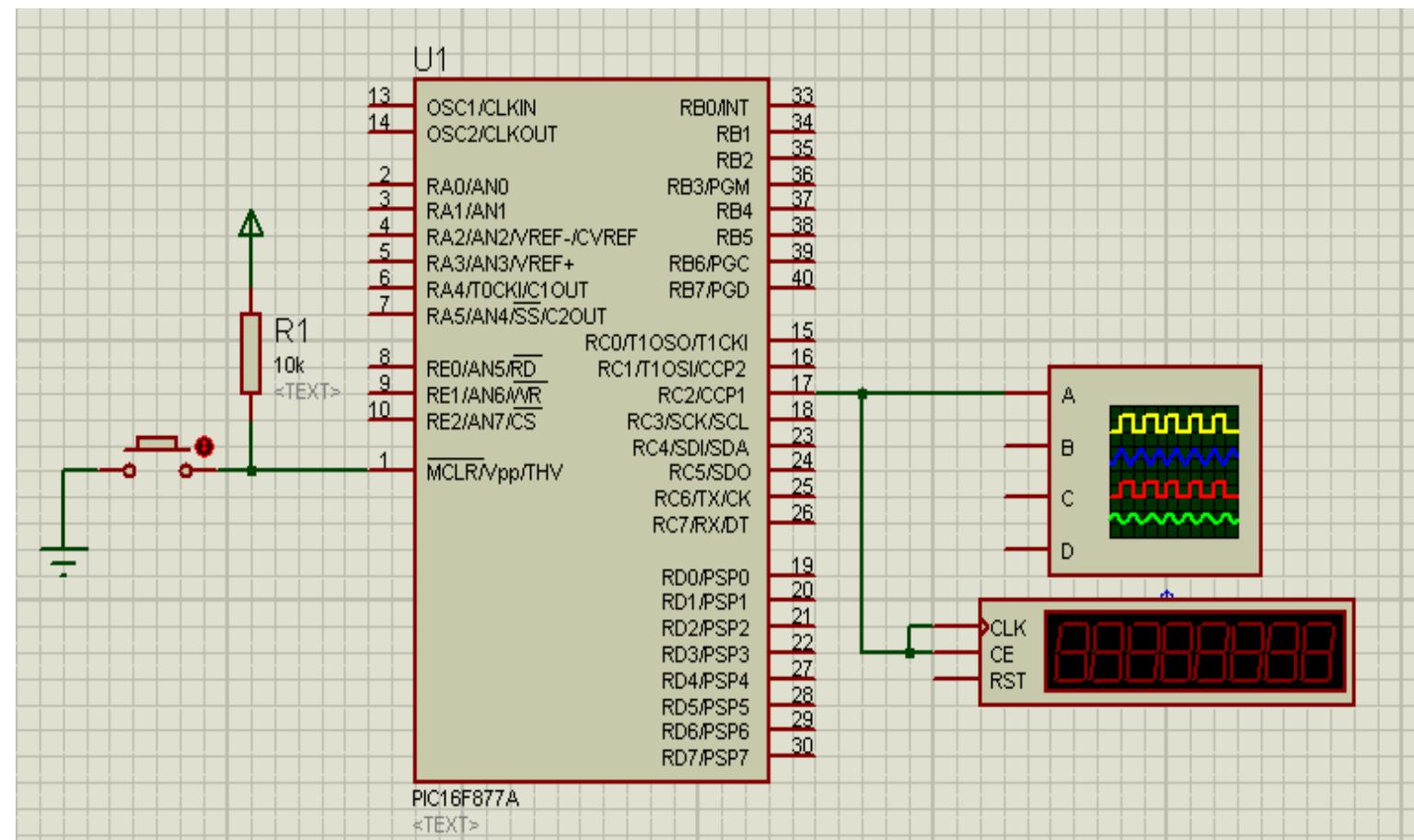
$$\begin{aligned}\text{Period Register} &= ((1 / 4000) / (1 / 20000000 * 4 * 16)) - 1 \\ &= 77.125 \rightarrow 77\end{aligned}$$

✿ The Experiment Components

- Microcontroller Pic 16F877A.
- Resistor 150 ohm.
- Push Button.

