**i. Write Assembly Level Language program for transfer of data from external memory to internal memory of 8051 uC.**
**Que. Explain structure of internal memory organization of PIC 18FXXXX uC.**

**Ans:**

**#Program:**

```
ORG 0000H
MOV R0, #20H
MOV DPTR, #2430H
MOV R2, #05H
BACK: MOV A, @R0
MOVX @DPTR, A
INC R0
INC DPTR
DJNZ R2, BACK
END
```

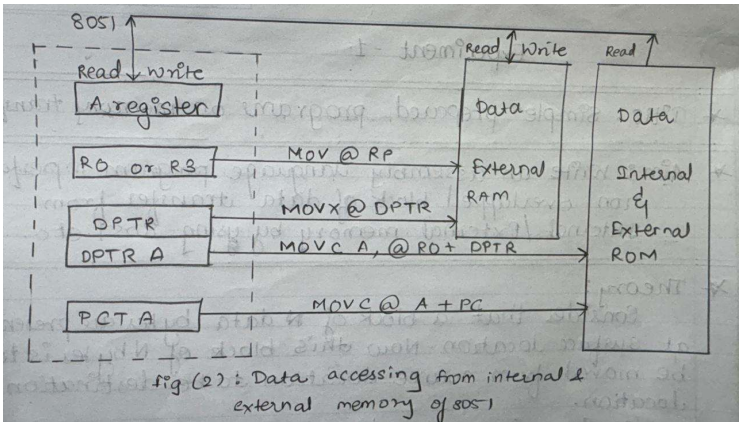
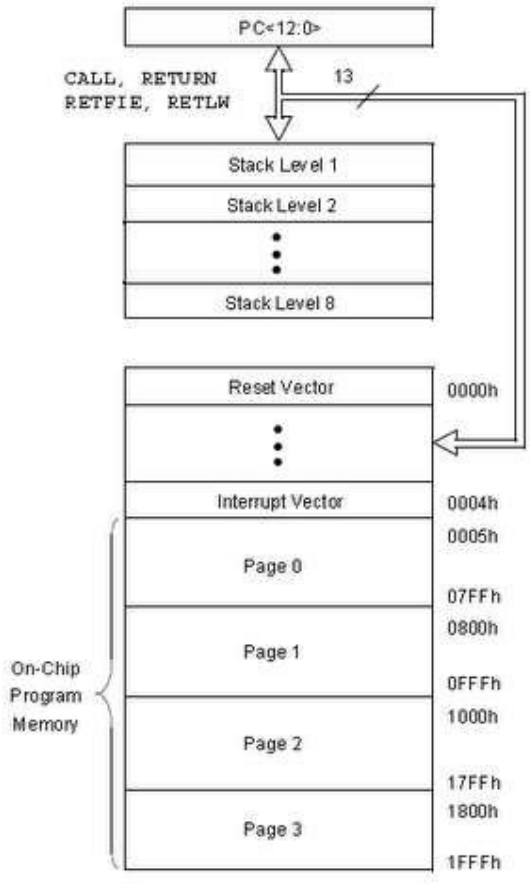


fig (2): Data accessing from internal & external memory of 8051

**#Ans:**



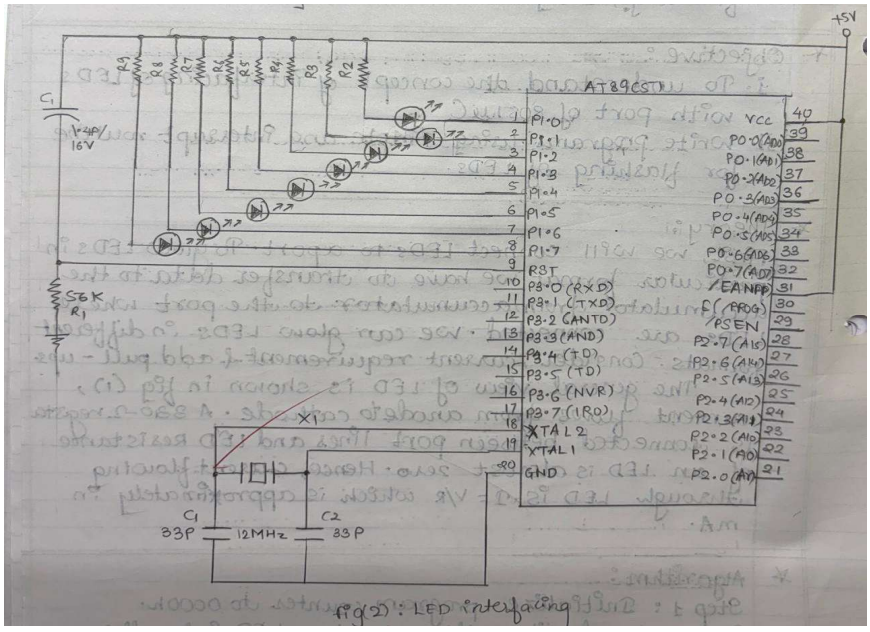

- **Harvard Architecture** → Separate buses for **program** and **data memory**.
- **Program Memory (Flash):**
  - Stores program code & interrupt vectors.
  - Non-volatile, 16-bit wide.
  - Self-programmable.
- **Data Memory (RAM):**
  - Volatile, 8-bit wide.
  - Divided into **GPRs (General Purpose Registers)** and **SFRs (Special Function Registers)**.
  - Organized in **banks**; selected using **BSR (Bank Select Register)**.
- **EEPROM:**
  - Non-volatile data memory.
  - Stores user data permanently.
  - Accessed via EECON, EEADR, and EEDATA registers.
- **Stack Memory:**
  - 31-level hardware stack.
  - Stores **return addresses** of subroutines and interrupts.
- **Configuration Bits:**
  - Define device settings (oscillator type, watchdog, etc.).

