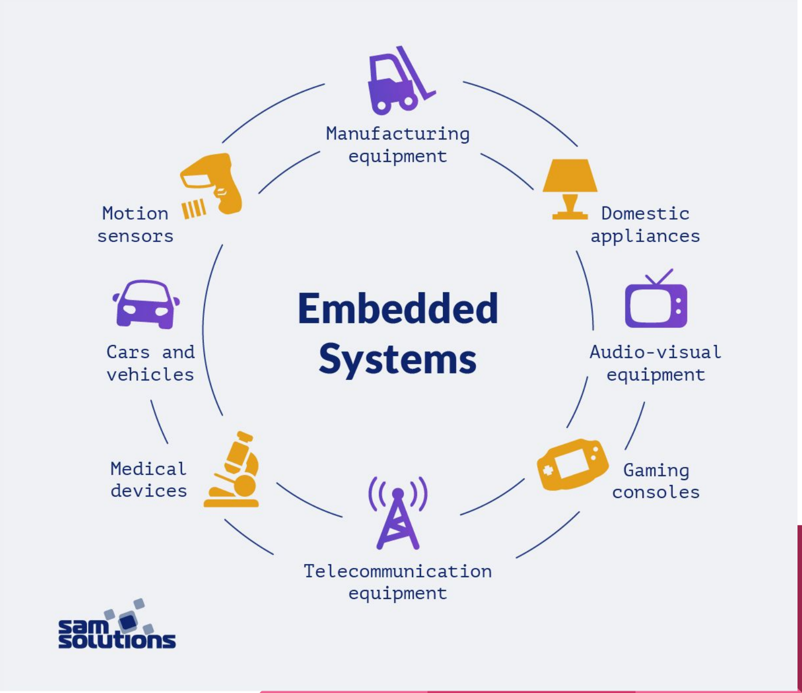
EMBEDDED SYSTEM

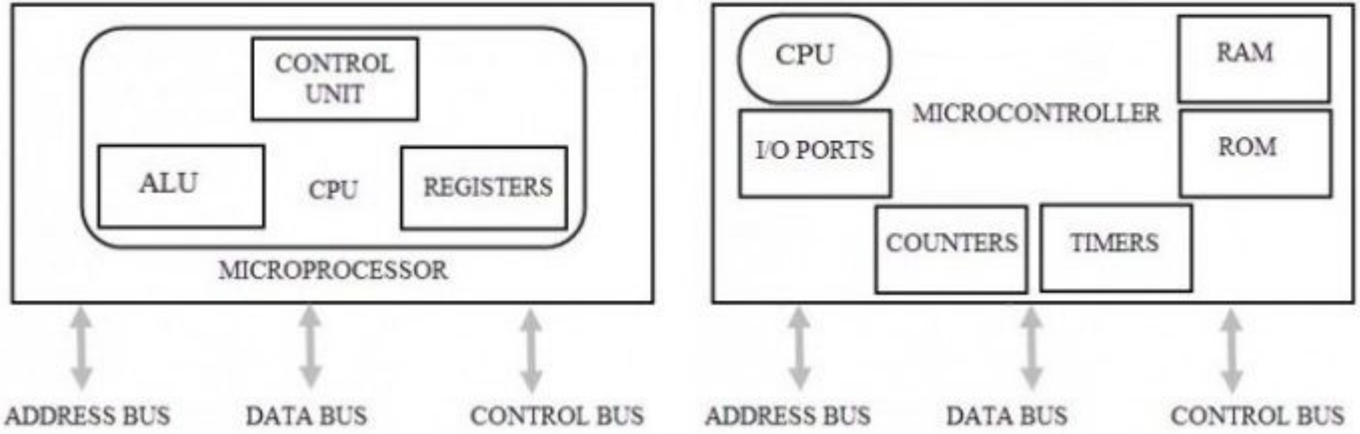
WHAT IS AN EMBEDDED SYSTEM:

- As its name suggests, Embedded means something that is attached to another thing. An embedded system can be thought of as a computer hardware system having software embedded in it.
- An embedded system can be an independent system or it can be a part of a large system.
- An embedded system is a microcontroller or microprocessor based system which is designed to perform a specific task.



Microprocessor

Microcontroller



Microprocessor vs Microcontroller by EEEPROJECT.COM

Types of Microcontrollers:

- ❖ Microcontrollers are divided into various categories based on memory, architecture, bits and instruction sets. Following is the list of their types –

Bit:

Based on bit configuration, the microcontroller is further divided into three categories.

- **8-bit microcontroller** – This type of microcontroller is used to execute arithmetic and logical operations like addition, subtraction, multiplication division, etc. For example, Intel 8031 and 8051 are 8 bits microcontroller.

- **16-bit microcontroller** – This type of microcontroller is used to perform arithmetic and logical operations where higher accuracy and performance is required. For example, Intel 8096 is a 16-bit microcontroller.
- **32-bit microcontroller** – This type of microcontroller is generally used in automatically controlled appliances like automatic operational machines, medical appliances, etc.

Applications of Embedded System:

Automotive Vehicles

Phones

Medical Equipment

Industrial Machines

Security Systems

Military Equipment

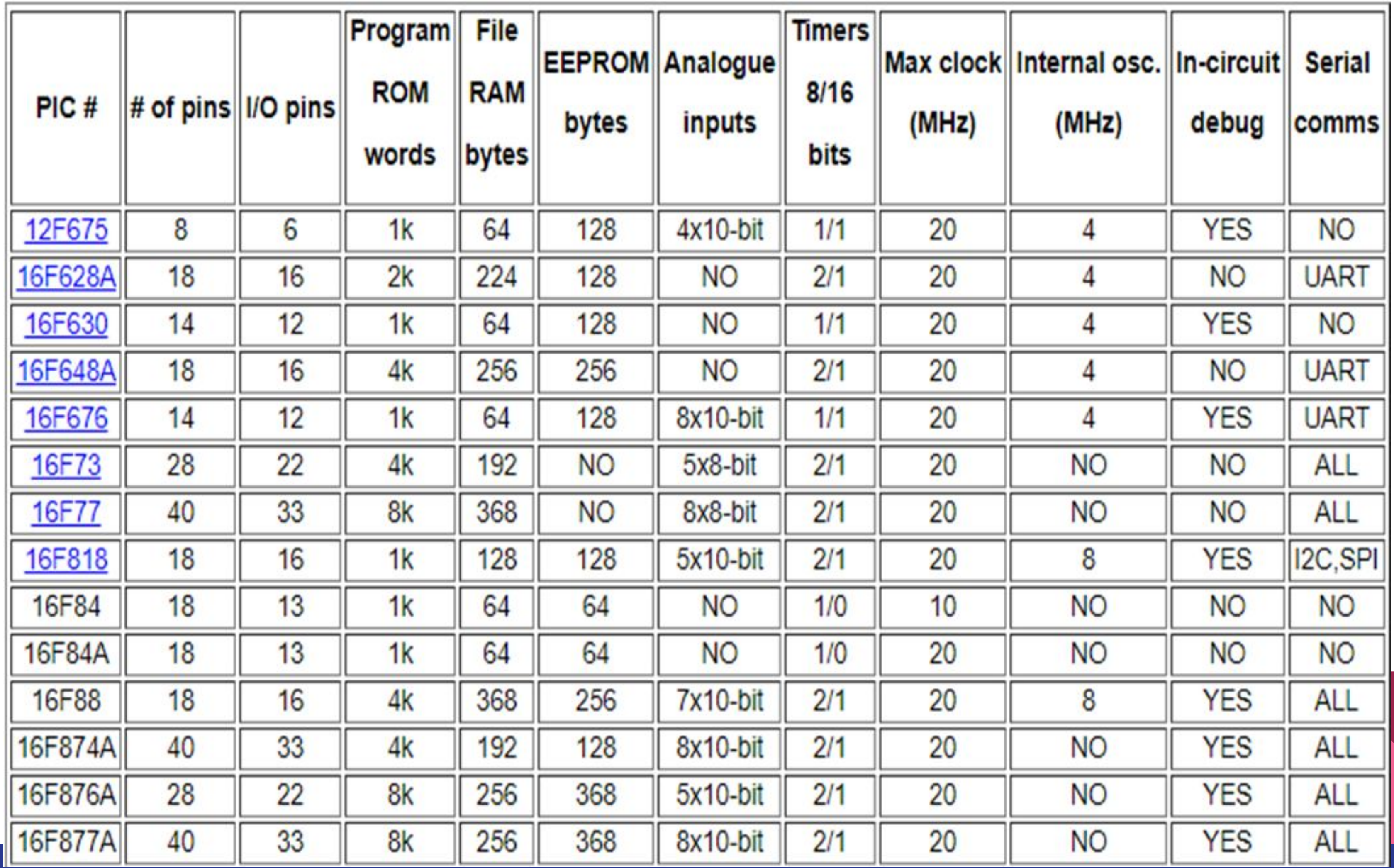
PIC18f4580 Microcontroller

PIC Introduction:

- Microchip was invented the PIC in 1993.
- PIC Microcontroller is based Harvard Architecture.
- PIC Microcontroller are available are wide between 8 bit to 32 bits.

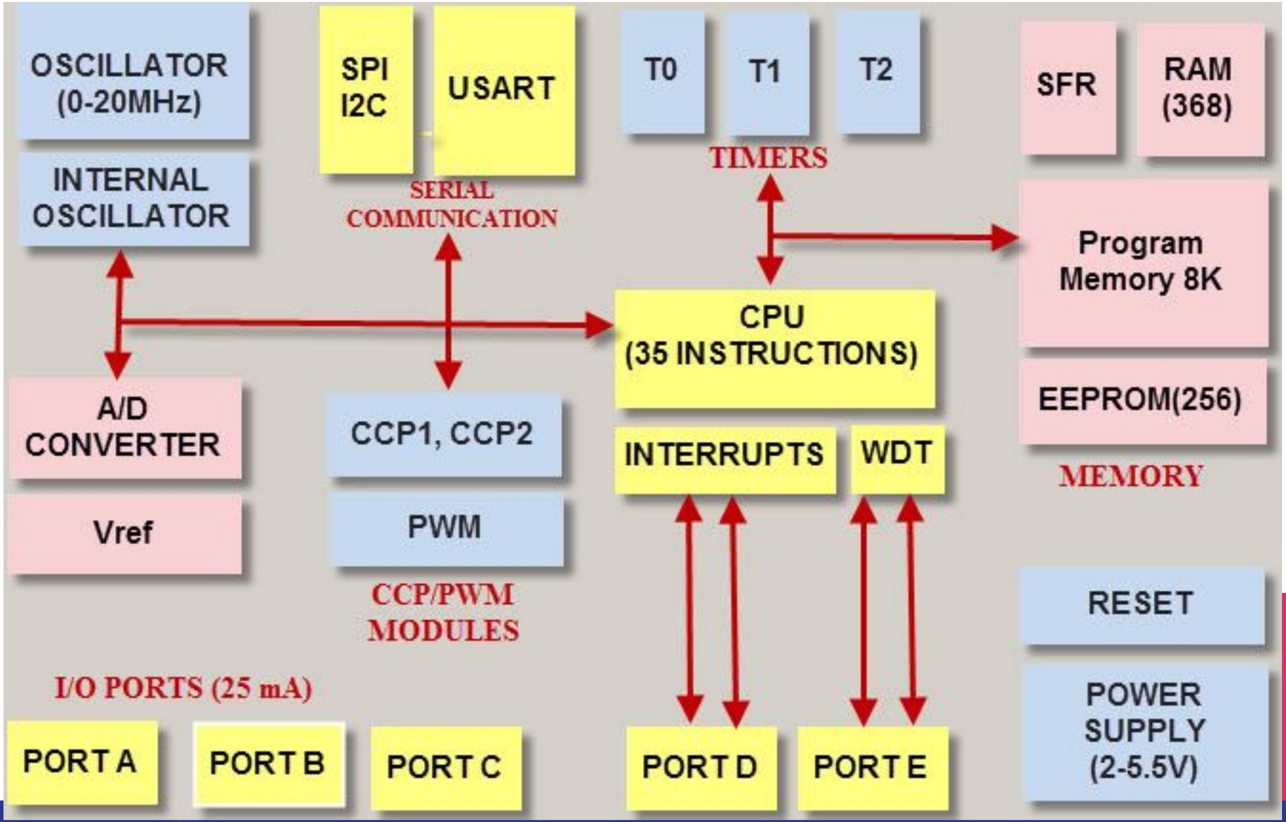
PIC Family:

- PIC Microcontrollers from Microchip Company are divided into 5 large families. Each family has a variety of components that provide built-in special features:
  - The first family, PIC10 (10FXXX) - is called Low End.
  - The second family, PIC12 (PIC12FXXX) – is called Mid-Range.
  - The third family is PIC16 (16FXXX).
  - The fourth family is PIC 17/18(18FXXX)
  - The fivth family is PIC32 (32FXXX)



