PSDL LAB

PSDL PRACTICAL STUDY MATERIAL

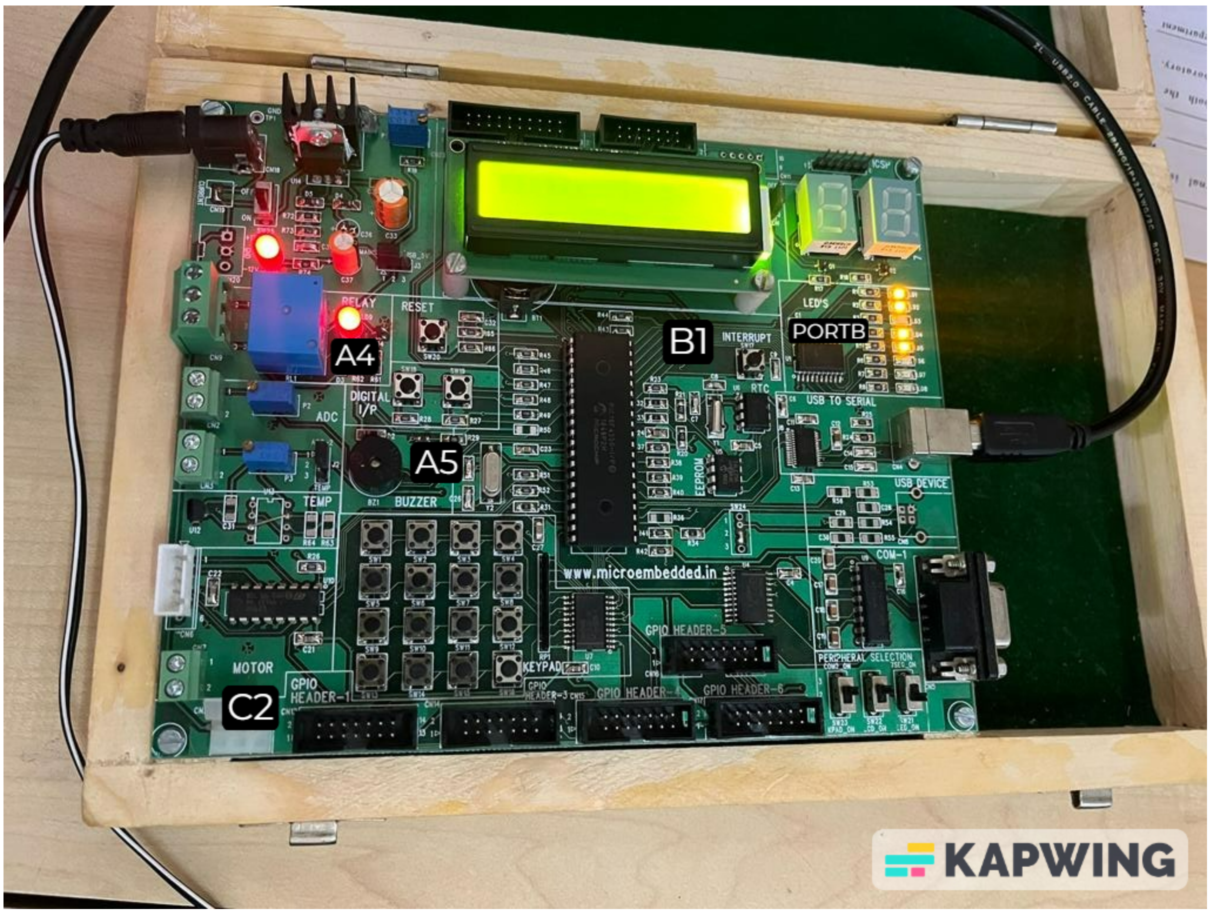
LIST OF PINS IN ASSIGNMENT :-

ASSIGNMENT 4 - LEDS - PORTB ALL

ASSIGNMENT 5 - BUZZER - PORTA A5

ASSIGNMENT 6 - RELAY - PORTA A4 ; EXTERNAL INTERRUPT - PORTB B1

ASSIGNMENT 7 - DC MOTOR - PORTC C2



Assignment-1

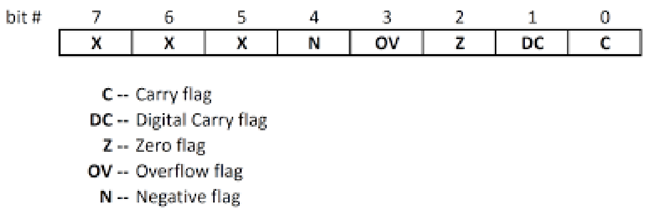
Theory :-

The PIC18F4550 microcontroller has a total of 40 pins. These pins are divided into different ports as follows:

1. PORTA: PORTA is an 7-bit port, consisting of pins RA0 to RA6. These pins can be configured as general-purpose input/output (GPIO) pins.

2. PORTB: PORTB is an 8-bit port, consisting of pins RB0 to RB7. These pins can be configured as GPIO pins and also have additional functionality such as interrupts, analog input, and external memory interface signals.
3. PORTC: PORTC is an 8-bit port, consisting of pins RC0 to RC7. These pins can be configured as GPIO pins and also have additional functionality such as interrupts, PWM output, analog input, and communication interfaces such as SPI and I2C.
4. PORTD: PORTD is an 8-bit port, consisting of pins RD0 to RD7. These pins can be configured as GPIO pins and also have additional functionality such as interrupts, PWM output, and communication interfaces such as USART (UART).
5. PORTE: PORTE is a 3-bit port, consisting of pins RE0 to RE2. These pins can be configured as GPIO pins and also have additional functionality such as interrupts and analog input.

STATUS REGISTER



bits in the STATUS register:

1. C (Carry) Bit: This bit is used to indicate a carry or borrow in arithmetic or logical operations. It is set or cleared by instructions that perform addition, subtraction, shift, or rotate operations.
2. DC (Digit Carry) Bit: The DC bit is used to indicate a carry or borrow in the least significant 4 bits (nibble) during arithmetic or logical operations. It is set or cleared by instructions that perform BCD (Binary Coded Decimal) arithmetic or logical operations.
3. Z (Zero) Bit: The Z bit is set when the result of an arithmetic or logical operation is zero. It is cleared when the result is non-zero.
4. Overflow (OV) Flag: The overflow flag (OV) is set when the result of a signed number operation is too large, causing the high order bit to overflow into the sign bit. Generally, the carry flag is used to detect errors in unsigned arithmetic operations while the overflow flag is used to detect errors in signed **arithmetic operations**.
5. Negative (N) Flag: The negative flag (N) is set when the result of an operation is negative. The N flag represents the sign of the result. If the most significant bit (MSB) of the result is set to 1, indicating a negative value, the N flag is set. If the MSB is 0, indicating a positive or zero value.