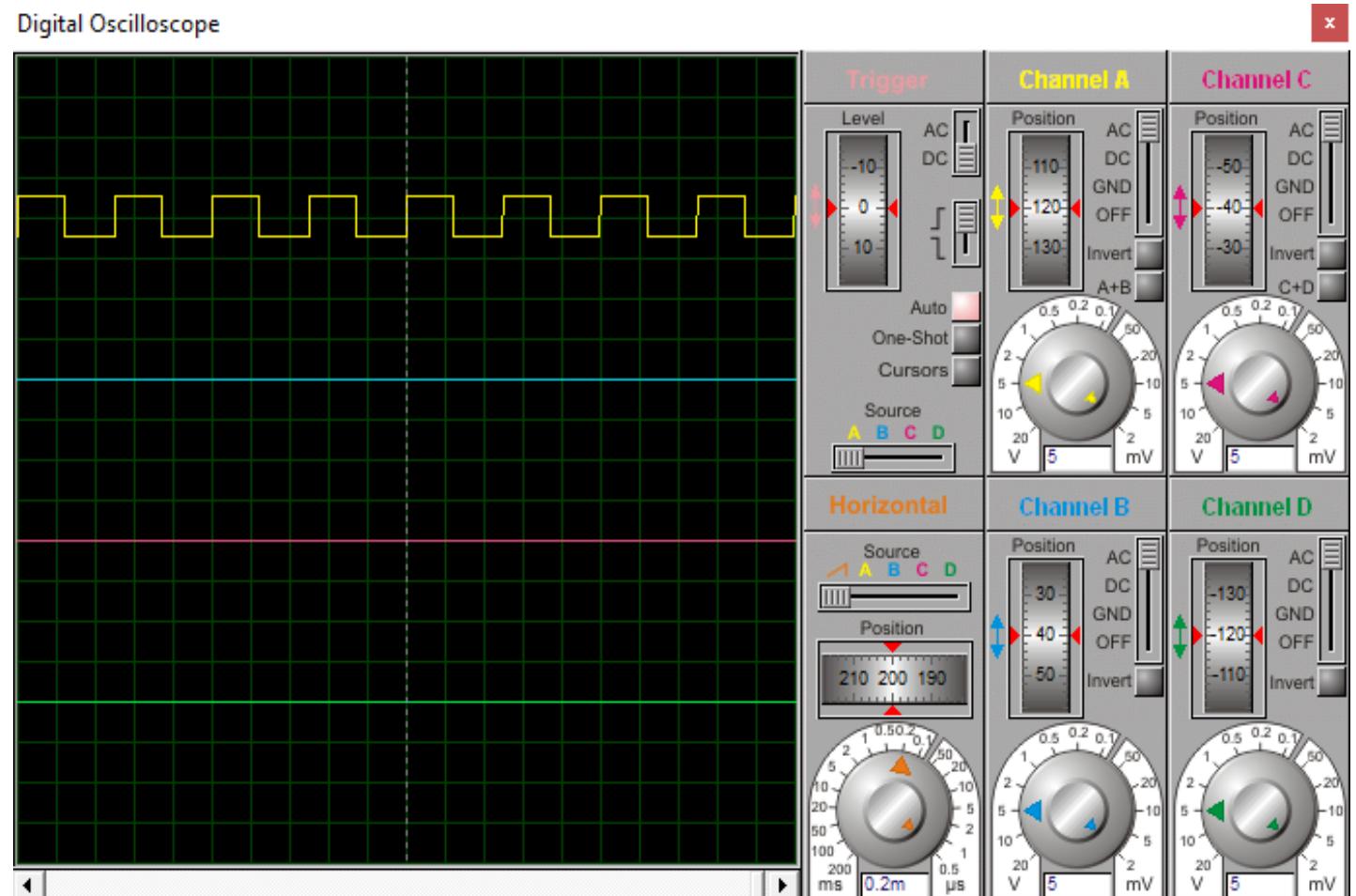
Procedure

Connect the circuit as shown in the figure.



Procedure (Cont.)

By oscilloscope display the pwm signal as show in figure.



❖ Procedure (Cont.)