PIC #	# of pins	I/O pins	Program ROM words	File RAM bytes	EEPROM bytes	Analogue inputs	Timers 8/16 bits	Max clock (MHz)	Internal osc. (MHz)	In-circuit debug	Serial comms
12F675	8	6	1k	64	128	4x10-bit	1/1	20	4	YES	NO
16F628A	18	16	2k	224	128	NO	2/1	20	4	NO	UART
16F630	14	12	1k	64	128	NO	1/1	20	4	YES	NO
16F648A	18	16	4k	256	256	NO	2/1	20	4	NO	UART
16F676	14	12	1k	64	128	8x10-bit	1/1	20	4	YES	UART
16F73	28	22	4k	192	NO	5x8-bit	2/1	20	NO	NO	ALL
16F77	40	33	8k	368	NO	8x8-bit	2/1	20	NO	NO	ALL
16F818	18	16	1k	128	128	5x10-bit	2/1	20	8	YES	I2C,SPI
16F84	18	13	1k	64	64	NO	1/0	10	NO	NO	NO
16F84A	18	13	1k	64	64	NO	1/0	20	NO	NO	NO
16F88	18	16	4k	368	256	7x10-bit	2/1	20	8	YES	ALL
16F874A	40	33	4k	192	128	8x10-bit	2/1	20	NO	YES	ALL
16F876A	28	22	8k	256	368	5x10-bit	2/1	20	NO	YES	ALL
16F877A	40	33	8k	256	368	8x10-bit	2/1	20	NO	YES	ALL

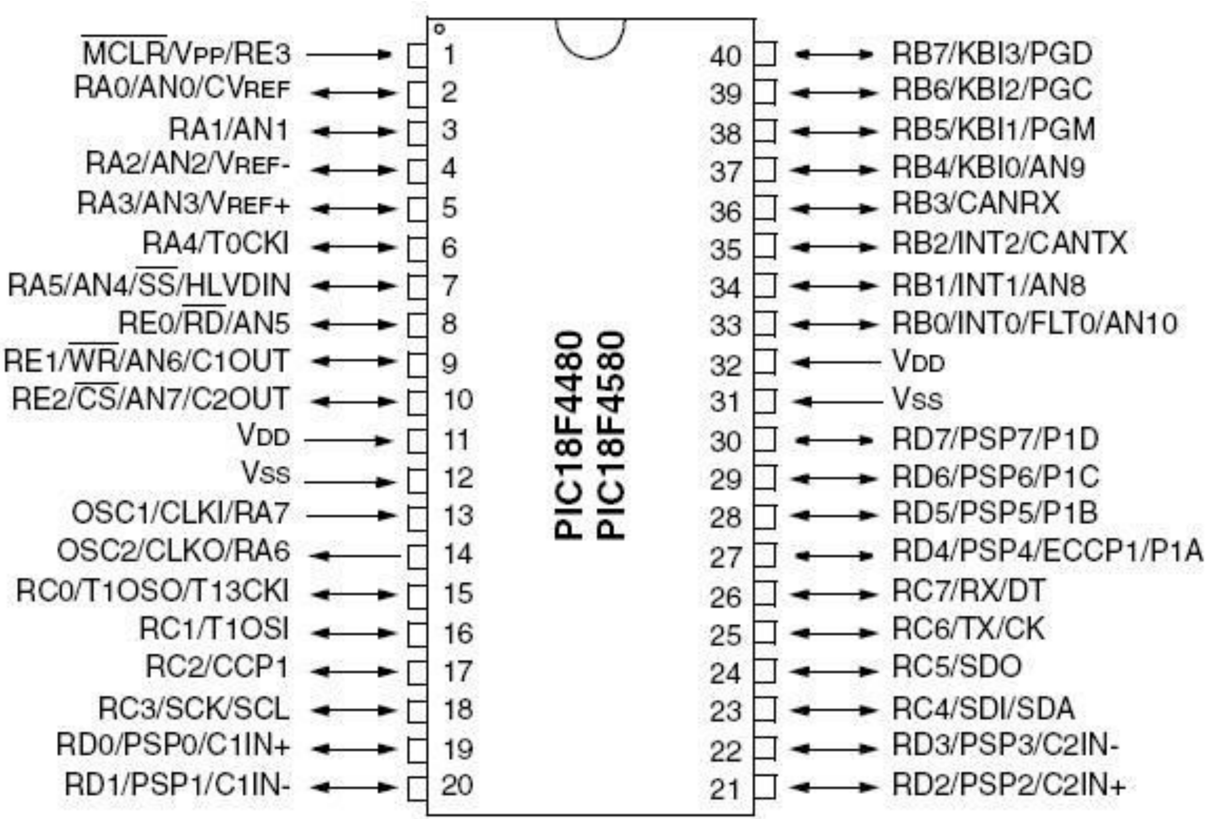
PIC18f4580 Block Diagram:



ALU(Arithmetic Logic Unit):

- ALU perform arithmetic and logic operation.
- ALU also manipulate the data.
- Individual bit of register can be set, reset, cleared, complemented, used in logical computation.

Pin Diagram


MCLR/VPP/RE3
RA0/AN0/CVREF
RA1/AN1
RA2/AN2/VREF-
RA3/AN3/VREF+
RA4/T0CKI
RA5/AN4/SS/HLVDIN
RE0/RD/AN5
RE1/WR/AN6/C1OUT
RE2/CS/AN7/C2OUT
VDD
VSS
OSC1/CLKI/RA7
OSC2/CLKO/RA6
RC0/T1OSO/T13CKI
RC1/T1OSI
RC2/CCP1
RC3/SCK/SCL
RD0/PSP0/C1IN+
RD1/PSP1/C1IN-
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
PIC18F4480
PIC18F4580
40
39
38
37
36
35
34
33
32
31
30
29
28
27
26
25
24
23
22
21
RB7/KBI3/PGD
RB6/KBI2/PGC
RB5/KBI1/PGM
RB4/KBI0/AN9
RB3/CANRX
RB2/INT2/CANTX
RB1/INT1/AN8
RB0/INT0/FLT0/AN10
VDD
VSS
RD7/PSP7/P1D
RD6/PSP6/P1C
RD5/PSP5/P1B
RD4/PSP4/ECCP1/P1A
RC7/RX/DT
RC6/TX/CK
RC5/SDO
RC4/SDI/SDA
RD3/PSP3/C2IN-
RD2/PSP2/C2IN+

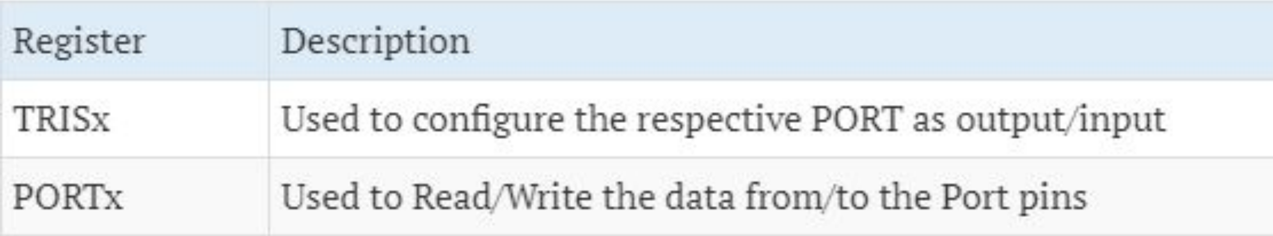
PIC18f Features:

- 40 Pins low Power uc.
- Flash Program memory: 32Kb .
- EEPROM Data Memory: 256 bytes.
- SRAM Data Memory: 1536 bytes.
- Input & Output pins : 33.
- Timers: One 8 bit / Three 16 bit.
- A/D Converter : 10 bit Eleven Channels.
- CAN: CAN 2.0B Specification with transmit & Receive Buffers.
- PWM: 10-bit two modules.
- MSSP: SPI & I2C Master & Slave support.
- Enhanced USART: Addressable with RS-485,RS232 & LIN Support.
- External Oscillator.: upto 40MHz.
- Internal Oscillator: 4MHz.

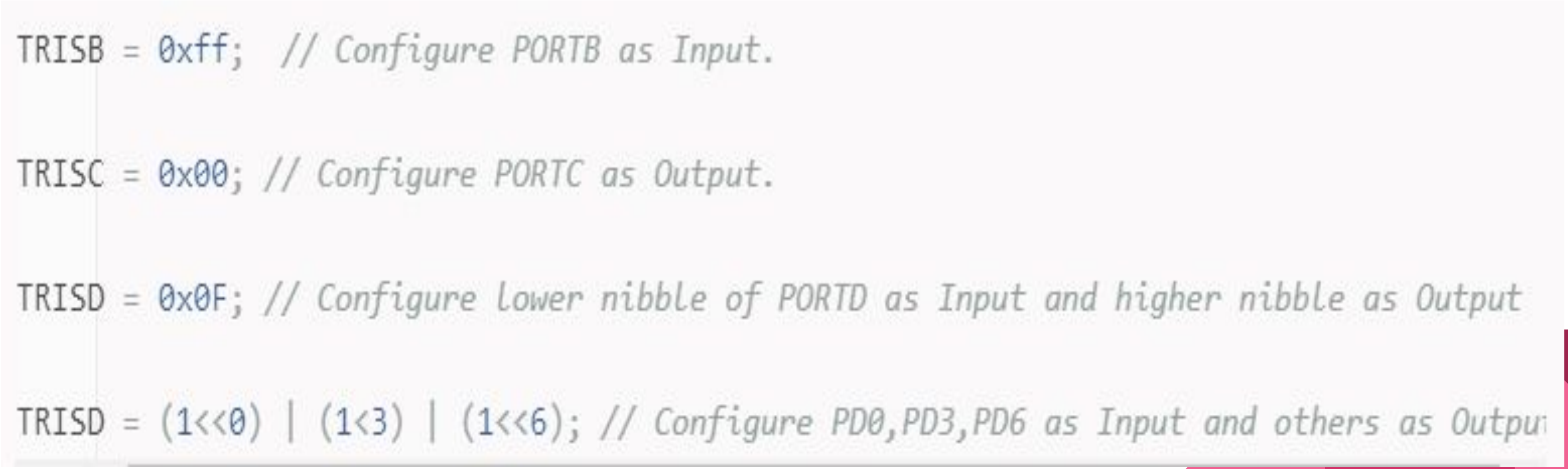
HOW TO PROGRAM THE INPUT AND OUTPUT PORTS

- As we have studied 5 input and output ports namely PORTA, PORTB, PORTC, PORT D and PORT E which can be digital as well as analog.
- We will configure them according to our requirements. But in case of analog mode, the pins or the ports can only act as inputs. There is a built in A to D converter which is used in such cases. Multiplexer circuits are also used.
- But in digital mode, there is no restriction. We can configure the ports as output or as input. This is done through programming. For PIC the preferable compiler is mikroC pro which can be downloaded from their website.
- There is a register named as ‘TRIS’ which controls the direction of ports. For different ports there are different registers such as TRISA, TRISB etc.
- If we set a bit of the TRIS register to 0, the corresponding port bit will act as the digital output.
- If we set a bit of the TRIS register to 1, the corresponding port bit will act as the digital input.
- For example to set the whole portb to output we can write the program statement as:

```
TRISB=0;
```



Register	Description
TRISx	Used to configure the respective PORT as output/input
PORTx	Used to Read/Write the data from/to the Port pins