Codes :-

1.) ADD 8-bit with 8-bit

```
#include <pic18f458.h>
```

```
void main(void){
    TMR0L=0x01;
    TMR0H=0x02;

    TMR2=TMR0L+TMR0H;
    return;
}

//TO SHOW ON A PORT

void main(void){
    TMR0L=0x01;
    TMR0H=0x02;

    TMR2=TMR0L+TMR0H;

    TRISB=0;
    PORTB=TMR2;
    return;
}
```

2.) ADD 16-bit with 16-bit

```
#include <pic18f458.h>

void main(void){
    TMR0=0x0101;
    TMR1=0x0002;

    TMR1=TMR1+TMR0;
    return;
}

//TO SHOW ON A PORT

void main(void){
    TMR0=0x0101;
    TMR1=0x0002;

    TMR1=TMR1+TMR0;

    TRISB=0;
    TRISC=0;
    PORTB=TMR1H;
    PORTC=TMR1L;
    return;
}
```

3.) SUB 8-bit from 8-bit

```
#include <pic18f458.h>

void main(void){
    TMR0L=0x01;
    TMR0H=0x02;

    TMR2=TMR0L-TMR0H;
    return;
}

//TO SHOW ON A PORT

void main(void){
    TMR0L=0x01;
    TMR0H=0x02;

    TMR2=TMR0L-TMR0H;

    TRISB=0;
    PORTB=TMR2;
    return;
}
```

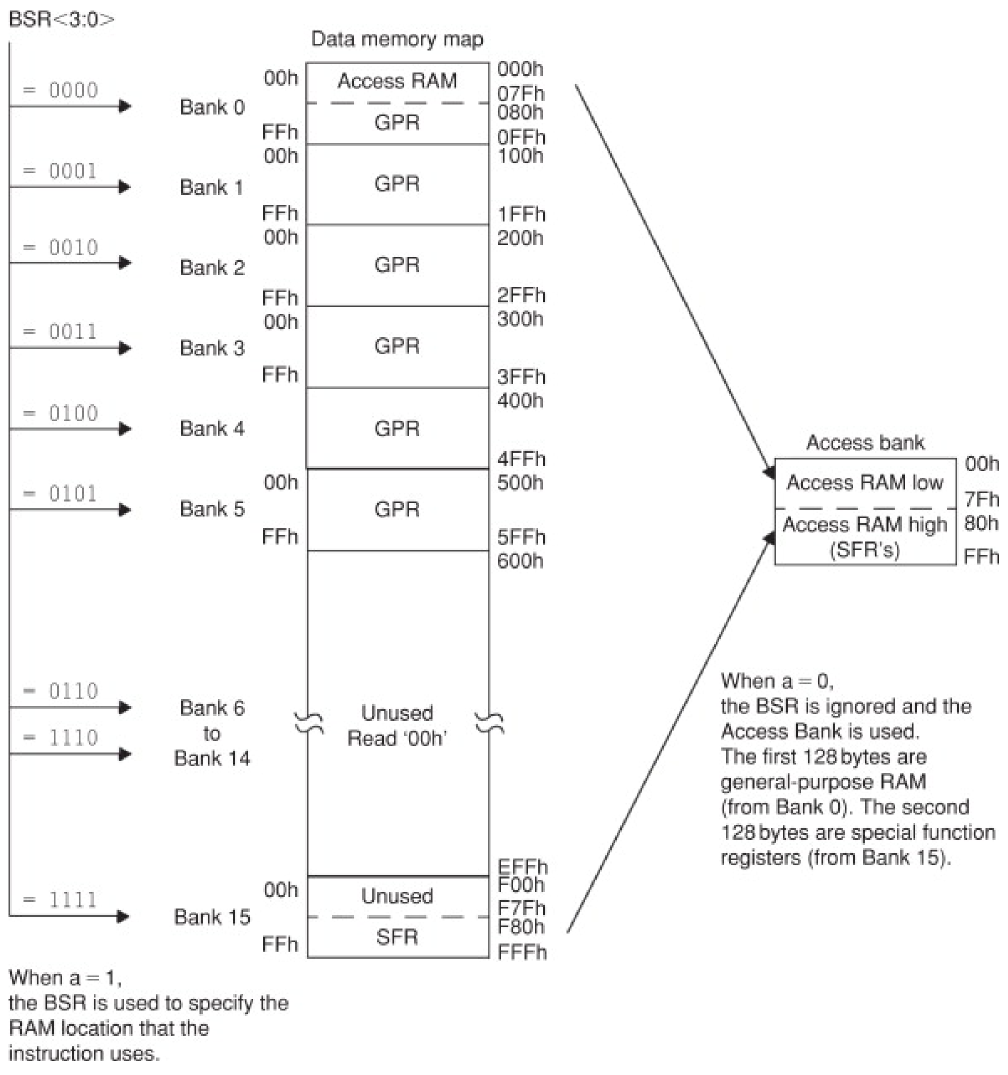
4.) SUB 16-bit with 16-bit

```
#include <pic18f458.h>

void main(void){
    TMR0=0x0101;
    TMR1=0x0002;

    TMR1=TMR1-TMR0;
    return;
}

//TO SHOW ON A PORT

void main(void){
    TMR0=0x0101;
    TMR1=0x0002;

    TMR1=TMR1-TMR0;

    TRISB=0;
    TRISC=0;
    PORTB=TMR1H;
    PORTC=TMR1L;
    return;
}
```

Assignment-2

Theory :-

- The file register of the PIC18 family can have a maximum of 4096 (4K) bytes. With 4096 bytes, the file register has addresses of 000-FFFH. The file register in the PIC18 is divided into 256 byte banks. We can have up to a maximum of 16 banks (16 X 256=4096).
- The 256 byte access bank is divided into two sections with 96 bytes of general purpose registers and 160 bytes of special function registers.



Codes :-

```
#include <pic18f458.h>
#include<stdio.h>

void main(void) {

    int ar[80];
    for(int i=0;i<80;i++)
    {
        ar[i]=i+1;
    }
    int arr[128],sum=0;
    for(int i=0;i<128;i++)
    {
        arr[i]=i+1;
        sum+=arr[i];
    }
    int s= sum;
    return;
}
```

Assignment-3

Theory :-

The registers associated with ports are used for configuring and controlling the input/output (I/O) operations of the general-purpose I/O (GPIO) pins. Each port usually consists of three primary registers:

1. TRISx (Data Direction Register):
   The TRISx register (where 'x' represents the port letter) determines the direction of the I/O pins.

- Setting a bit to 1 configures as an input
- clearing it to 0 configures the pin as an output.