Write the code in MikroC
program:

```
void main()
{
    trisc.b2 = 0; //C2 as output for PWM

    PR2 = 249 ; //load period register

    //set prescaler 1:4
    T2CKPS1_bit = 0;
    T2CKPS0_bit = 1;

    PWM1_Set_Duty(127); //set duty cycle

    /*//PWM mode, or use PWM1_Start(); // start PWM1
    CCP1M3_bit = 1;
    CCP1M2_bit = 1;
    CCP1M1_bit = 0;
    CCP1M0_bit = 0;*/

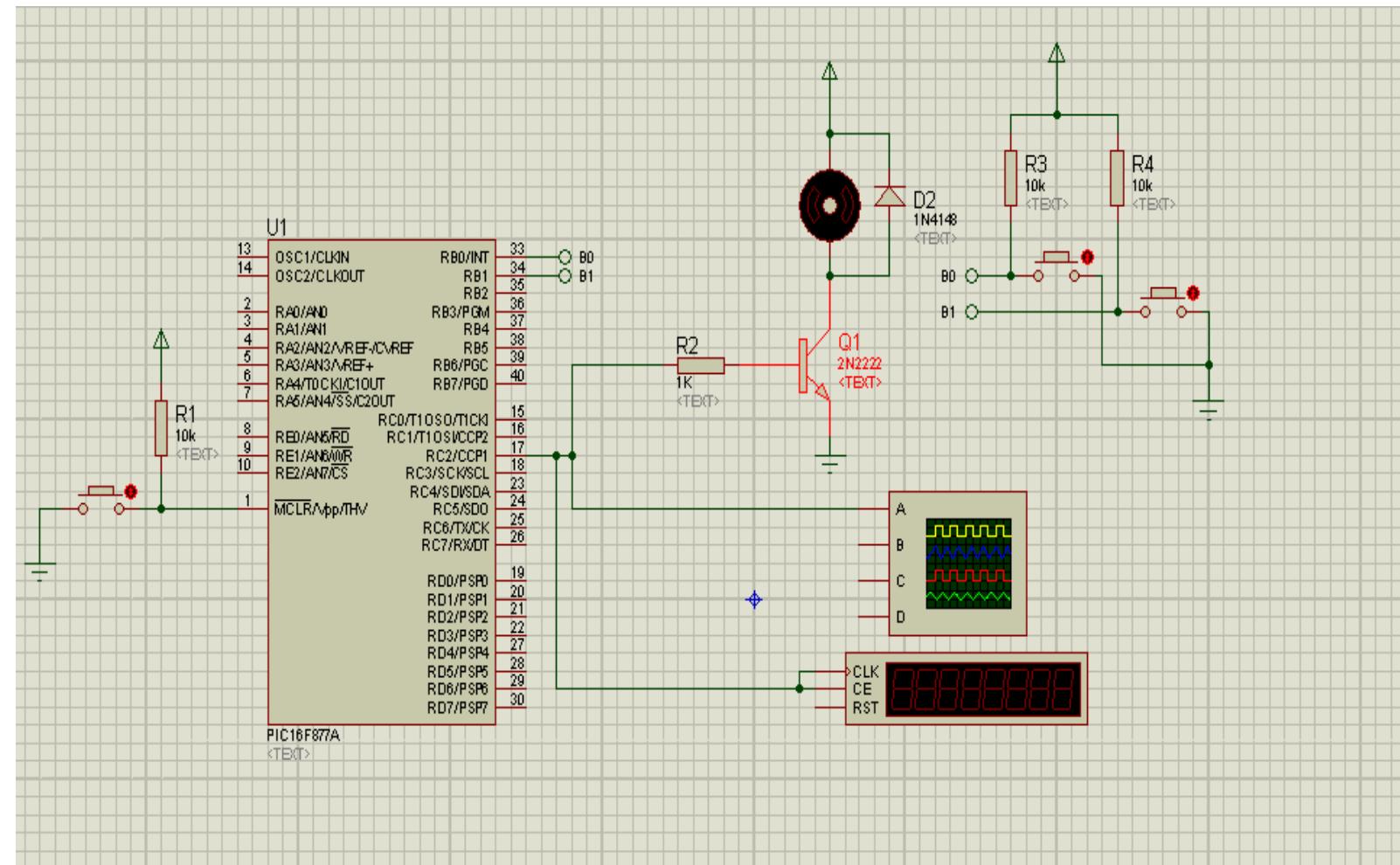
    PWM1_Start();

}
```

Procedure (Cont.)