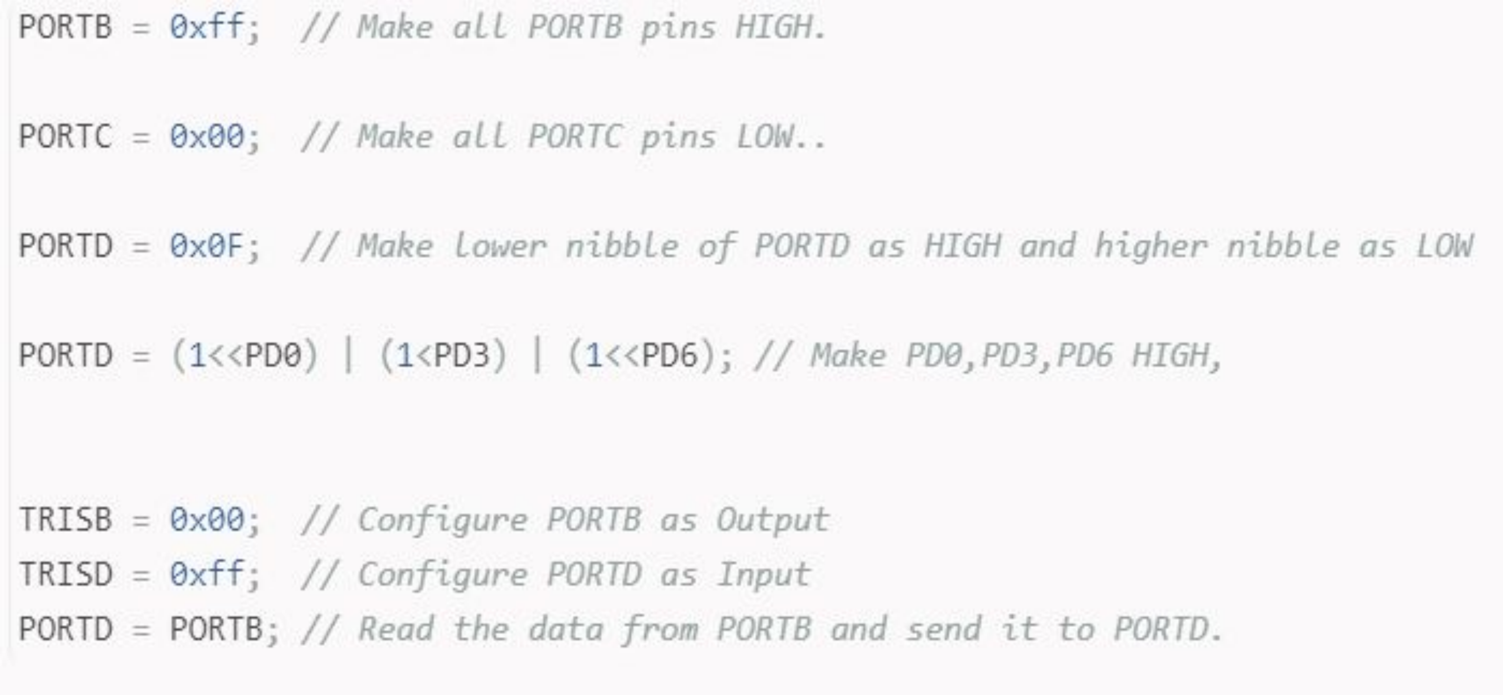


```
TRISB = 0xff;  // Configure PORTB as Input.

TRISC = 0x00; // Configure PORTC as Output.

TRISD = 0x0F; // Configure lower nibble of PORTD as Input and higher nibble as Output

TRISD = (1<<0) | (1<3) | (1<<6); // Configure PD0,PD3,PD6 as Input and others as Outpu
```

PORTx:

This register is used to read/write the data from/to port pins. Writing 1's to PORTx will make the corresponding PORTx pins as HIGH. Similarly writing 0's to PORTx will make the corresponding PORTx pins as LOW.

Before reading/writing the data, the port pins should be configured as Input Output.



```
PORTB = 0xff;  // Make all PORTB pins HIGH.

PORTC = 0x00;  // Make all PORTC pins LOW..

PORTD = 0x0F;  // Make lower nibble of PORTD as HIGH and higher nibble as LOW

PORTD = (1<<PD0) | (1<PD3) | (1<<PD6); // Make PD0,PD3,PD6 HIGH,


TRISB = 0x00;  // Configure PORTB as Output
TRISD = 0xff;  // Configure PORTD as Input
PORTD = PORTB; // Read the data from PORTB and send it to PORTD.
```

Now the port will act as the output port and we can send any value on the output such as

```
PORTB=0XFF;
```

FF represents all 1’s in binary i.e. FF=11111111, now all the pins of port b are high. If we connect LEDs at all the pins then they will all start glowing in this condition.

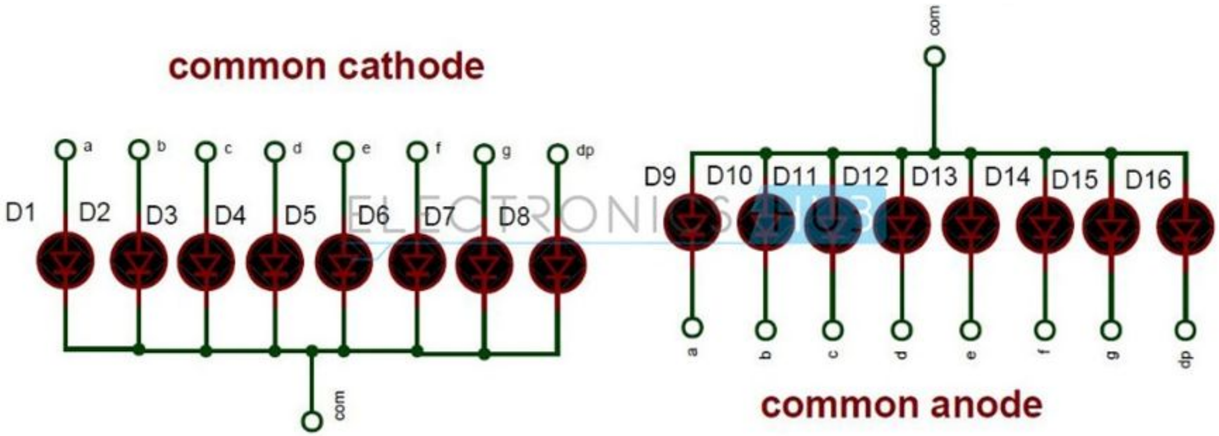
- If we want to negate the values of the port b we can use the statement:

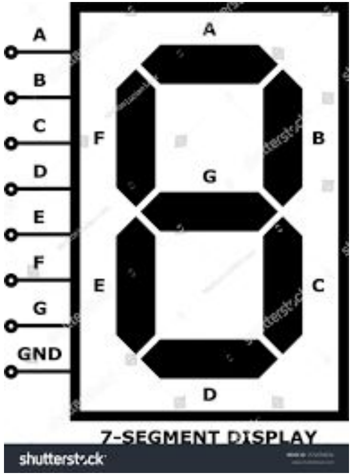
```
PORTB=~PORTB;
```

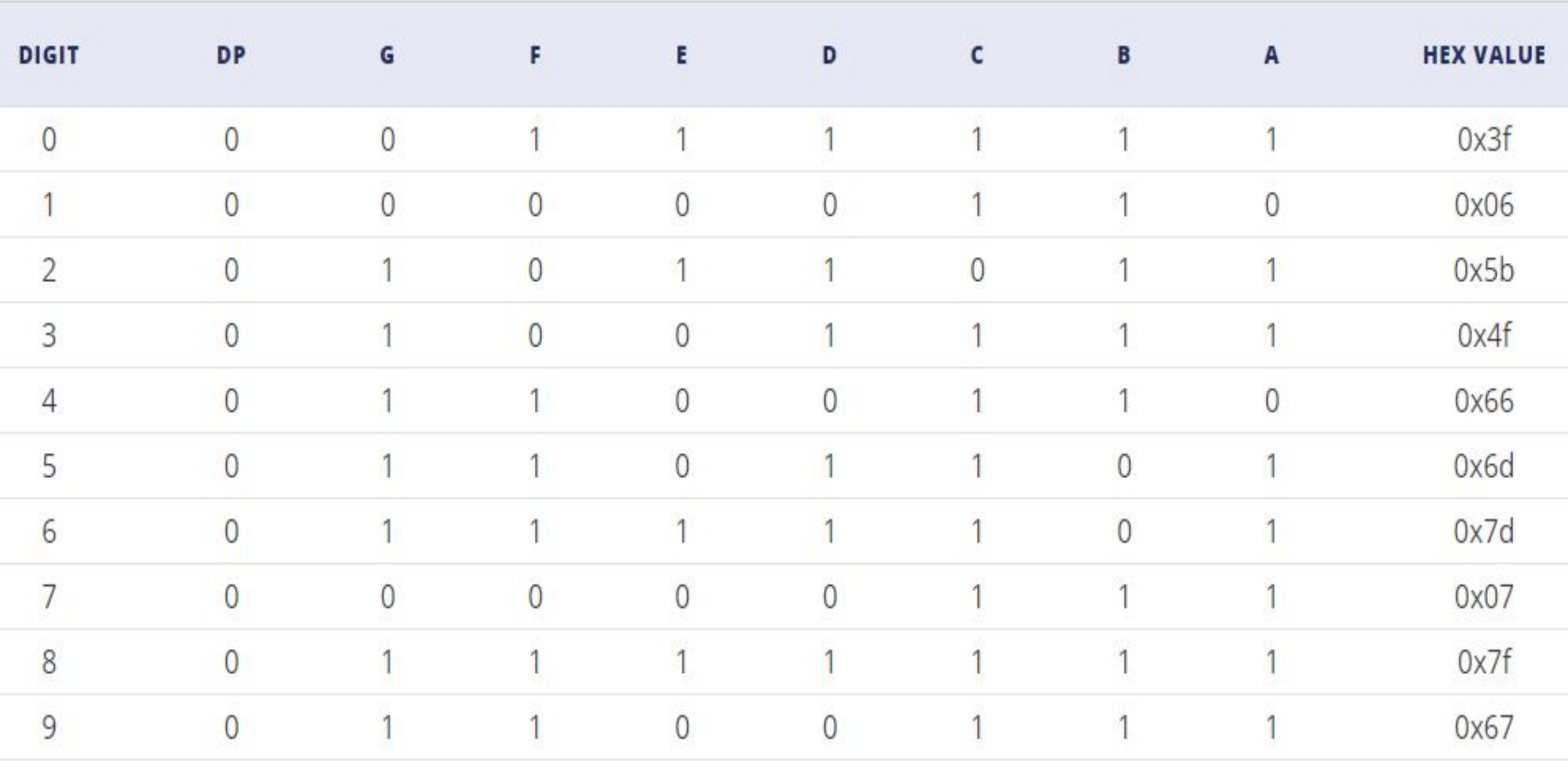
Now all the pins of the port b will be low.

PIC Interfacing with Seven Segment:

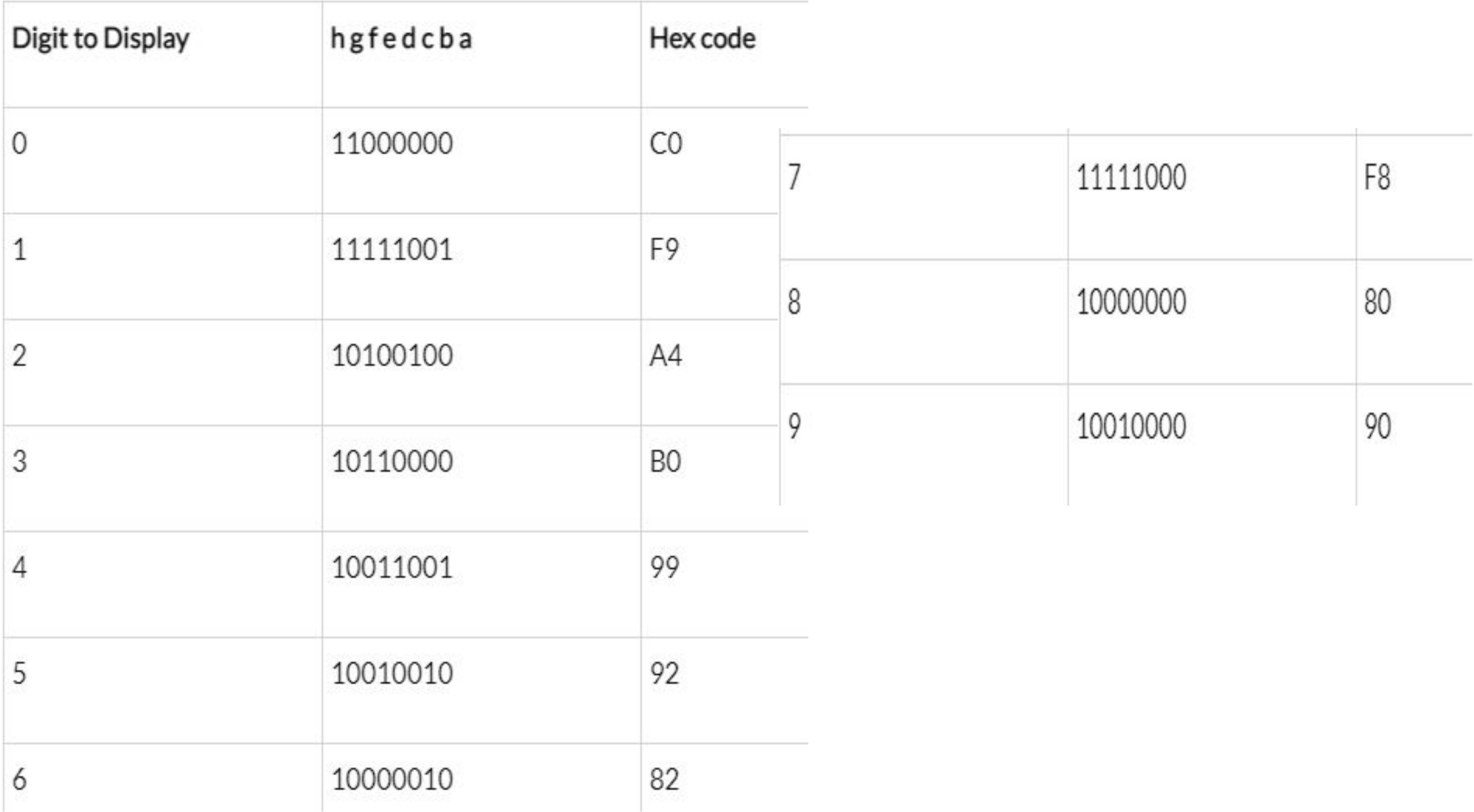
- Seven segment displays are used to indicate numerical information.
- Seven segments display can display digits from 0 to 9 and even we can display few characters like A, b, C, H, E, e, F, etc.
- Seven segment displays internally consist of 8 LEDs. In these LEDs, 7 LEDs are used to indicate the digits 0 to 9 and single LED is used for indicating decimal point.
- Generally, seven segments are two types, one is common cathode and the other is common anode.