1. PORTx (Port Register):
   The PORTx register is used for reading the current state of the I/O pins. Each bit in the PORTx register corresponds to an individual pin of the port. When a pin is configured as an input, reading the corresponding bit in the PORTx register retrieves the logic level (high or low) of that pin.
2. LATx (Latch Register):
   The LATx register (Latch Register) is used for writing data to the output pins of the port. Each bit in the LATx register corresponds to an individual pin of the port. Writing a 1 to a bit in the LATx register sets the corresponding pin to a high logic level, and writing a 0 sets it to a low logic level.

Codes :-

```
#include <pic18f458.h>

void main(void) {
    TRISB = 0;
    TRISC = 0;
    TMR1H = 0X05;
    TMR1L = 0X0A;

    int ch = 0XFF;
    while(1)
    {
        if(ch == 1)
        {
            TMR1 = TMR1H * TMR1L;
            PORTB = TMR1H;
            PORTC = TMR1L;
        }
        if(ch == 2)
        {
            PORTB = TMR1H/TMR1L;
            PORTC = TMR1H % TMR1L;
        }
    }
}
```

Assignment-4

Theory :-

Here are the timers available in the PIC18F4550:

1. Timer0:
   Timer0 is an 8-bit/16-bit timer with an associated prescaler. It can be used for general-purpose timing and generating periodic interrupts. Timer0 can also be configured as a counter to count external events.
2. Timer1:
   Timer1 is a 16-bit timer/counter with an associated prescaler. It can be used for more precise timing requirements. Timer1 also supports input capture and output compare functions, making it suitable for applications like pulse width modulation (PWM) and frequency measurement.
3. Timer2:
   Timer2 is an 8-bit timer/counter with an associated postscaler and prescaler. Timer2 is commonly used for generating fixed-frequency PWM signals.
4. Timer3 (Enhanced Timer):
   The PIC18F4550 microcontroller also features 16-bit timers, Timer3 .These timers provide more advanced features, including input capture, output compare, and PWM capabilities.

These timers can be configured and controlled using specific registers associated with each timer module. The configuration includes settings for prescaler, postscaler, timer mode, interrupt enable, and other parameters depending on the specific requirements of your application.

Control Register

The T0CON register is a readable and writable register that controls all the aspects of Timer0, including the prescale selection.

Register 13-1: T0CON: TImer0 Control Register

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
TMR0ON	T08BIT	T0CS	T0SE	PSA	T0PS2	T0PS1	T0PS0
bit 7							bit 0

bit 7	**TMR0ON:** Timer0 On/Off Control bit 1 = Enables Timer0 0 = Stops Timer0
bit 6	**T08BIT**: Timer0 8-bit/16-bit Control bit 1 = Timer0 is configured as an 8-bit timer/counter 0 = Timer0 is configured as a 16-bit timer/counter
bit 5	**T0CS**: Timer0 Clock Source Select bit 1 = Transition on T0CKI pin is clock (counter mode) 0 = Internal instruction cycle is clock (timer mode)
bit 4	**T0SE**: Timer0 Source Edge Select bit 1 = Increment on high-to-low transition on T0CKI pin 0 = Increment on low-to-high transition on T0CKI pin
bit 3	**PSA**: Timer0 Prescaler Assignment bit 1 = Timer0 prescaler is NOT assigned. Timer0 clock input bypasses prescaler. 0 = Timer0 prescaler is assigned. Timer0 clock input comes from prescaler output.
bit 2-0	**T0PS2:T0PS0**: Timer0 Prescaler Select bits These bits are ignored if PSA = 1 111 = 1:256 prescale value 110 = 1:128 prescale value 101 = 1:64 prescale value 100 = 1:32 prescale value 011 = 1:16 prescale value 010 = 1:8 prescale value 001 = 1:4 prescale value 000 = 1:2 prescale value

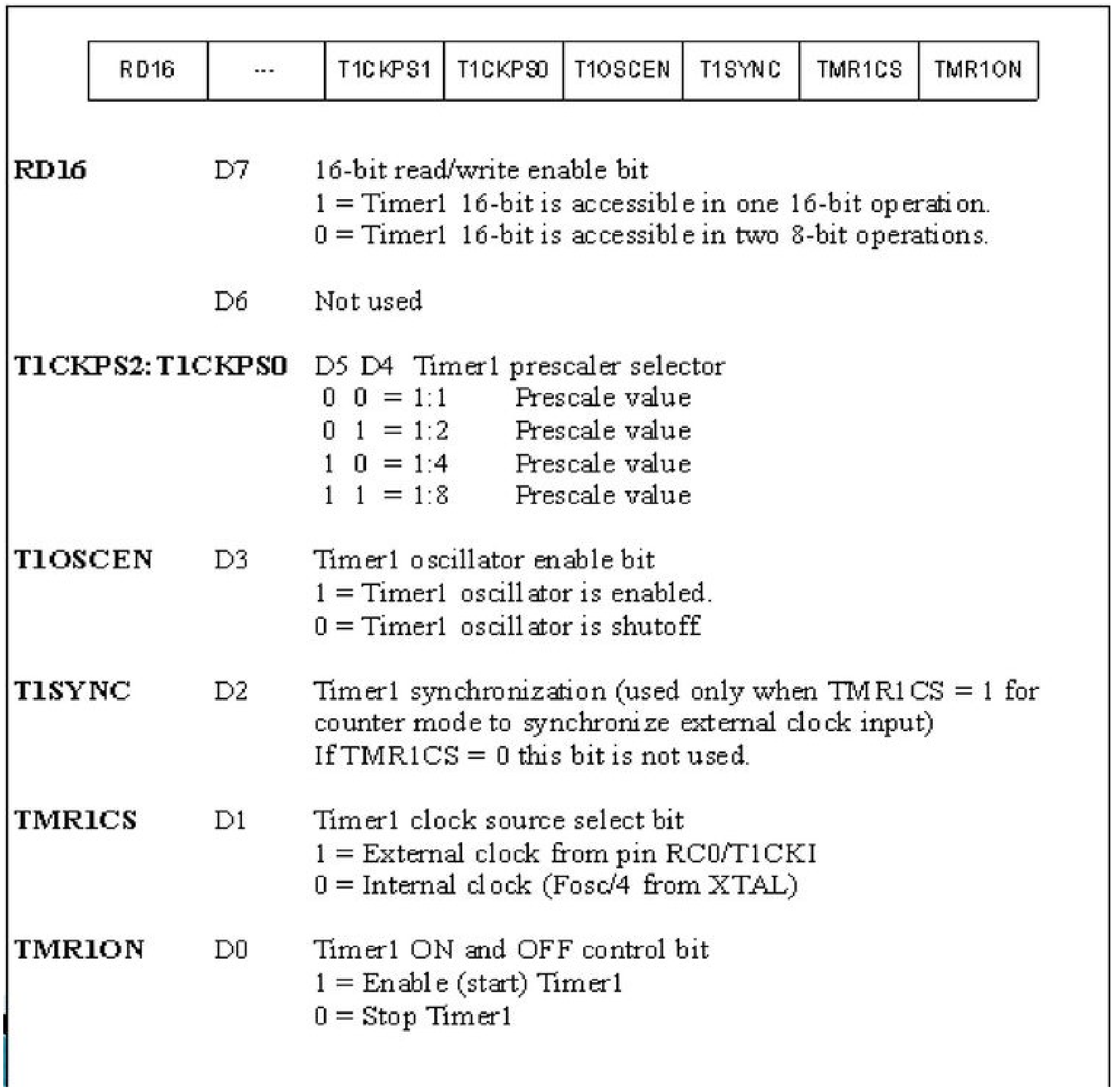
Legend			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as ‘0’	
- n = Value at POR reset	’1’ = bit is set	’0’ = bit is cleared	x = bit is unknown