- ❖ Control the Speed of DC Motor

Connect the circuit as shown in the figure.



Procedure (Cont.)

Write the code in MikroC program:

```
unsigned short duty_cycle=0; //stores the value of the duty cycle

void main()
{
    trisb.b0 = 1; //B0 as input for incrementing duty_cycle
    trisb.b1 = 1; //B1 as input for decrementing duty_cycle

    trisc.b2 = 0; //C2 as output for PWM

    PR2 = 124; //load period register with 124 to get 500Hz Frequency

    //set prescaler 1:16
    T2CKPS1_bit = 1;
    T2CKPS0_bit = 1;

    PWM1_Set_Duty(duty_cycle); //set duty cycle
    PWM1_Start(); // start PWM1

    while(1)
    {
        if(portb.b0 == 0) //if switch B0 has pressed
        {
            duty_cycle++; //increment duty cycle
        }
    }
}
```

Procedure (Cont.)

Write the code in MikroC
program:

```
    PWM1_Set_Duty(duty_cycle); //set duty cycle
    delay_ms(50); //delay for debounce
}
if(portb.b1 == 0) //if switch B1 has pressed
{
    duty_cycle--; //decrement duty cycle
    PWM1_Set_Duty(duty_cycle); //set duty cycle
    delay_ms(50); //delay for debounce
}
}
```

❖ Discussion

- 1- Write and design the program by using mikroC and protues to control the brightness of led.
- 2- Write and design the program by using mikroC and protues to display the value of pwm and frequency on the LCD.

EXPERIMENT 15

External Interrupt

❖ Experiment Object

The object of this experiment is to learn how to use interrupt signal with microcontroller.

❖ Theory

■ External Interrupts

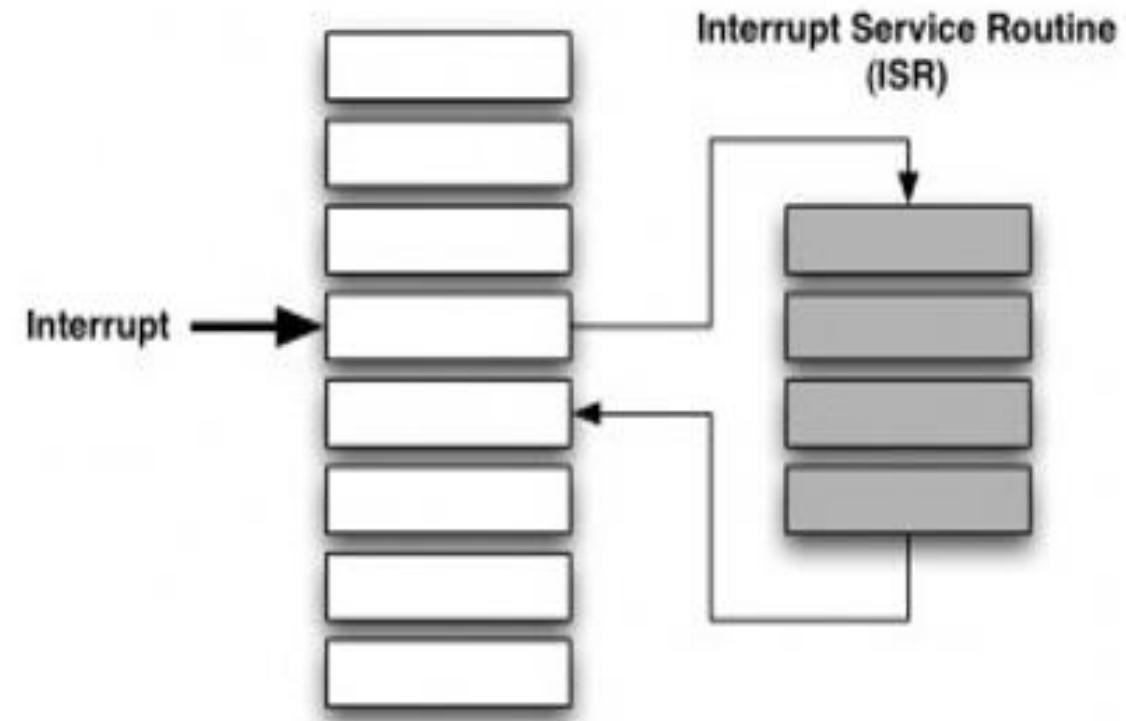
It is a case where the wizard leaves the current task to perform another task.