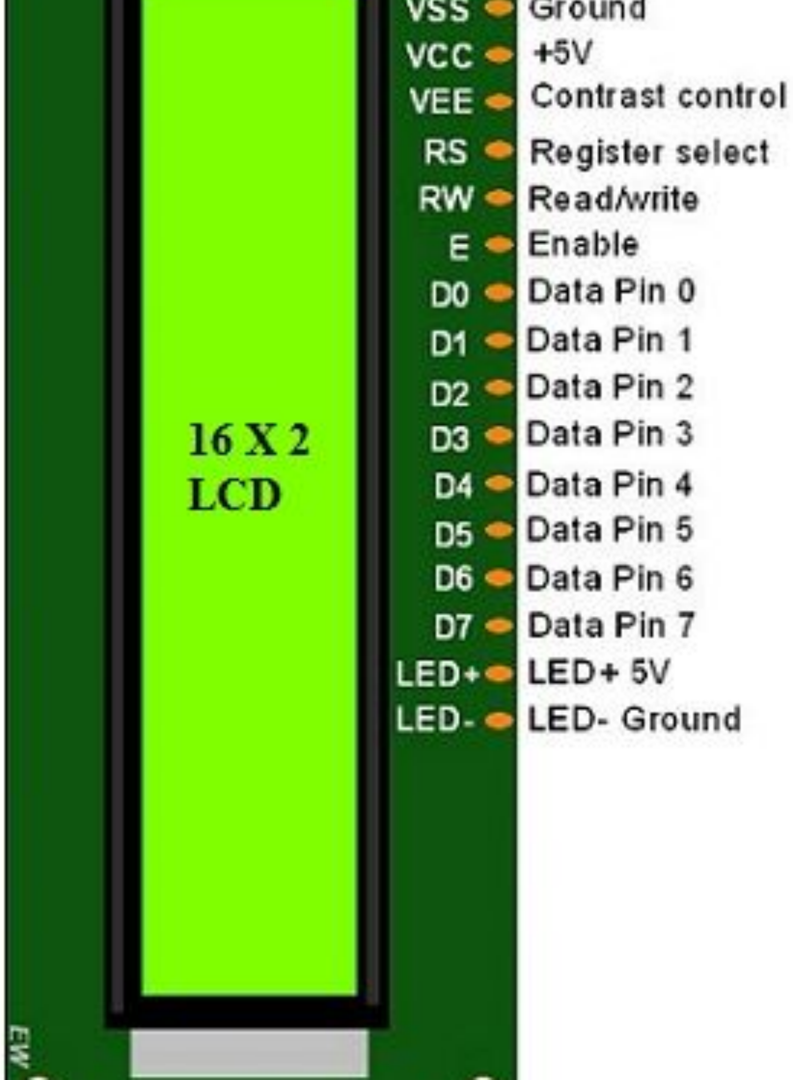




DIGIT	DP	G	F	E	D	C	B	A	HEX VALUE
0	0	0	1	1	1	1	1	1	0x3f
1	0	0	0	0	0	1	1	0	0x06
2	0	1	0	1	1	0	1	1	0x5b
3	0	1	0	0	1	1	1	1	0x4f
4	0	1	1	0	0	1	1	0	0x66
5	0	1	1	0	1	1	0	1	0x6d
6	0	1	1	1	1	1	0	1	0x7d
7	0	0	0	0	0	1	1	1	0x07
8	0	1	1	1	1	1	1	1	0x7f
9	0	1	1	0	0	1	1	1	0x67



Digit to Display	h g f e d c b a	Hex code
0	11000000	C0
1	11111001	F9
2	10100100	A4
3	10110000	B0
4	10011001	99
5	10010010	92
6	10000010	82
7	11111000	F8
8	10000000	80
9	10010000	90


VSS Ground
VCC +5V
VEE Contrast control
RS Register select
RW Read/write
E Enable
D0 Data Pin 0
D1 Data Pin 1
D2 Data Pin 2
D3 Data Pin 3
D4 Data Pin 4
D5 Data Pin 5
D6 Data Pin 6
D7 Data Pin 7
LED+ LED+ 5V
LED- LED- Ground
16 X 2
LCD

Interfacing 16×2 LCD with 8051

16×2 Liquid Crystal Display which will display the 32 characters at a time in two rows (16 characters in one row). Each character in the display is of size 5×7-pixel matrix.

Simple steps for displaying a character or data

E=1; enable pin should be high

RS=1; Register select should be high

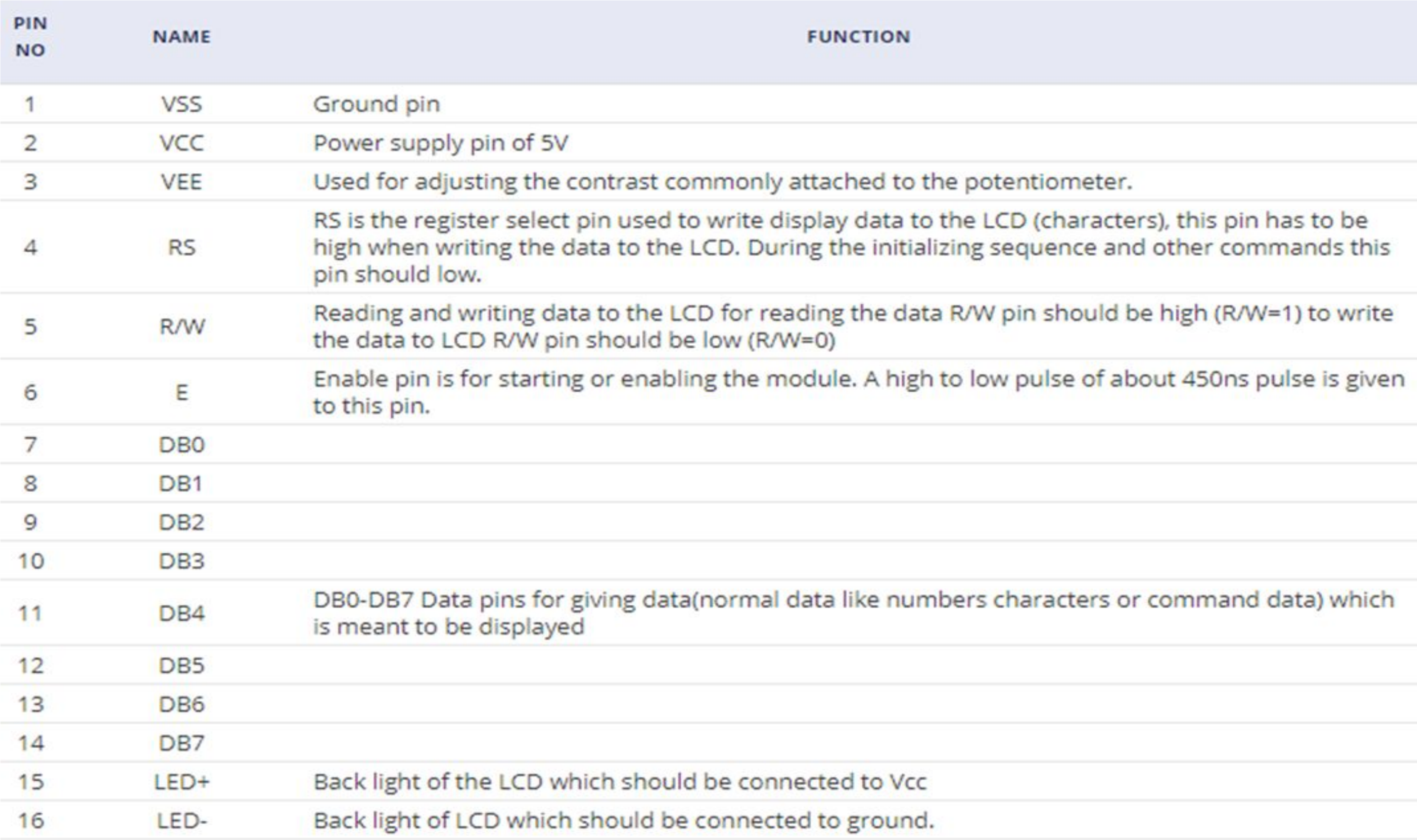
R/W=0; Read/Write pin should be low.

Send a command to the LCD just follows these steps.

E=1; enable pin should be high

RS=0; Register select should be low

R/W=0; Read/Write pin should be low.



PIN NO	NAME	FUNCTION
1	VSS	Ground pin
2	VCC	Power supply pin of 5V
3	VEE	Used for adjusting the contrast commonly attached to the potentiometer.
4	RS	RS is the register select pin used to write display data to the LCD (characters), this pin has to be high when writing the data to the LCD. During the initializing sequence and other commands this pin should low.
5	R/W	Reading and writing data to the LCD for reading the data R/W pin should be high (R/W=1) to write the data to LCD R/W pin should be low (R/W=0)
6	E	Enable pin is for starting or enabling the module. A high to low pulse of about 450ns pulse is given to this pin.
7	DB0	
8	DB1	
9	DB2	
10	DB3	
11	DB4	DB0-DB7 Data pins for giving data(normal data like numbers characters or command data) which is meant to be displayed
12	DB5	
13	DB6	
14	DB7	
15	LED+	Back light of the LCD which should be connected to Vcc
16	LED-	Back light of LCD which should be connected to ground.

For displaying a character, you should enable the enable pin (pin 6) by giving a pulse of 450ns, after enabling the pin6 you should select the register select pin (pin4) in write mode. To select the register, select pin in write mode you have to make this pin high (RS=1), after selecting the register select you have to configure the R/W to write mode that is R/W should be low (R/W=0).

Simple steps for displaying a character or data:

E=1; enable pin should be high

RS=1; Register select should be high

R/W=0; Read/Write pin should be low.

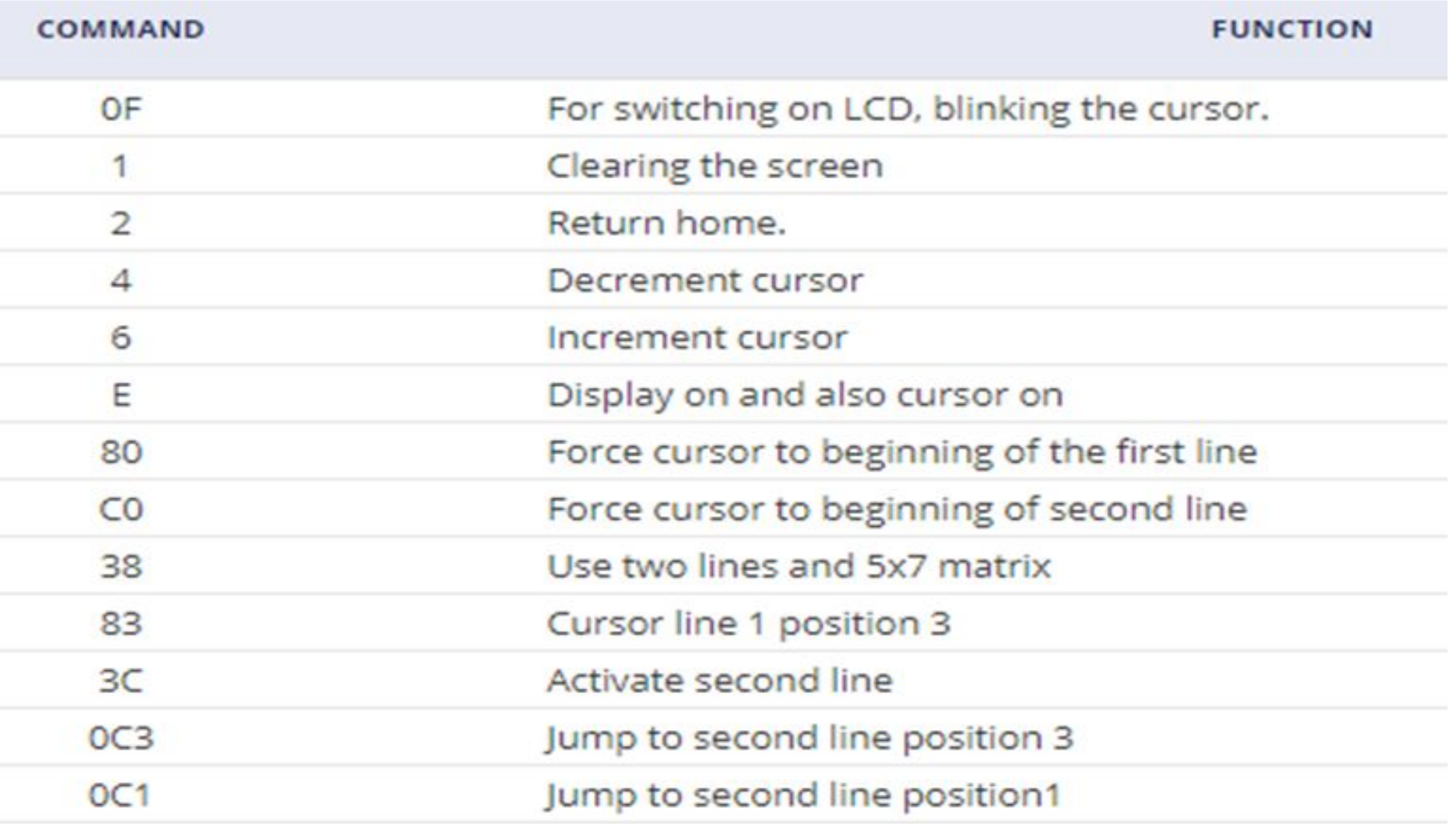
Send a command to the LCD just follows these steps:

E=1; enable pin should be high

RS=0; Register select should be low

R/W=0; Read/Write pin should be low.

Commands List



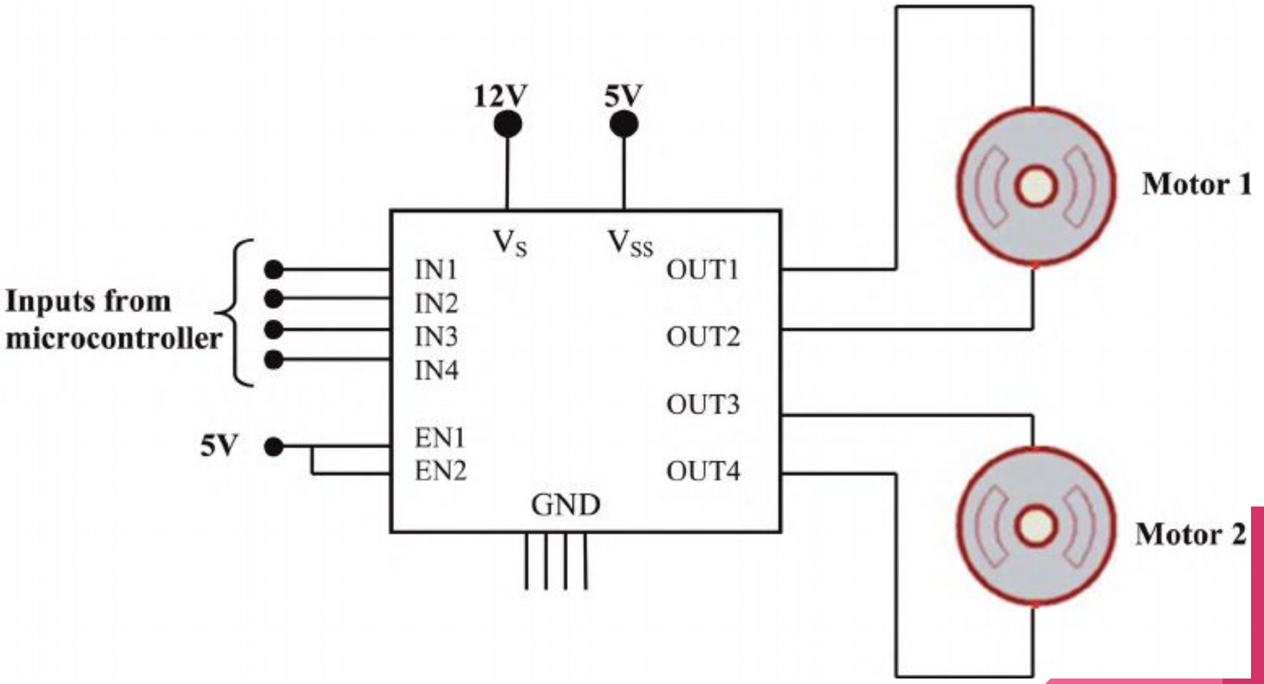
COMMAND	FUNCTION
0F	For switching on LCD, blinking the cursor.
1	Clearing the screen
2	Return home.
4	Decrement cursor
6	Increment cursor
E	Display on and also cursor on
80	Force cursor to beginning of the first line
C0	Force cursor to beginning of second line
38	Use two lines and 5x7 matrix
83	Cursor line 1 position 3
3C	Activate second line
0C3	Jump to second line position 3
0C1	Jump to second line position1

DC Motor Interfacing with PIC:

- A DC motor in simple words is a device that converts electrical energy (direct current system) into mechanical energy. It is of vital importance for the industry today.
- The maximum output current of the microcontroller pin is 15mA at 0 to 5V. But the power requirements of most DC motors are out of reach of the microcontroller and even the back emf (electromotive force) which is produced by the motor may damage the microcontroller. Hence it is not good to interface the DC motor directly to the controller. So use a motor driver circuit in between of DC motor and controller.

- Here, we are using an L293D motor driver IC to drive DC motors. Using this IC, we can drive 2 DC motors at a time. For this IC motor supply is variable 4.5 to 36V and it provides a maximum current of 600mA.
- The maximum current capacity of L293 is 600mA/channel. So do not use a motor that consumes more than that.
- The supply voltage range of L293 is between 4.5 and 36V DC. So you can use a motor falling in that range.
- Mostly in Robotic application, we will use DC gear Motor. So the same logic is used for that Gear motor also.

DC Motor Driver:



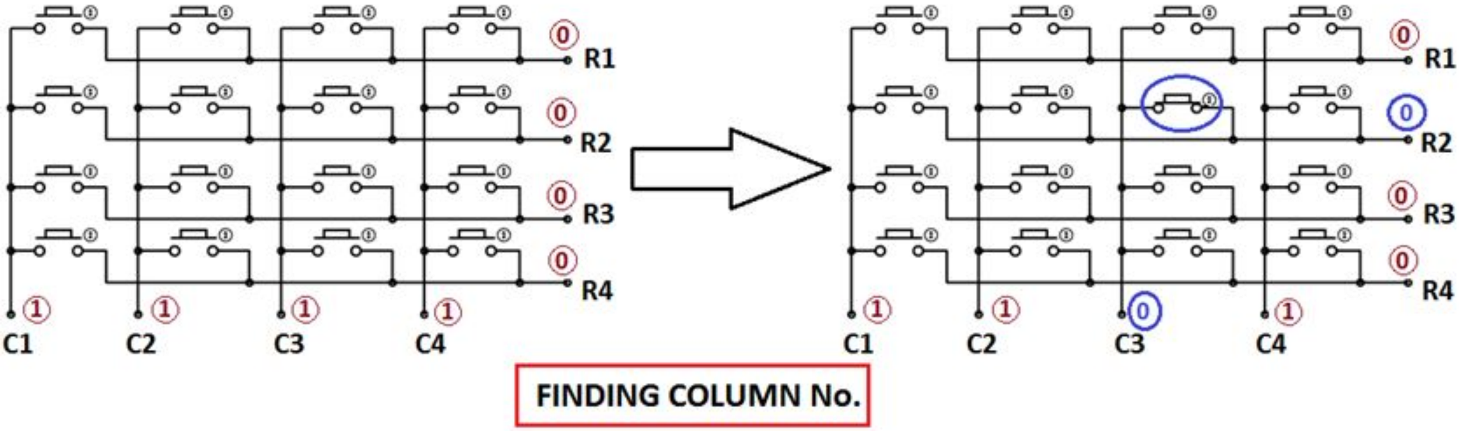
4x4 Keypad interfacing with PIC:

For finding Column number:

When a switch/key is pressed, the corresponding row and column will get short.

Output of the corresponding column goes to go low.

Since we have made all the rows zero so this gives the column number of the pressed key.

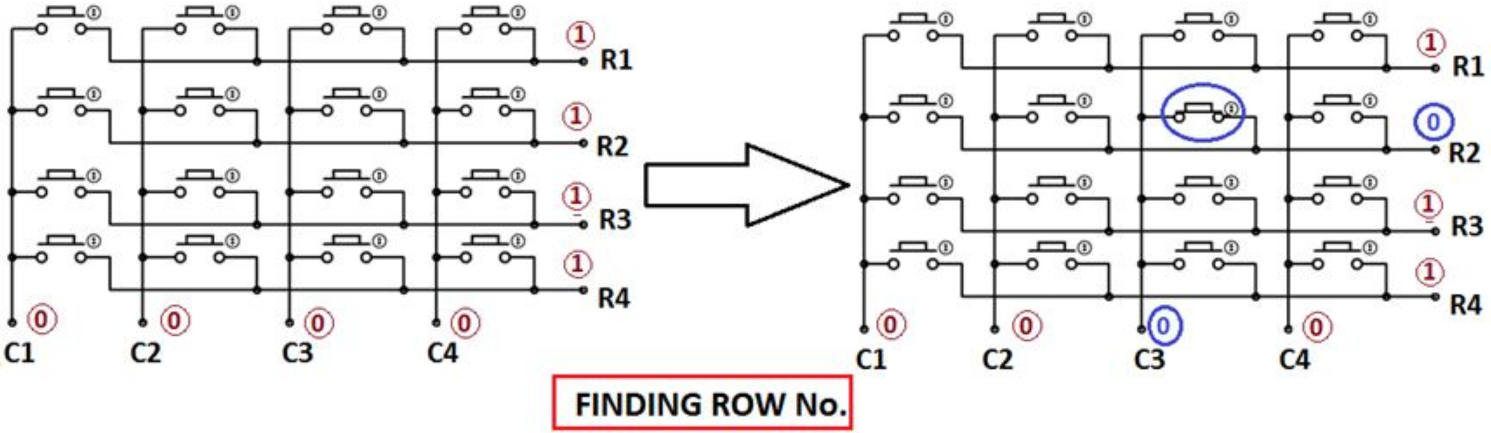


For Finding Row number:

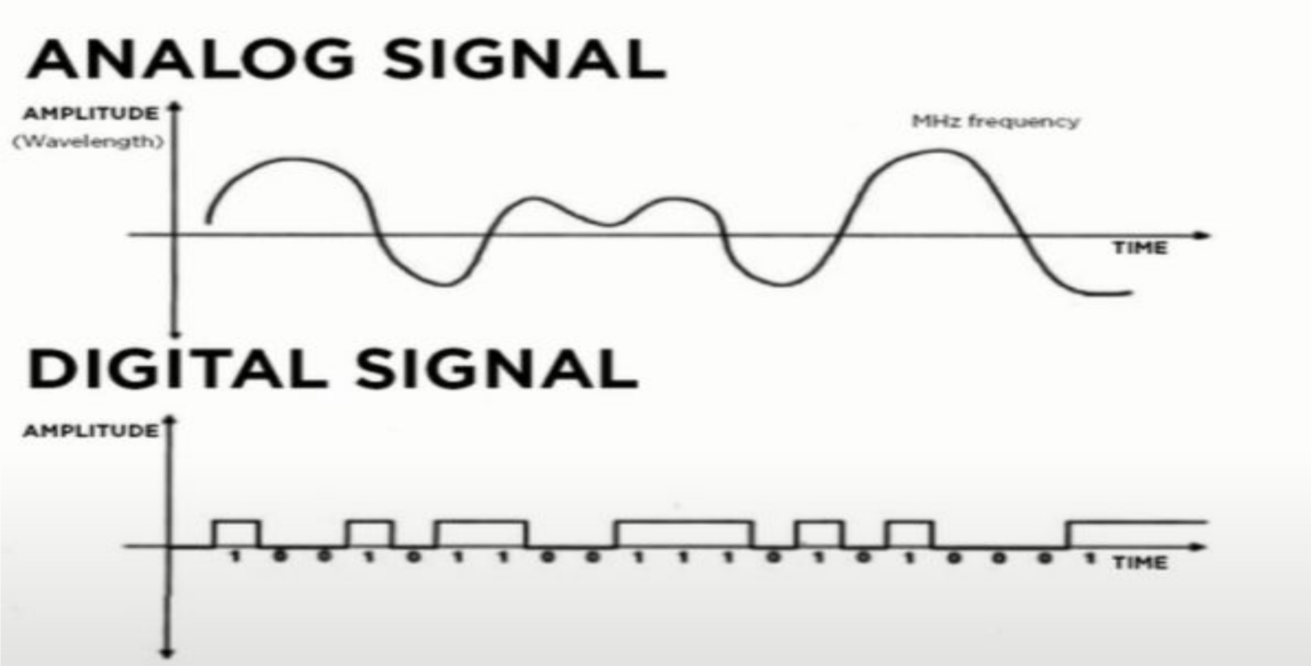
After the detection of column number, the controller set’s all the rows to high.