Codes :-

```
#include <pic18f458.h>

void DELAY(){
    T0CON = 0X07; //configure control register
    TMR0 = 18661; //set tmr as Calculated
    INTCONbits.TMR0IF = 0; //RESET Interrupt FLAG
    T0CONbits.TMR0ON =1; // Turn ON TIMER
    while(INTCONbits.TMR0IF == 0); // till the timer stops and sets the flag
    T0CONbits.TMR0ON = 0; // trun OFF TIMER
    INTCONbits.TMR0IF = 0; //RESET Interrupt FLAG
}

void main(void) {
    TRISB = 0; //SET to OUTPUT MODE
    PORTB= 0x0F; // first four LED ON next Four OFF

    while(1)
    {
        PORTB=~PORTB; // toggle state of each led
        DELAY();
    }
    return;
}
```

Assignment-5

Theory :-



RD16	---	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC	TMR1CS	TMR1ON

RD16	D7	16-bit read/write enable bit 1 = Timer1 16-bit is accessible in one 16-bit operation. 0 = Timer1 16-bit is accessible in two 8-bit operations.
	D6	Not used
T1CKPS2:T1CKPS0	D5 D4	Timer1 prescaler selector 0 0 = 1:1 Prescale value 0 1 = 1:2 Prescale value 1 0 = 1:4 Prescale value 1 1 = 1:8 Prescale value
T1OSCEN	D3	Timer1 oscillator enable bit 1 = Timer1 oscillator is enabled. 0 = Timer1 oscillator is shutoff
T1SYNC	D2	Timer1 synchronization (used only when TMR1CS = 1 for counter mode to synchronize external clock input) If TMR1CS = 0 this bit is not used.
TMR1CS	D1	Timer1 clock source select bit 1 = External clock from pin RC0/T1CKI 0 = Internal clock (Fosc/4 from XTAL)
TMR1ON	D0	Timer1 ON and OFF control bit 1 = Enable (start) Timer1 0 = Stop Timer1

THREE BITS MUST BE SET TO GET OUTPUT OF INTERRUPT :-

1. Interrupt enable (IE)
2. Peripheral Interrupt enable (PEIE)
3. Global Interrupt enable (GIE)

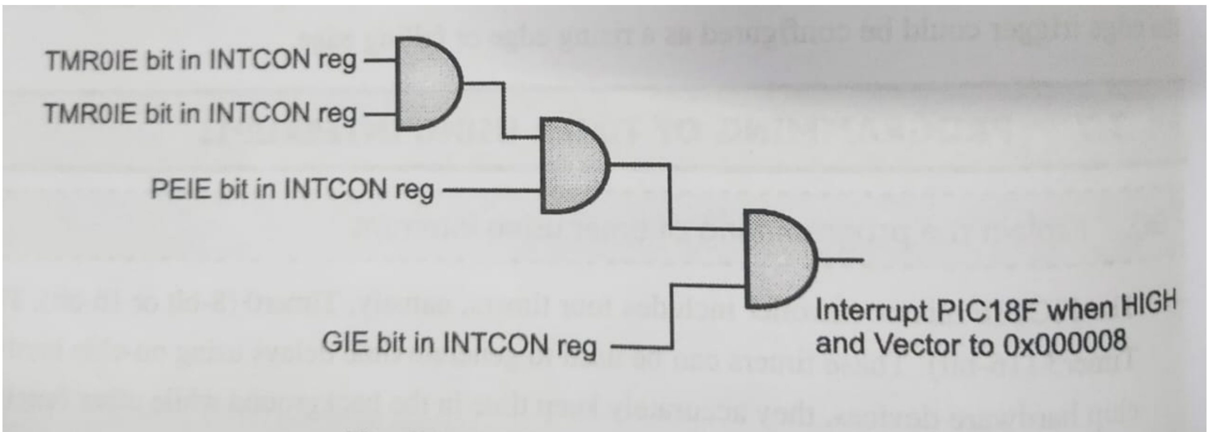


Fig. 3.7.1 : Timer0 interrupt

Codes :-

```
#include<pic18f4550.h>

int count=0;
int buzzer_on=0;

void __interrupt() Buzz(){
    if(PIR1bits.TMR1IF==1){
        if(buzzer_on && count==10){ //Condition for buzz is on (10 x 0.025 = 0.25 sec)
            buzzer_on=0;
            PORTAbits.RA5=0; // Buzzer off
            count=0;
        }
        if(!buzzer_on && count==40){ //Condition for buzz is off (40 x 0.025 = 1 sec)
            buzzer_on=1;
            PORTAbits.RA5=1; // Buzzer on
            count=0;
        }
        count++;
        TMR1=28036; // timer of 0.025 sec
        PIR1bits.TMR1IF=0; //reset Flag
    }
}
void main(void) {
    T1CON=0XB0;
    TRISAbits.TRISA5=0; //Set A5 pin in Output mode
    PORTAbits.RA5=0; // Buzzer off

    PIE1bits.TMR1IE=1; // enable TMR1 Interrupt
    PIR1bits.TMR1IF=0; // reset flag zero
    INTCONbits.PEIE=1; // enable peripheral Interrupt
    INTCONbits.GIE=1; // enable All Interrupt
    T1CONbits.TMR1ON=1; // Turn on the timer
    while(1);
    return;
}
```

Assignment-6

Theory :-

Interrupt Service Routine (ISR) :-

ISR stands for Interrupt Service Routine. It is a specific function or subroutine in a microcontroller program that is executed in response to an interrupt event. An interrupt is an event that causes the microcontroller to temporarily suspend its current execution and jump to a predefined location in memory to handle the interrupt.