How?

When the wizard works in any task comes and then the wizard stops the performance of the current task and go to the interrupt service routine program to implement which caused the interrupt and then return to complete the task that stopped them.

Theory

- External Interrupts



✿ Theory

■ External Interrupts (Cont.)

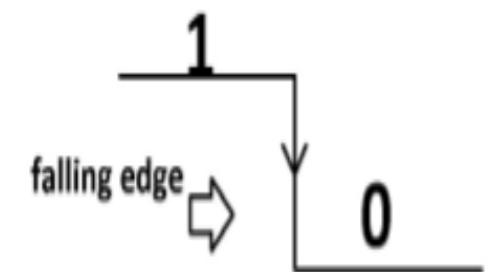
Types of interrupts:

A- External interrupt

- MCLR.
- INT,RB

B- Internal interrupts

- Timer0, Timer1, ...
- USART, ADC.



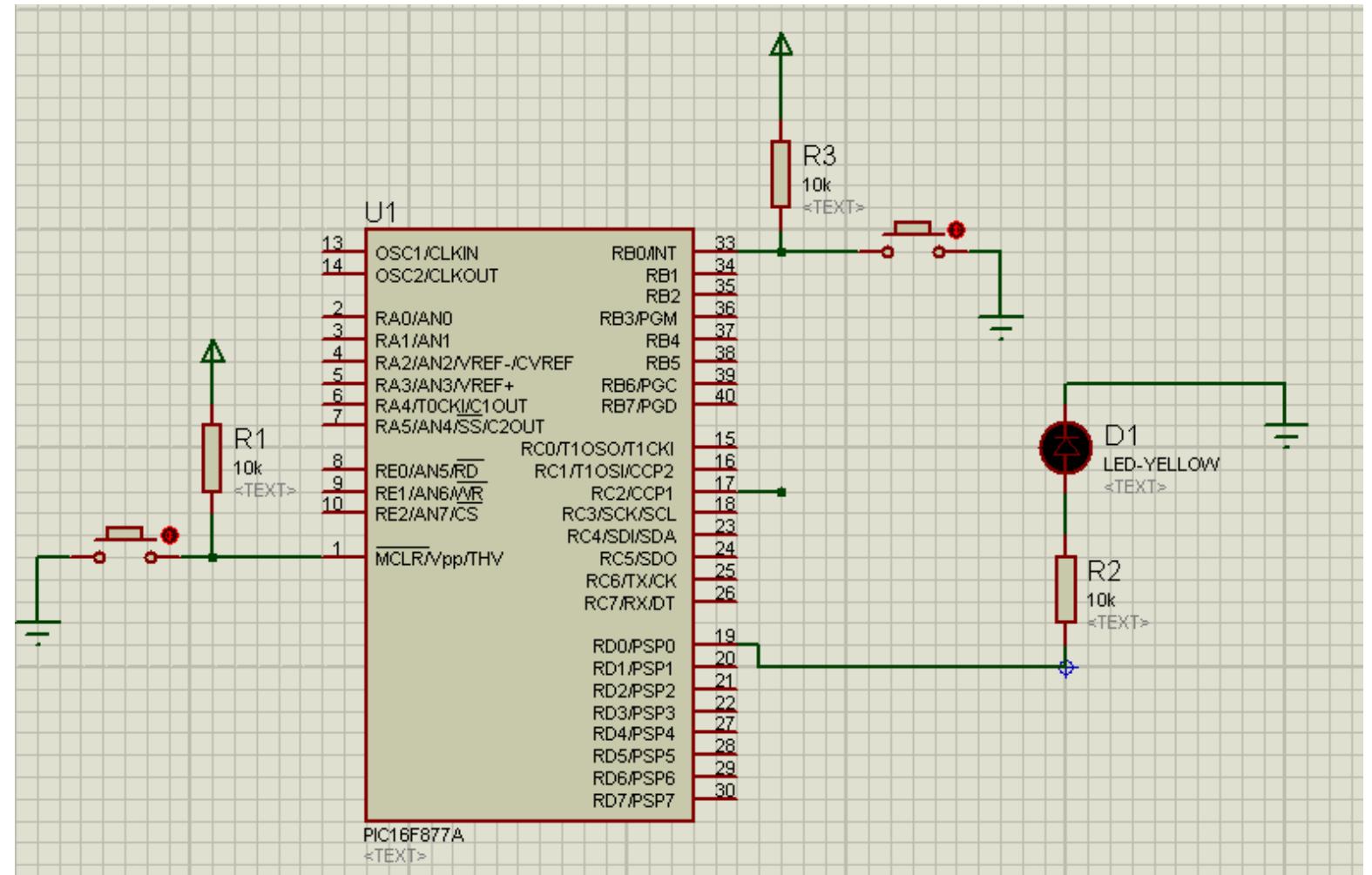
✿ The Experiment Components

- Microcontroller Pic 16F877A.
- Resistor 150 ohm.
- Push Button.
- LED.

Procedure

External Interrupt

Connect the circuit as shown in the figure.



Procedure (Cont.)

External Interrupt

Write the code in MikroC
program:

```
//interrupt initialisation
void init_interrupt()
{
    option_reg.intedg=0;      //select Interrupt on falling edge of INT0 pin

    intcon.inte=1;            //enable port INT0 external interrupt

    intcon.PEIE=1;            //Enables all unmasked peripheral interrupts
    intcon.gie=1;             //enable global interrupts
}

//INT0 isr
void interrupt()
{
    if(intcon.intf==1)        //if INT0 interrupt occurred
    {
        portd.b0 = ~portd.b0; //toggle pin D0