Each row is one by one set to high by the microcontroller and the earlier detected column is checked and obviously it becomes zero.

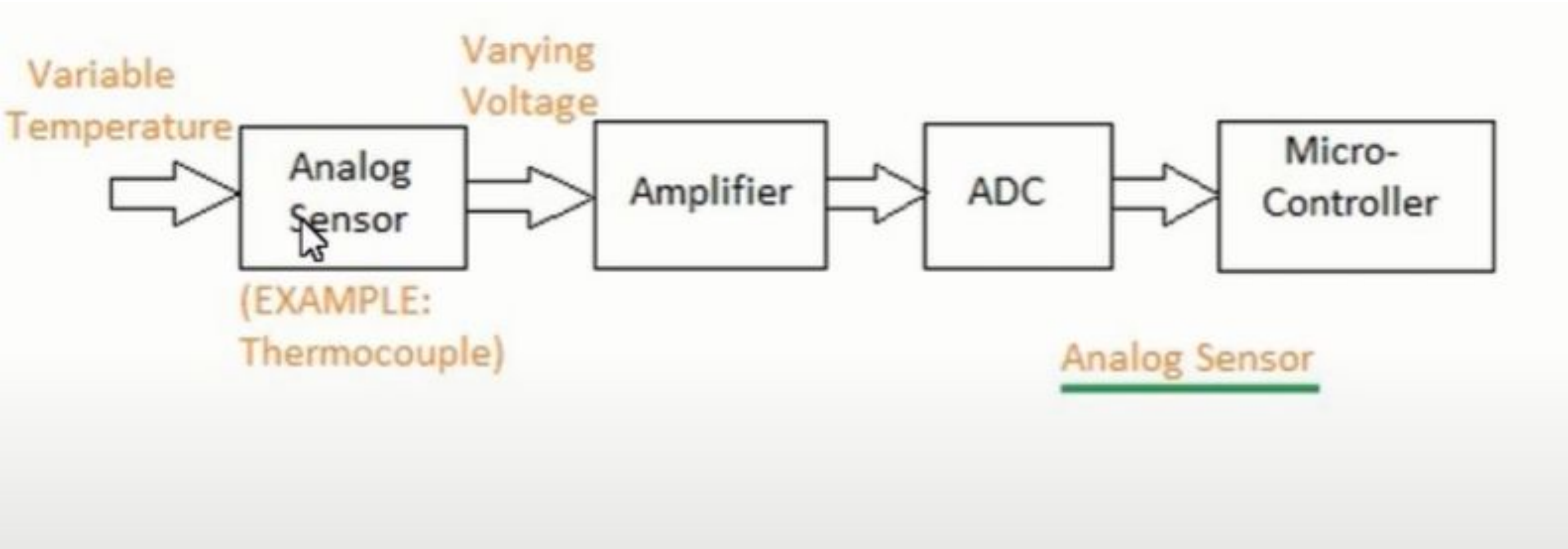
The row due to which the column gets zero is the row number of the pressed key.



Sensor Classification:

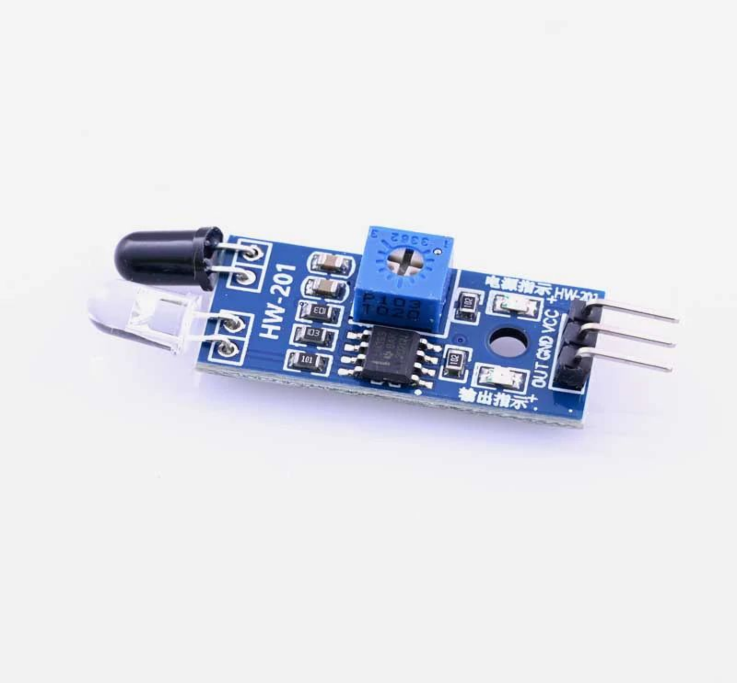


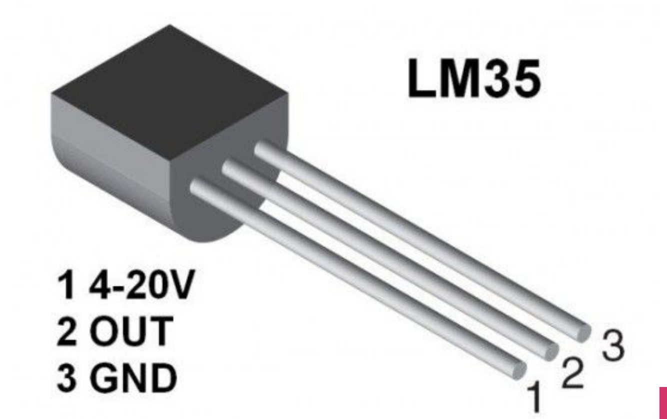
Analog to Digital Conversion Process:



Digital Sensor

Analog Sensor





Resolution:

Resolution refers to the conversion of an analog voltage to a digital value. It indicates the number of discrete values that an ADC can produce over the range of analog values. The values are usually stored electronically in binary form, so the resolution is usually expressed in bits.

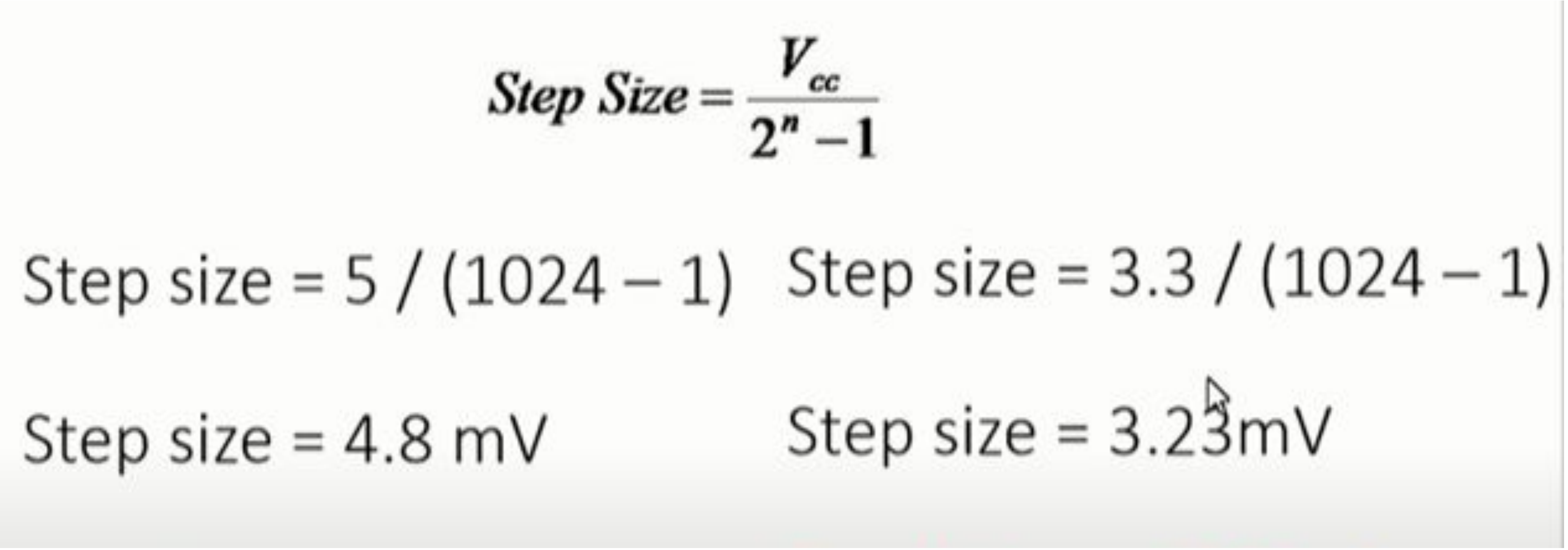
Resolution:

- 10 Bit ADC
- 10 Bit = 2^10
- 10 bit = 0 to 1023.

Step Size:

It is the voltage difference between one digital level and the next level that can be measured by ADC. In a 4-bit converter, an input of 1 Volt produce an output of 0001, then the step size is 1 Volt. 0 volts is always considered 0000. Distortion in the signal can be reduced by decreasing the step size to 0.5 volts but it lowers the maximum input reading. Ife we are using smaller step size, higher bit converter is needed to have maximum voltage.

It has **10 bit** resolution. Other ADC can have n bit resolution and n can be 8,10,12,16 or 24 bits. If ADC has higher resolution, it gives smaller step size.If ADC has 8 bit resolution, input voltage span is **0-5V** and the step size is 19.53mV (5V/1023).



$$Step\ Size = \frac{V_{cc}}{2^n - 1}$$

Step size = 5 / (1024 – 1)

Step size = 4.8 mV

Step size = 3.3 / (1024 – 1)

Step size = 3.23mV

ADC Value:

ADC value is 10 Bit Value

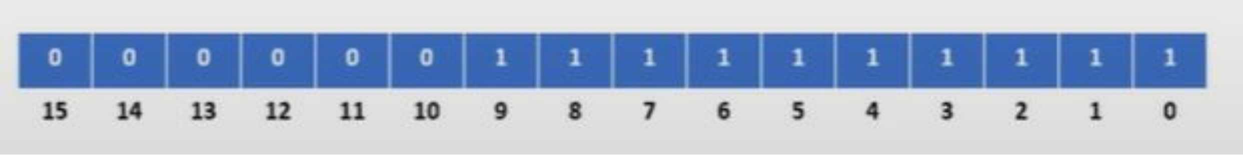
So the max. Value is 1023

And the min. Value is 0

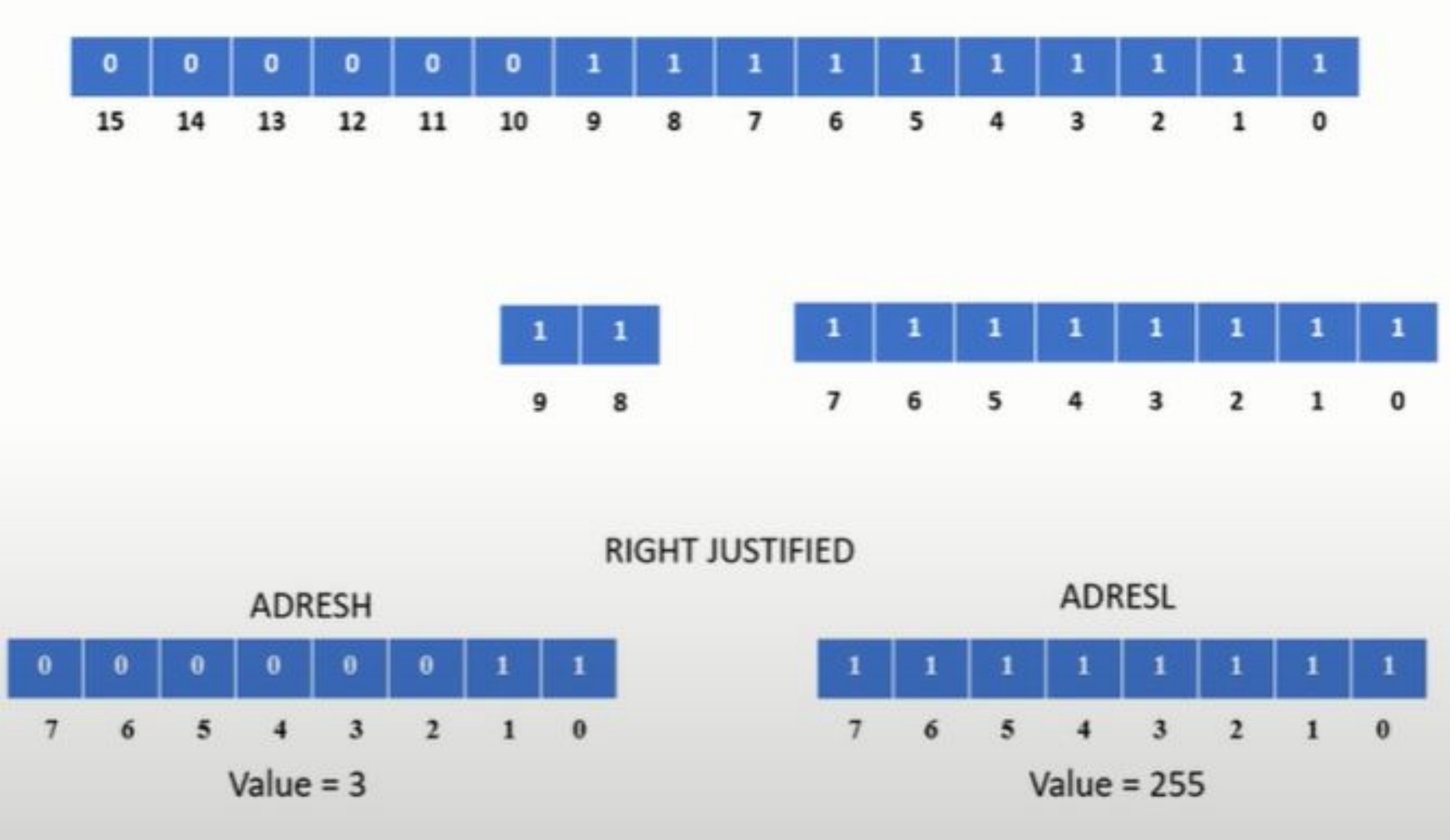
Let's assume the

Value is 1023

1023 in binary represented as,



0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0



0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

1	1
9	8

1	1	1	1	1	1	1	1
7	6	5	4	3	2	1	0

RIGHT JUSTIFIED

ADRESH

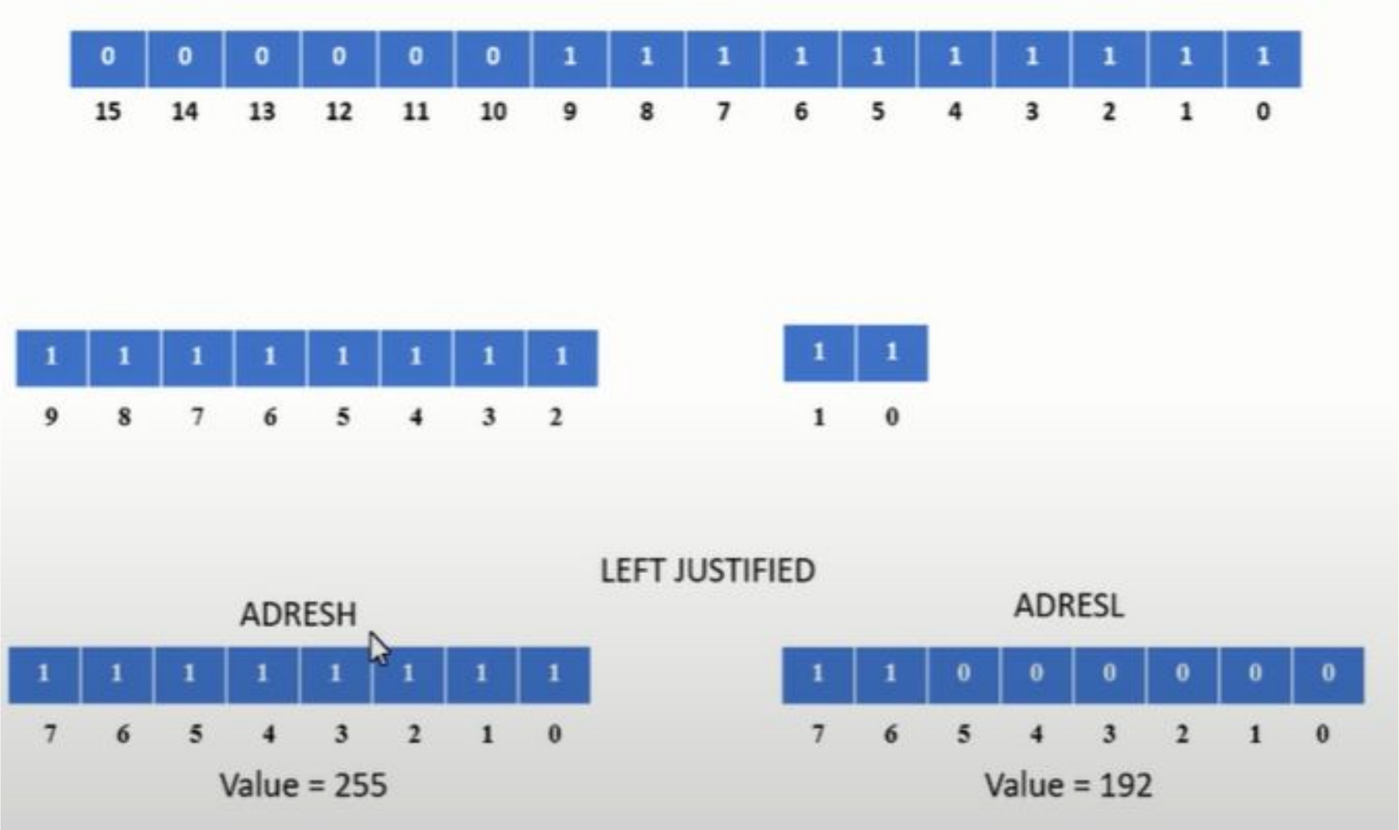
0	0	0	0	0	0	1	1
7	6	5	4	3	2	1	0

Value = 3

ADRESL

1	1	1	1	1	1	1	1
7	6	5	4	3	2	1	0

Value = 255



15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1

9	8	7	6	5	4	3	2
1	1	1	1	1	1	1	1

1	0
1	1

LEFT JUSTIFIED

ADRESH

7	6	5	4	3	2	1	0
1	1	1	1	1	1	1	1

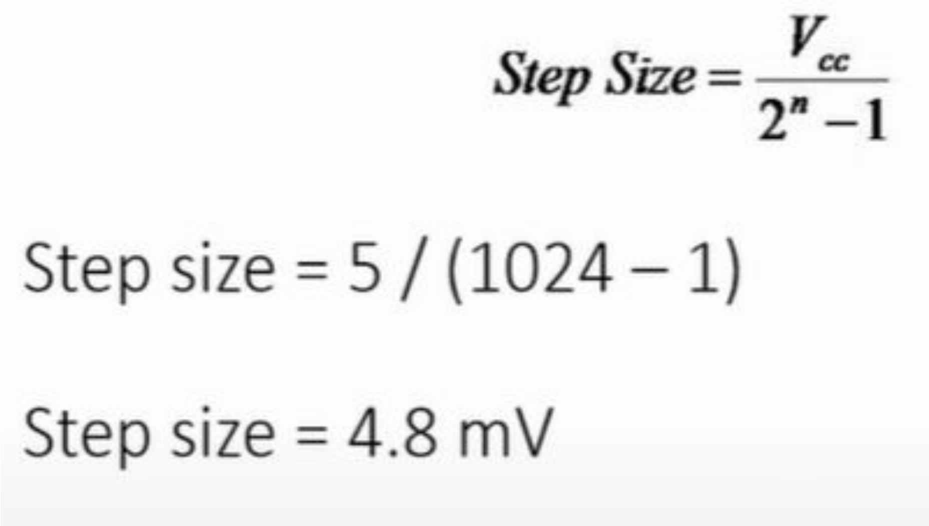
Value = 255

ADRESL

7	6	5	4	3	2	1	0
1	1	0	0	0	0	0	0

Value = 192

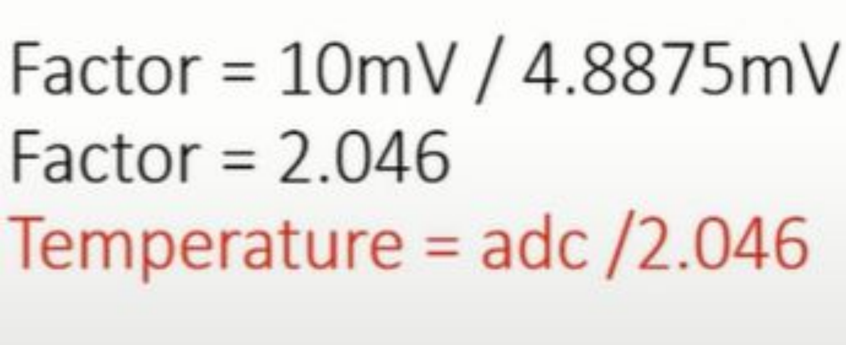
Step Size:



$$Step\ Size = \frac{V_{cc}}{2^n - 1}$$

Step size = 5 / (1024 – 1)

Step size = 4.8 mV



Factor = 10mV / 4.8875mV

Factor = 2.046

Temperature = adc /2.046

TIMER 0:

Ftimer = Fosc / Machine cycle

= 4 MHz / 4

= 1 MHz

Prescaler = Its Frequency divider.

F = 1MHz / 256

= 0.0039MHz

Ttimer = 1 / 0.0039MHz = 256.41 us.

Delay = Max. Count - Desired value / Ttimer

= 65535 - 20 ms / 256us

= 65535 - 20 X 10^(-3) / 256 X 10 (-6)

= 65535 - 0.0781 X 10^3

= 65535 - 78.12

Delay = 65456

Now, Convert Decimal into Hexa-decimal

Delay = FFB0.

TMR0H=0xFF

TMR0L=0xB0.

$$\boxed{fout = \frac{fclk}{4 * \text{Prescaler} * (256 - \text{TMR0}) * \text{Count}}} \quad \textit{where} \quad \boxed{Tout = \frac{1}{fout}}$$

Programming steps for delay function:

1. Load TXCON register value i.e.T0CON
2. Load calculated TMRxH value i.e. here TMR0H = 0x
3. Load calculated TMRxL value i.e. here TMR0L = 0x.
4. Start the timer by setting a TRx bit. i.e. here TMR0ON = 1.
5. Poll TMR1IF flag till it does not get set.
6. Stop the timer by clearing TMRxON bit. i.e. here TMR0ON = 0.
7. Clear timer flag TMRxIF bit i.e. here TMR0IF = 0.
8. Repeat from step 2 to 7 for the delay again.

TIMER 1:

Ftimer = Fosc / Machine cycle
= 4 MHz / 4
= 1 MHz

Prescaler = Its Frequency divider.

F = 1 MHz / 8
= 0.125MHz

Ttimer = 1 / 0.125 MHz = 8 us.