External Interrupts :-

- pin RB0 - INT0
- pin RB1 - INT1
- pin RB2 - INT2

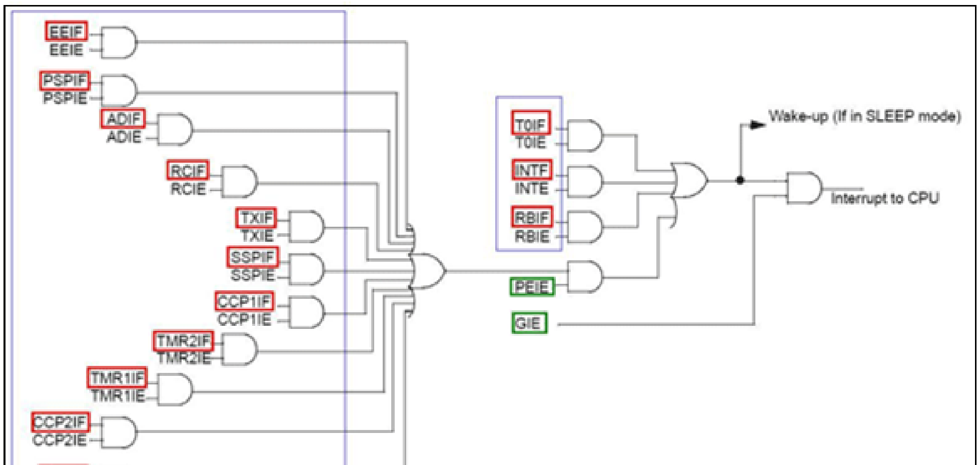
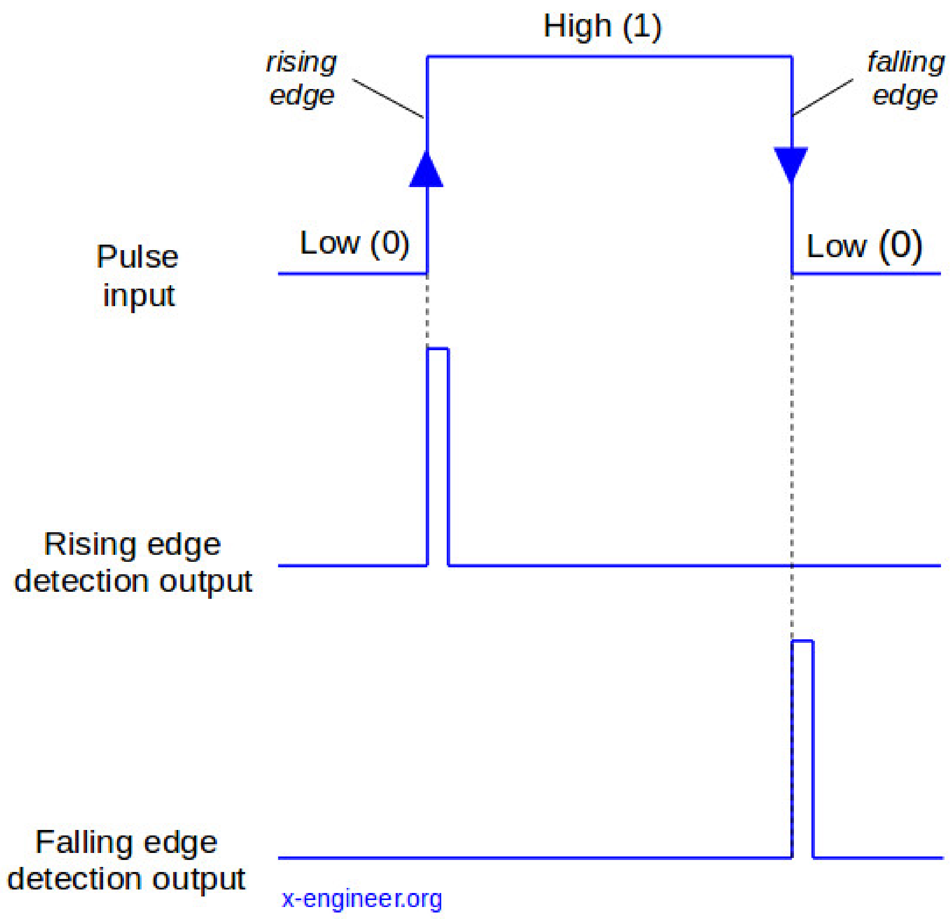


Figure: Interrupt Sorces and the interrupt and Flag bits

We can change edge by INTEDGx bit :

- 1 for Rising Edge ; Interrupt as soon Button pressed
- 0 for falling Edge ; Interrupt as soon Button is released