        intcon.intf=0;         //reset INT0 flag
    }
}

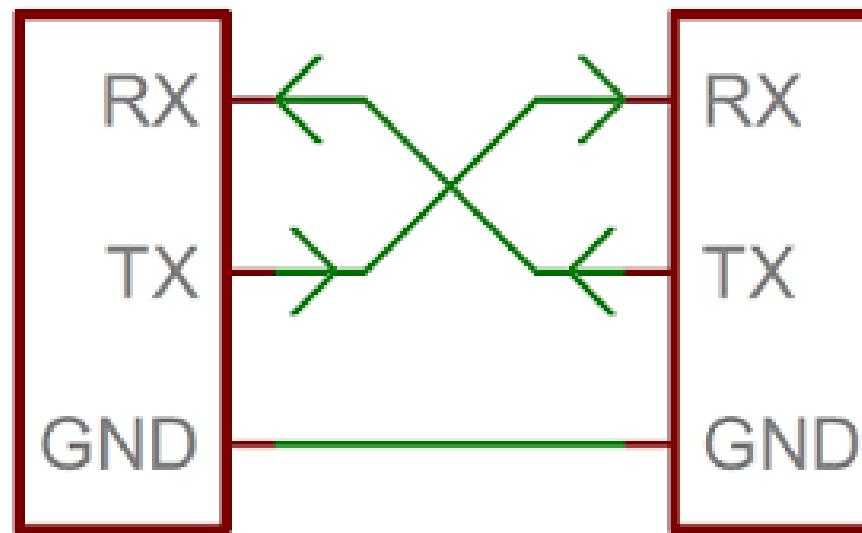
void main() {
    init_interrupt(); //interrupt initialization
    trisb.b0=1;       //INT0 pin as output
    trisd.b0=0;        portd.b0=0;
}
```

❖ Discussion

- 1- Write and design the program by using mikroC and protues to count the number of the boxes and when the number of boxes reaches to 10 the led turn on and after 1s the led return OFF.

EXPERIMENT 16

Serial Communication (UART)



Experiment Object

The object of this experiment is to learn serial communication and UART communication microcontroller.

Theory

■ Serial Communication

Serial communication is the process of transmitting data one bit at a time. In contrast, parallel communication is where data bits are sent as a whole. Parallel data transmission is faster than serial transmission but with a number of disadvantages:

- It needs more wires and therefore can be more expensive to implement
- The greater number of wires limit it to shorter transmission distances
- It is susceptible to clock skew, which limits the speed of transmission to the slowest of the links
- Crosstalk is also an issue due to the proximity of the wires

✿ Theory (Cont.)