Delay = Max. Count - Desired value / Ttimer
= 65535 - 50 ms / 8 us
= 65535 - 50 X 10^(-3) / 8 X 10 (-6)
= 65535 - 6.25 X 10^3
= 65535 - 6250

Delay = 59285

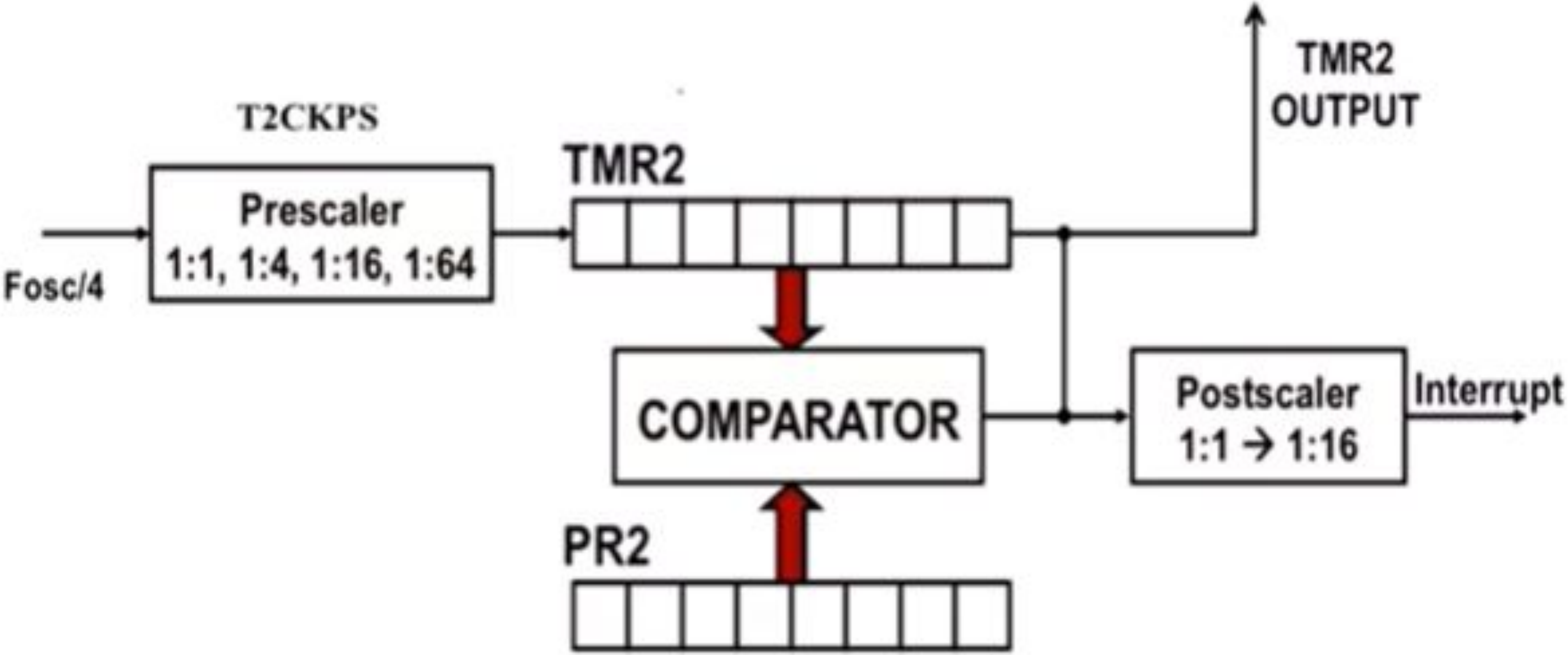
Now, Convert Decimal into Hexadecimal

Delay = E795.

TMR0H=0xE7

TMR0L=0x95.

TIMER 2 Diagram:



TIMER 2:

The TMR2 and PR2 compare its value. TMR2 has 00h but its increment by 1 per clock cycle, PR2 has 20h.
>> Max. value should not be greater than 255.

Ftimer = Fosc / Machine cycle
= 4 MHz / 4
= 1 MHz

Prescaler = Its Frequency divider. (16)

F = 1 MHz / 16
= 0.0625MHz

>> Now, select the postscaler (1)
F= 0.0625 / 1
= 0.0625

T timer = 1 / F
= 1 / 0.0625
= 16 us.

Delay = Desired time / Ttimer
= 5ms / 16 us.
= 0.3125
=312.5
255 < 312

INTERRUPT

An interrupt is an event that occurs randomly in the flow of continuity. It is just like a call you have when you are busy with some work and depending upon call priority you decide whether to attend or neglect it.

What is a PWM Signal?

Pulse Width Modulation (PWM) is a digital signal which is most commonly used in control circuitry. This signal is set high (5v) and low (0v) in a predefined time and speed. The time during which the signal stays high is called the “on time” and the time during which the signal stays low is called the “off time”.

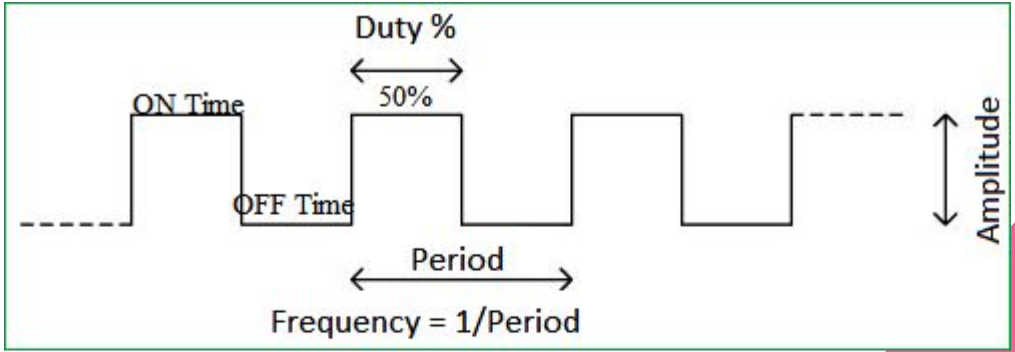
PWM using PIC18f:

PWM signals can be generated in our PIC Microcontroller by using the **CCP (Compare Capture PWM)** module. The resolution of our PWM signal is 10-bit, that is for a value of 0 there will be a duty cycle of 0% and for a value of 1024 (2^10) there be a duty cycle of 100%. There are two CCP modules in our PIC MCU (CCP1 And CCP2), this means we can generate two PWM signals on two different pins.

Duty cycle of the PWM:

The percentage of time in which the PWM signal remains HIGH (on time) is called as duty cycle. If the signal is always ON it is in 100% duty cycle and if it is always off it is 0% duty cycle.

Duty Cycle =Turn ON time/ (Turn ON time + Turn OFF time)



Ton: Time for which the signal is ON/HIGH.
Toff: Time for which the signal is OFF/LOW.
Period: The cycle time of the signal which is nothing but the sum of Ton and Toff period.

Period= Ton+Toff
Freq of Pwm signal = 1/Period.

PIC uses TIMER2 for generating the PWM signals. Let's relate the above concepts with the PIC registers.

Period:PR2 register is used to specify the PWM Period. The PWM period can be calculated using the following.

formula. PWM Period = [(PR2) + 1] * 4 * TOSC * (TMR2 Prescale Value) we have to **set the Frequency of the PWM signal**. The value of the frequency has to be written to the PR2 register. The desired frequency can be set by using the below formulae.

PWM Period = [(PR2) + 1] * 4 * TOSC * (TMR2 Prescale Value)

Rearranging these formulae to get PR2 will give

```
PR2 = (Period / (4 * Tosc * TMR2 Prescale )) - 1
```

We know that Period = (1/PWM_freq) and Tosc = (1/_XTAL_FREQ). Therefore.....

```
PR2 = (_XTAL_FREQ/ (PWM_freq*4*TMR2 PRESCALE)) - 1;
```

```
PR2 = (_XTAL_FREQ/ (PWM_freq*4*TMR2 PRESCALE)) - 1;
```

= 4 MHz / 5 KHz * 4 *16

=12-1

=11

Duty cycle means On-Time of the Total time (pulse)

when TMR2 and Duty cycle(CCPR1L) is matched then pulse go to Low

when TMR2 and PR2 is matched then pulse go to High.

For 10% Duty Cycle :

Duty cycle= 11 x 10 / 100

= 11x 0.1

= 1.1

= 1

Duty cycle means On-Time of the Total time (pulse).

For 50% of Duty Cycle

Duty cycle= 11 x 50 / 100

= 11 x 0.5

= 5.5

= 5

DCB1 and DCB0 for decimal point fraction of Duty Cycle

0	0	0
0	1	0.25
1	0	0.50
1	1	0.75

CCPR1L: This Register is used to store the integer part of Duty cycle.

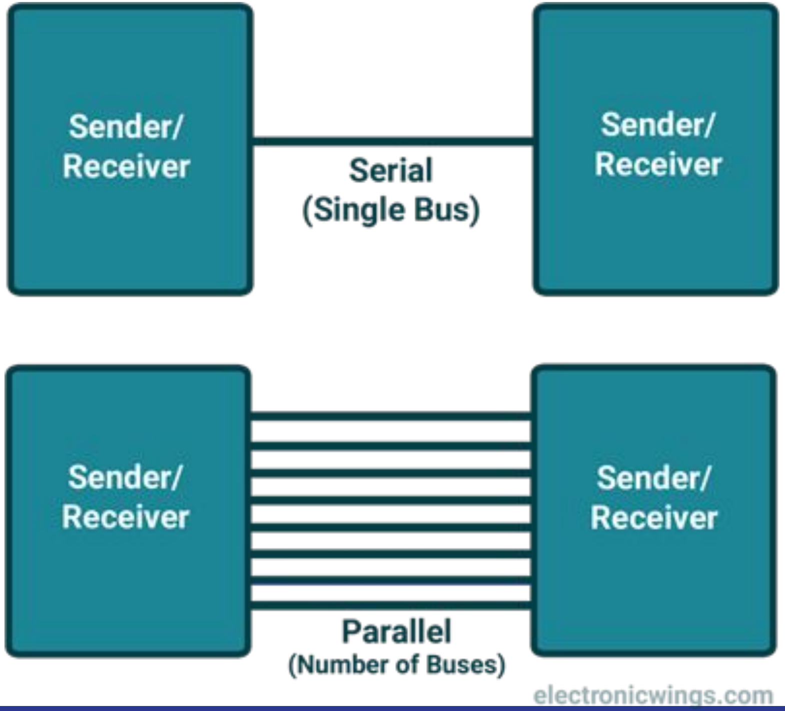
Example:

If Ans is 10.0 we mention then

Steps to Configure PWM:

1. Configure the CCP1 module for PWM operation.
2. Set the PWM period by writing to the PR2 register.
3. Set the PWM duty cycle by writing to the CCPR1L register and CCP1CON<5:4> bits.
4. Make the CCP1 pin an output by clearing the TRISC<2> bit.
5. Set the TMR2 prescale value and enable Timer2 by writing to T2CON.
6. Start the Timer.

UART(Universal Asynchronous Reception Transmit):



There are Two methods of serial communication:

- Synchronous Communication: Transfer of bulk data in the framed structure at a time
- Asynchronous Communication: Transfer of a byte data in the framed structure at a time

8051 has built-in UART with RXD (serial data receive pin) and TXD (serial data transmit pin).

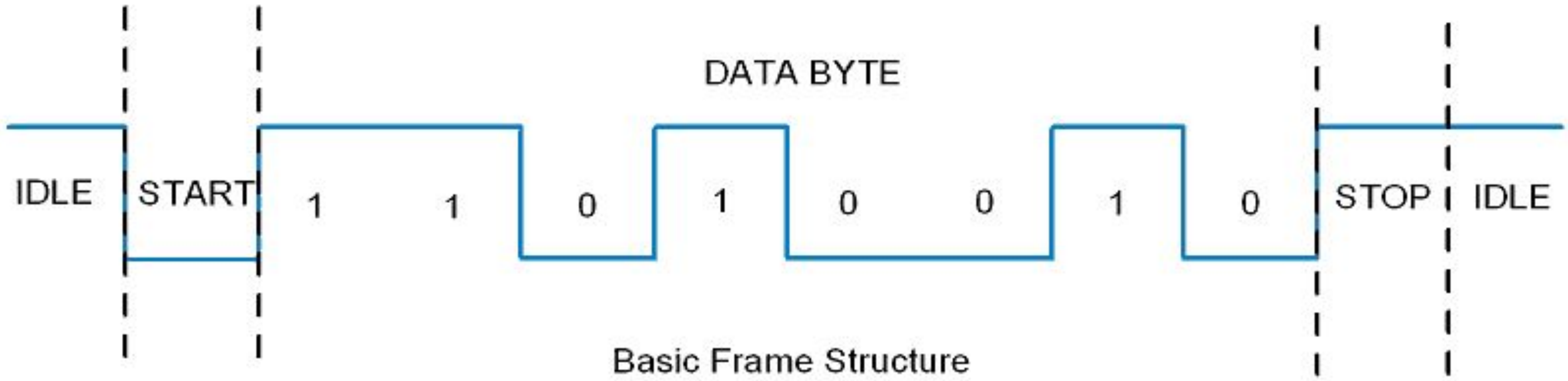
Asynchronous communication:

Asynchronous serial communication is widely used for byte-oriented transmission.

Frame structure in Asynchronous communication:

- **START bit:** It is a bit with which serial communication starts and it is always low.
- **Data bits packet**: Data bits can be 5 to 9 bits packet. Normally we use 8 data bit packet, which is always sent after the START bit.
- **STOP bit**: This is one or two bits. It is sent after the data bits packet to indicate the end of the frame. The stop bit is always logic high.

In an asynchronous serial communication frame, the first START bit followed by the data byte and at last STOP bit forms a 10-bit frame. Sometimes the last bit is also used as a parity bit.



Basic Frame Structure

UART:

When BRGH=1 (HighSpeed) the formula will be : Fosc / 16(n+1)

When BRGH=0 (Low Speed) the formula will be : Fosc / 64(n+1)

UART:

n= value of SPBRG

B.R = Fosc / 16 (n+1)

= 4 MHz / 16 (n+1)

9600 = 4 MHz / 16(n+1)

16(n+1) = 4 MHz / 9600

N+1 = [(4 x 10^6) / (16 x 9600)] - 1

N = [1000 / 384] - 1

= 26.89-1

N = 25