Codes :-

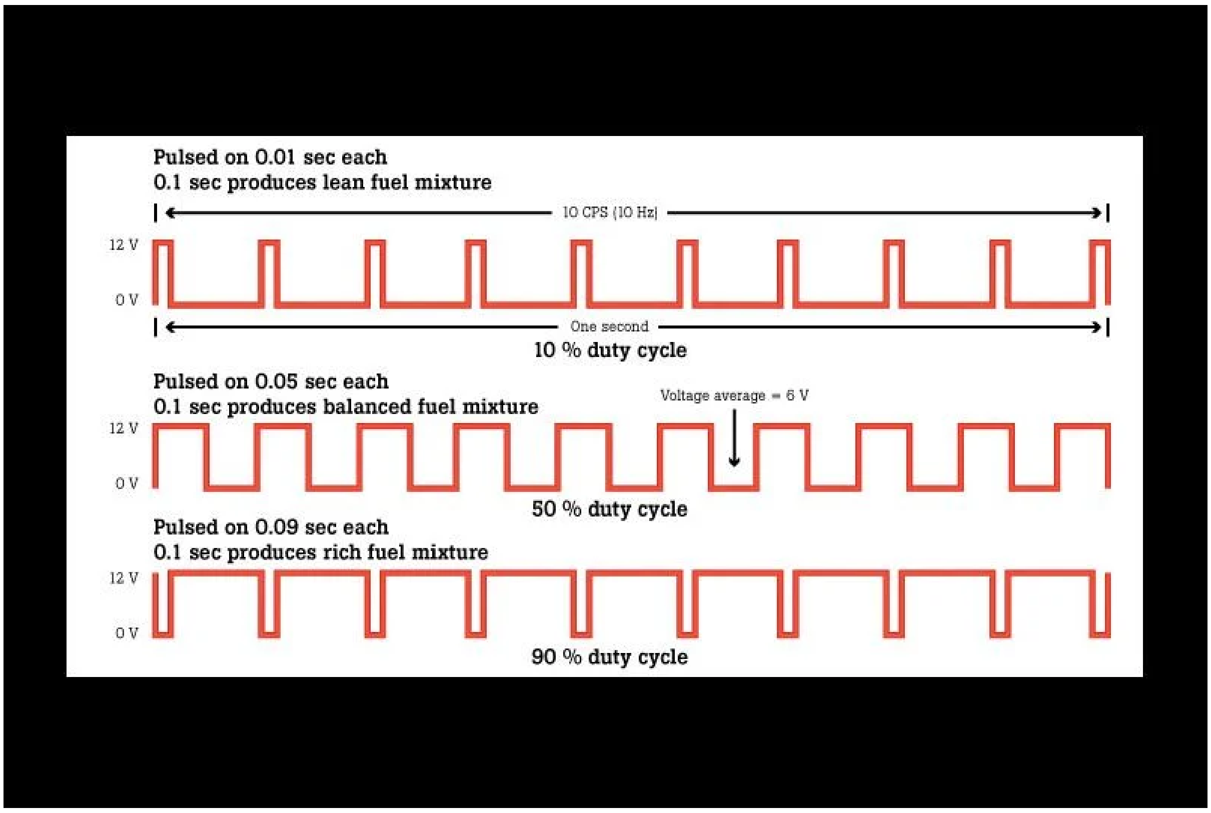
```
#include <pic18f4550.h>

void __interrupt() relay(){
    if(INTCON3bits.INT1IF == 1){
        INTCONbits.GIE = 0; // disable All Interrupt
        PORTAbits.RA4 = 1; //Turn ON RELAY

        for(int i=0; i<1000; i++)
            for(int j=0; j<100; j++); //DELAY

        PORTAbits.RA4 = 0; //TRUN OFF RELAY
        INTCON3bits.INT1IF = 0; // reset flag zero
        INTCONbits.GIE = 1; // enable All Interrupt
    }
}

void main(void){
    TRISBbits.RB1 = 1; // external interrupt INT1
    TRISAbits.RA4 = 0;      //RELAYYYY
    INTCON3bits.INT1IF = 0; // reset flag zero
    INTCON3bits.INT1IE = 1; // enable external Interrupt INT1
    INTCON2bits.INTEDG1 = 1; // 1 for Rising Edge , 0 for falling Edge
    INTCONbits.PEIE = 1; // enable peripheral Interrupt
    INTCONbits.GIE = 1; // enable All Interrupt
    while(1);

    return;
};
```

Assignment-7

Theory :-

DUTY CYCLE :

Duty cycle is the ratio of time a circuit is ON compared to the time circuit is OFF. When the signal is high, we call this "on time". To describe the amount of "on time" , we use the concept of duty cycle. Duty cycle is measured in percentage.



Pulse width is a measure of the actual ON time, measured in milliseconds. The OFF time does not affect signal pulse width.

PULSE WIDTH MODULATION :

Pulse width modulation or PWM is a commonly used control technique that generates analog signals from digital devices such as microcontrollers. The signal thus produced will have a train of pulses, and these pulses will be in the form of square waves. Thus, at any given time, the wave will either be high or low.

CCP MODULE (Capture/Compare/PWM) :

PIC18F4550 microcontroller has only one CCP module. Here are the key registers related to the CCP module:

1. CCP1CON (Capture/Compare/PWM1 Control Register):
   CCP1CON is the control register for CCP1 module. It is used to configure the operating mode, PWM output configuration, and other settings related to the CCP1 module.

2. CCPR1L (Capture/Compare/PWM Register 1 Low Byte):
3. CCPR1H (Capture/Compare/PWM Register 1 High Byte):

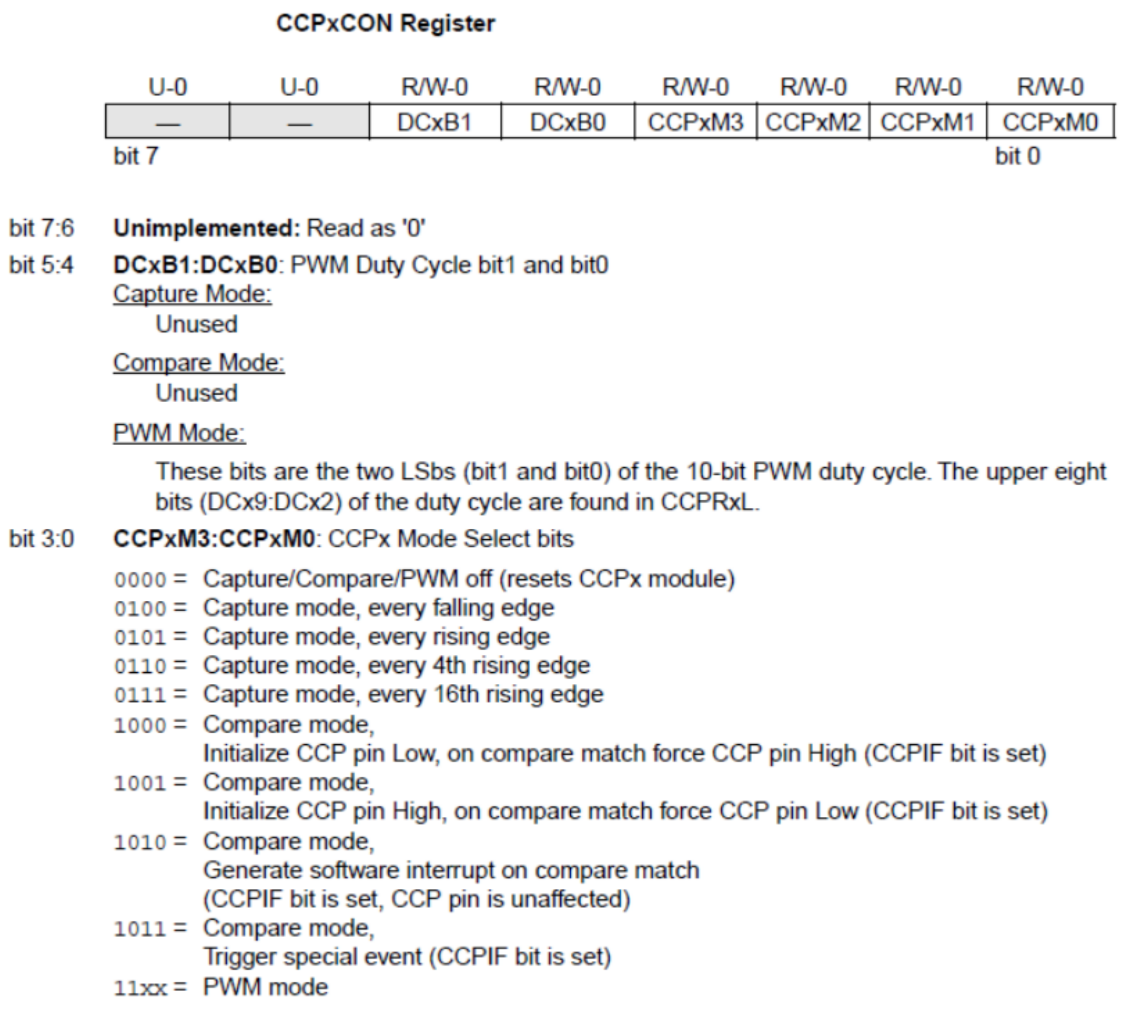
If CCP1 is configured as a PWM output, the duty cycle can be controlled using the register called CCPR1L (Capture/Compare/PWM Register 1 Low Byte) and the DC1B<1:0> bits in the CCP1CON register.