■ The Serial Protocol

Serial communication follows a simple protocol (rules to follow) to ensure correct transmission. A serial data consists of

- Data Bits
- Synchronization Bits
- Parity Bits

Data bits ranging from 5 to 9 bits then follows. The order of data bits is from the least significant bit (LSB) to the most significant bit (MSB).

Stop bits, which is one or two logic 1's, tail the serial data, marking the end of transmission.

The baud rate, or speed of transmission, is also part of the protocol. It is important that the sender and the receiver should have the same baud rate. The standard rates are as follows:

- 1200 2400 4800 9600 19200 38400 57600 115200

❖ Theory (Cont.)

- The Serial Protocol (Cont.)

9600 is the most common and is the default baud rate for most serial terminals and simulators.

- Using UART

A microcontrollers UART device has a transmit (Tx) and receive (Rx) line. If you want to interface your micro to another device which uses serial communication, you need to connect the micro's Tx to the device's Rx and vice versa.

The number of UART device varies among microcontrollers. For example, the Arduino UNO's **ATMega 328p** has one while the Arduino Mega's ATMega 2560 has four.

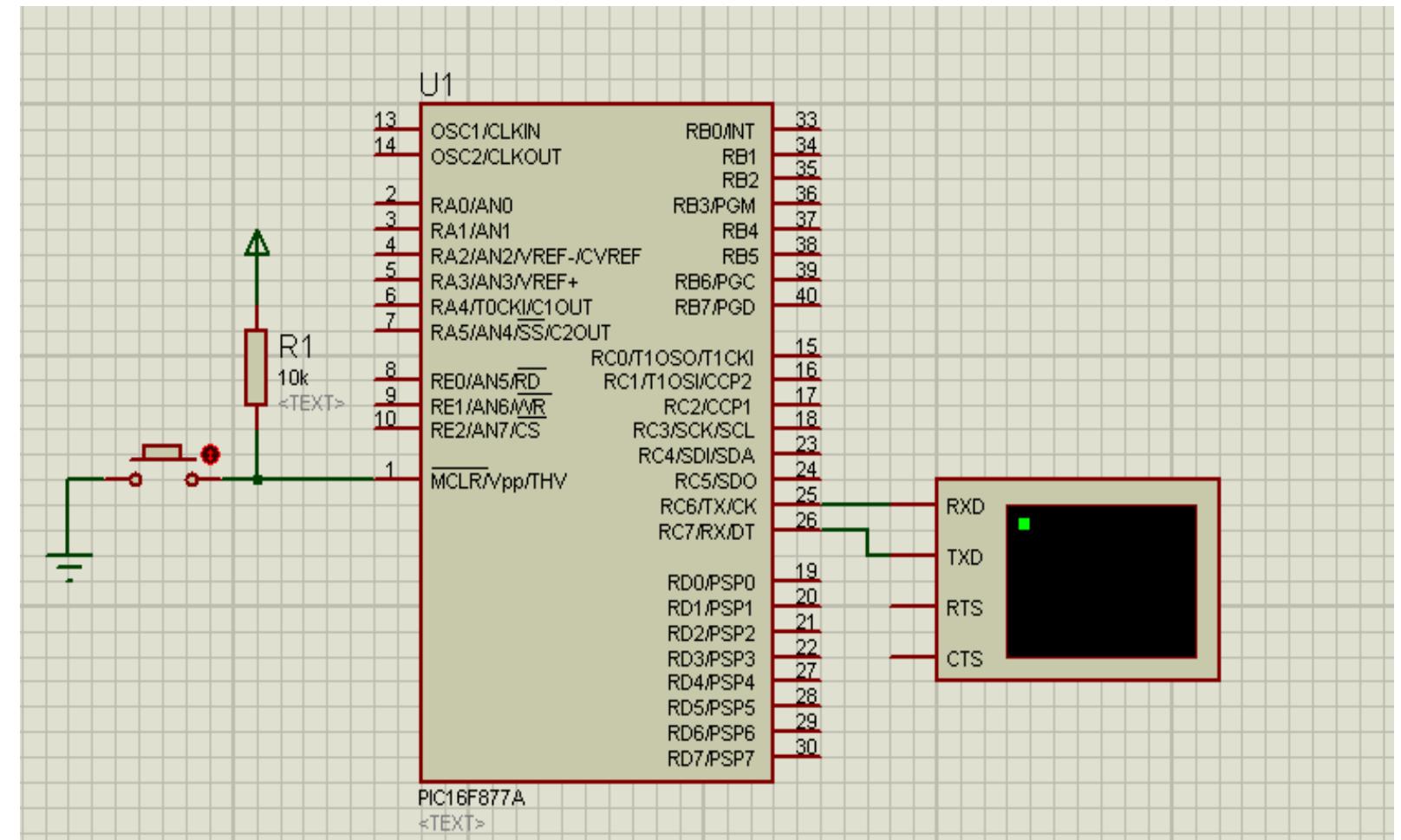
✿ The Experiment Components

- Microcontroller Pic 16F877A.
- Resistor 150 ohm.
- Push Button.
- LED.

Procedure

Send data to PC

Connect the circuit as shown in the figure.



Procedure (Cont.)

Send data to PC

Write the code in MikroC
program:

```
void main()
{
    UART1_Init(9600); // Initialize hardware UART1 and establish communication at 9600 bps

    UART1_Write('S');
    UART1_Write(10);
    UART1_Write(13);

    delay_ms(500);

    UART1_Write(0x41); // A
    UART1_Write(10);
    UART1_Write(13);

    delay_ms(500);

    uart1_write_text("Hi Guys");

}
```

Procedure (Cont.)

Receive Data

The same previous circuit connection.

Write the code in MikroC program:

```
void main()
{
    UART1_Init(9600); // Initialize hardware UART1 and establish communication at 9600 bps

    uart1_write_text("WELCOME"); //write text

    //new line
    UART1_Write(10);
    UART1_Write(13);

    trisb = 0; //port B as output
    portb = 0; //clear port B

    for(;;)
    {

        if (UART1_Data_Ready() == 1) // If data is ready
        {
            receive = UART1_Read(); //read data
            UART1_Write(receive); // and send data via UART
            //new line
            UART1_Write(10);
            UART1_Write(13);
        }
    }
}
```

❖ Procedure (Cont.)

Receive Data (Cont.)

The same previous circuit connection.

Write the code in MikroC program:

```
switch(receive) //test received data
{
    case 'a': //if is 'a'
        portb.b0 = ~portb.b0; //toggle LED B0
        receive=0; //reset received variable
        break;

    case 'b': //if is 'b'
        portb.b1 = ~portb.b1; //toggle LED B1
        receive=0; //reset received variable
        break;

    case 'c': //if is 'c'
        portb.b2 = ~portb.b2; //toggle LED B2
        receive=0; //reset received variable
        break;
}
```

Discussion

- 1- Write and design the program by using mikroC and protues to display the send data from PC to MCU on the LCD.
- 2- Write and design the program by using mikroC and protues to connect three LEDs and when send the (A) letter the LED1 ON ,(B)letter the LED 2 ON and (C) letter the LED 3 ON. When send the same letter in the second time the same led OFF.

contact

Email: elafe1888@gmail.com

LinkedIn: www.linkedin.com/in/elaf-a-saeed-97bbb6150

Twitter: <https://twitter.com/ElafASaeed1>

GitHub: <https://github.com/ElafAhmedSaeed>

YouTube: https://youtube.com/channel/UCE_RiXkyqREUdLAiZcbBqSg

SlideShare: <https://www.slideshare.net/ElafASaeed>

Slide player: <https://slideplayer.com/search/?q=Elaf+A.Saeed>

Google Scholar: <https://scholar.google.com/citations?user=VIpVZKkAAAAJ&hl=ar&gmla=AJsN-F7PIgAjWJ44Hzb18fwPqJaaUmG0XzbLdzx09>

In this book, a set of lessons related to microcontrollers and embedded systems and a set of experiments were made using the pic microcontroller. Beginners and those interested in the field of embedded systems and the use of microcontrollers can benefit from these lessons.