1. CCPR1L (Capture/Compare/PWM Register 1 Low Byte):
   CCPR1L is an 8-bit register that holds the lower 8 bits of the 10-bit value used to determine the duty cycle of the PWM signal. The 2 most significant bits of the duty cycle value are stored in the DC1B<1:0> bits of the CCP1CON register.

By adjusting the value in the CCPR1L register, you can control the duty cycle of the PWM output signal. Writing a new value to the CCPR1L register changes the PWM duty cycle accordingly.

1. DC1B<1:0> (PWM Duty Cycle bits):
   These two bits, located in the CCP1CON register, hold the two most significant bits of the 10-bit duty cycle value for the PWM output. They work in conjunction with the CCPR1L register to set the overall duty cycle of the PWM signal.

By modifying the DC1B<1:0> bits, you can adjust the higher 2 bits of the 10-bit duty cycle value, allowing finer control over the duty cycle.



CCPxCON Register

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	DCxB1	DCxB0	CCPxM3	CCPxM2	CCPxM1	CCPxM0
bit 7							bit 0

bit 7:6 **Unimplemented:** Read as '0'

bit 5:4 **DCxB1:DCxB0**: PWM Duty Cycle bit1 and bit0

Capture Mode:
Unused

Compare Mode:
Unused

PWM Mode:
These bits are the two LSbs (bit1 and bit0) of the 10-bit PWM duty cycle. The upper eight bits (DCx9:DCx2) of the duty cycle are found in CCPRxL.

bit 3:0 **CCPxM3:CCPxM0**: CCPx Mode Select bits

0000 =	Capture/Compare/PWM off (resets CCPx module)
0100 =	Capture mode, every falling edge
0101 =	Capture mode, every rising edge
0110 =	Capture mode, every 4th rising edge
0111 =	Capture mode, every 16th rising edge
1000 =	Compare mode, Initialize CCP pin Low, on compare match force CCP pin High (CCPIF bit is set)
1001 =	Compare mode, Initialize CCP pin High, on compare match force CCP pin Low (CCPIF bit is set)
1010 =	Compare mode, Generate software interrupt on compare match (CCPIF bit is set, CCP pin is unaffected)
1011 =	Compare mode, Trigger special event (CCPIF bit is set)
11xx =	PWM mode

Codes :-

In the below code we are decreasing duty cycle still the motor speed increases in practical because :

The Motor connections are interfaced as “ACTIVE LOW” in which speed increases when duty cycle decrease which is completely opposite to “ACTIVE HIGH” condition where speed increases when duty cycle increases.

```
#include <pic18f4550.h>

void main(void)
{
    TRISCbits.TRISC2 = 0;   //DC Motor
    CCP1CON = 0x0C;         //Set CCp Module in PWM Mode
    TRISAbits.TRISA4 = 0;   //Relay --> Just to visualize changes
    while(1)
    {
        for(int i=250; i>0; i-=50)//Loop to change Duty Cycle(inversely related to speed)
        {
            CCPR1L = i; //PWN Duty Cycle-->This Changes the speed of DC Motor
            for(int j=0; j<3000; j++)     //Delay
                for(int k=0; k<1000; k++);
                PORTAbits.RA4 = ~PORTAbits.RA4;     //Toggle Relay
        }
    }
}
```

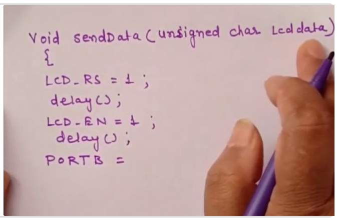
Assignment-8

Video :-

PIC_lecture 9: Interfacing of LCD with PIC Microcontroller | Embedded C program

For the theory of 8051 and PIC microcontroller refer the following blog: https://kkwtemicrocontrollers.blogspot.com/

https://www.youtube.com/watch?v=Jy1tbSEaBGc



```
void sendData(unsigned char Lcddata)
{
LCD_RS = 1;
delay();
LCD_EN = 1;
delay();
PORTB =
```

Codes :-

```
#include <pic18f4550.h>
#include <stdio.h>
#define LCD_RS PORTAbits.RA0
#define LCD_EN PORTAbits.RA1

void delay() {
    for (int i = 0; i < 1000; i++)
        for (int j = 0; j< 1000; j++);
}

void SendInstruction(int command) {
    LCD_RS = 0; // RS low : Instruction
    PORTB = command; // instruction no. for execution
    LCD_EN = 1; // EN High
    delay();
    LCD_EN = 0; // EN Low; command sampled at EN falling edge
    delay();
}

void SendData(int data) {
    LCD_RS = 1; // RS HIGH : DATA
    PORTB = data; // To display on LCD
    LCD_EN = 1; // EN High
    delay();
    LCD_EN = 0; // EN Low; data sampled at EN falling edge

    delay();
}

void main() {
    ADCON1 = 0x0F;
    TRISB = 0;
    TRISAbits.TRISA0 = 0; // For register select pin
    TRISAbits.TRISA1 = 0; // For Enable pin
    SendInstruction(0x38); //8 bit mode, 2 line,5x7 dots
    SendInstruction(0x01); //Clear display
    for (int i = 0; i < 1000; i++)
        delay();
    SendInstruction(0x80); //set address to 1st line
    while (1) {
        SendData(PORTB); // Display data stored at PORTB

        for (int i = 0; i < 1000; i++)
            delay();

    }

}
```

Assignment-9

Video :-

Interfacing Temperature Sensor LM35 with PIC18F4550

Interfacing Temperature Sensor LM35 with PIC18F4550

https://www.youtube.com/watch?v=lCV-n9RdKK4

Codes :-

```
#include <pic18f4550.h>
#include <stdio.h>

#define LCD_EN LATAbits.LA1
#define LCD_RS LATAbits.LA0
#define LCDPORT LATB

unsigned char str[16];

void lcd_delay(unsigned int time) {
    unsigned int i, j;

    for (i = 0; i < time; i++) {
        for (j = 0; j < 100; j++);
    }
```

```
}

void SendInstruction(unsigned char command) {
    LCD_RS = 0; // RS low : Instruction

    LCDPORT = command;
    LCD_EN = 1; // EN High
    lcd_delay(10);
    LCD_EN = 0; // EN Low; command sampled at EN falling edge
    lcd_delay(10);
}

void SendData(unsigned char lcddata) {
    LCD_RS = 1; // RS HIGH : DATA
    LCDPORT = lcddata;
    LCD_EN = 1; // EN High
    lcd_delay(10);
    LCD_EN = 0; // EN Low; data sampled at EN falling edge
    lcd_delay(10);
}

void InitLCD(void) {
    ADCON1 = 0x0F;
    TRISB = 0x00; //set data port as output
    TRISAbits.RA0 = 0; //RS pin
    TRISAbits.RA1 = 0; // EN pin

    SendInstruction(0x38); //8 bit mode, 2 line,5x7 dots

    SendInstruction(0x06); //entry mode
    SendInstruction(0x0C); //Display ON cursor OFF
    SendInstruction(0x01); //Clear display
    SendInstruction(0x80); //set address to 0
}

void LCD_display(unsigned int row, unsigned int pos, unsigned char *ch) {
    if (row == 1)
        SendInstruction(0x80 | (pos - 1));
    else
        SendInstruction(0xC0 | (pos - 1));

    while (*ch)
        SendData(*ch++);
}

void ADCInit(void) {
    TRISEbits.RE2 = 1; //ADC channel 7 input

    ADCON1 = 0b00000111; //Ref voltages Vdd &amp; Vss; AN0 - AN7 channels Analog
    ADCON2 = 0b10101110; //Right justified; Acquisition time 4T; Conversion clock Fosc/64
}

unsigned short Read_Temp(void) {
    ADCON0 = 0b00011101; //ADC on; Select channel;
    GODONE = 1; //Start Conversion

    while (GO_DONE == 1); //Wait till A/D conversion is complete
    return ADRES; //Return ADC result
}

int main(void) {
    unsigned int temp;
    InitLCD();
    ADCInit();
    LCD_display(1, 1, "Temperature:");
    while (1) {
        temp = Read_Temp();
        temp = ((temp * 500) / 1023);
        sprintf(str, "%d'C ", temp);
        LCD_display(2, 1, str);
        lcd_delay(9000);
    }
    return 0;
}
```

Assignment-10

Theory :-

The primary difference between a PIC microcontroller and an Arduino board lies in their underlying hardware and software architectures. Here are some key distinctions:

1. Microcontroller vs. Development Board:
   A PIC microcontroller refers to a family of microcontrollers produced by Microchip Technology. PIC microcontrollers are standalone integrated circuits that require additional external components for proper operation. On the other hand, an Arduino board is a development board that uses a microcontroller (usually an AVR microcontroller) as its core. Arduino boards are designed to simplify the process of prototyping and programming by providing a user-friendly platform with built-in hardware and software features.

2. Hardware Features:
   PIC microcontrollers offer a wide range of models with varying capabilities and features. They provide a rich set of peripherals such as timers, UART, SPI, I2C, ADC, and more. Arduino boards, being built around microcontrollers like AVR or ARM, typically have a set of built-in peripherals and additional components like voltage regulators, USB interfaces, and headers for connecting external devices.

3. Programming Environment:
   PIC microcontrollers are typically programmed using the MPLAB IDE (Integrated Development Environment) or other programming tools provided by Microchip. The programming is done using languages like C or assembly language. Arduino boards, on the other hand, have their own development environment called the Arduino IDE. The Arduino IDE simplifies the programming process by providing an easy-to-use interface and a simplified version of the C++ language.

4. Community and Ecosystem:
   The Arduino platform has a large and active community of users, makers, and developers. There is an extensive collection of open-source libraries, examples, and tutorials available for Arduino boards, making it easier for beginners to get started and for experienced users to share and collaborate on projects. PIC microcontrollers also have a dedicated community, but Arduino's community and ecosystem are generally more accessible and beginner-friendly.

5. Flexibility and Customizability:
   PIC microcontrollers offer a high level of flexibility and customization options since they can be used in various hardware designs and applications. With PIC microcontrollers, you have more control over the circuit design and component selection. Arduino boards, on the other hand, are pre-designed and pre-assembled with a specific microcontroller. While they offer some flexibility, they are generally less customizable than using a standalone PIC microcontroller.

In summary, PIC microcontrollers are versatile microcontroller chips with a wide range of capabilities, while Arduino boards are development platforms built around microcontrollers that provide a simplified and beginner-friendly environment for prototyping and programming. The choice between PIC microcontrollers and Arduino boards depends on factors such as project requirements, complexity, familiarity with programming, and level of customization needed.

Codes :-

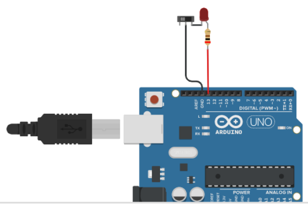
```
#define LED 13 //DEFINE 13 AS LED PIN

void setup()
{
  pinMode(LED, OUTPUT); //SET PIN 13 TO OUTPUT MODE
}

void loop()
{
  digitalWrite(LED, HIGH); //TURN ON THE LED
  delay(1000); // Wait for 1000 millisecond(s)
  digitalWrite(LED, LOW); //TURN OFF THE LED
  delay(1000); // Wait for 1000 millisecond(s)
}
```

Simulation link :-

Circuit design ASSIGNMENT 10 | Tinkercad

Circuit design ASSIGNMENT 10 created by Shivam Khetan with Tinkercad

https://www.tinkercad.com/things/0mN5bzzKd8i



Assignment-11

Theory :-

Working of temperature sensor :

Temperature sensor is transducer which converts heat energy into electrical signal. It contains thermistors and TDR (temperature dependent resistors) which can change voltages resulting in signal.

These values are read by analog pin and converted into digital signal via A/D converter. The digital signals are then processed for human understanding.

Codes :-

```
int TEMP = 0;

void setup()
{
  pinMode(A0, INPUT);
  pinMode(13, OUTPUT);
  pinMode(8, OUTPUT);
  pinMode(12, OUTPUT);
}

void loop()
{
  TEMP = map(((analogRead(A0) - 20) * 3.043), 0, 1023, -40, 125);
  if (TEMP < 10) {
    digitalWrite(13, HIGH);
    digitalWrite(8, LOW);
    digitalWrite(12, LOW);
  } else {
    if (TEMP < 35) {
      digitalWrite(12, HIGH);
      digitalWrite(8, LOW);
      digitalWrite(13, LOW);
    } else {
      digitalWrite(8, HIGH);
      digitalWrite(12, LOW);
      digitalWrite(13, LOW);
    }
  }
  delay(10); // Delay a little bit to improve simulation performance
}
```

Simulation link :-