**ii. Write C-program for Parallel port interacting of LEDS with 8051 uC - Flashing LED.**
***Que.** Explain the Reset Control Register (RCON) PIC 18FXXXX uC*

**Ans:**

**#Program:**

```
#include <reg51.h>
void T1M1delay(void);
void main(void){
while(1){
P0 = 0x55;
T1M1delay();
P0 = 0xAA;
T1M1delay();
}
}
void T1M1delay(void){
unsigned char z;
for(z = 0; z < 50; z++){
TMOD = 0x10;
TH1 = 0xA5;
TL1 = 0x00;
TR1 = 1;
while (TF1 == 0);
TR1 = 0;
TF1 = 0;
}
}
```



Fig(2): LED interfacing

| Bit | Name | Description |
|---|---|---|
| 7 | **IPEN** | Enables **Interrupt Priority Levels** (1 = Enabled, 0 = Disabled). |
| 6 | **SBOREN** | Enables **Software Brown-out Reset**. |
| 5 | **RI** | **RESET Instruction Flag** — set if RESET instruction executed. |
| 4 | **TO** | **Time-out Flag** — 1 = No Watchdog timeout, 0 = Watchdog caused reset. |
| 3 | **PD** | **Power-down Flag** — 1 = After power-on, 0 = Woke up from sleep. |
| 2 | **POR** | **Power-on Reset Flag** — 1 = No POR occurred, 0 = Power-on reset occurred. |
| 1 | **BOR** | **Brown-out Reset Flag** — 1 = No BOR, 0 = Brown-out reset occurred. |
| 0 | — | Unused / Reserved. |

**#Ans:**

RCON stands for Reset Control Register. It is an 8-bit special function register (SFR) that indicates the cause of reset and controls power-down and brown-out features.

Purpose:
To detect the source of the last reset (like Power-on, Brown-out, Watchdog, etc.), To enable/disable certain reset features, To control memory state after sleep or reset.

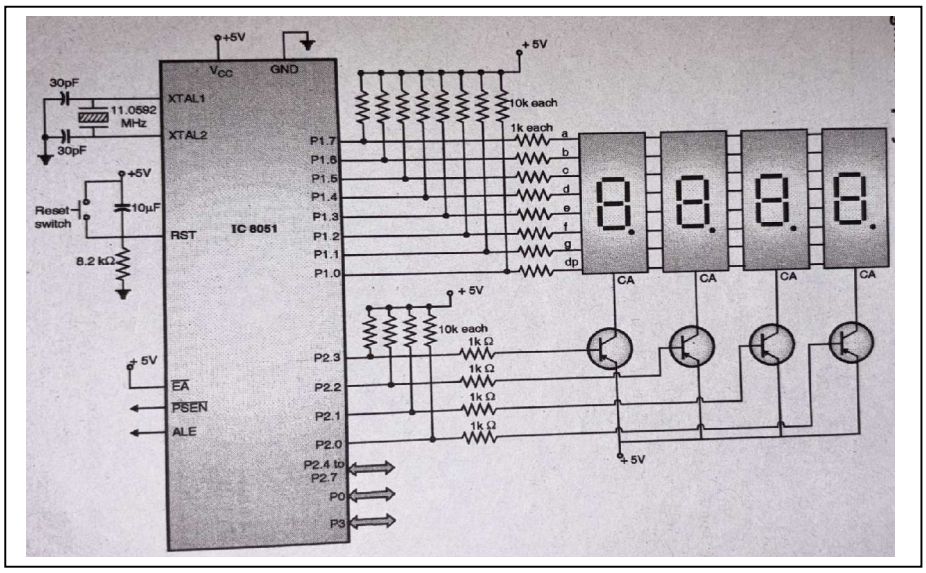
**iii. Write C-program for Interfacing of 7-segment display with 8051 uC for decade counting application.**

***Que.*** ***Explain the status register of PIC uC.***

**Ans:**

**#Program:**

```
#include <at89c51xd2.h>
#include <stdio.h>
unsigned int k[] = {0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07, 0x7F, 0x6F};
int s;
void MYDEL(void);
int main(void){
P1 = 0x00;
while(1){
for(s = 0; s < 10; s++){
P1 = k[s];
MYDEL();
}
}
}
void MYDEL(void){
int i;
for(i = 0; i < 20000; i++);
}
```




**#Ans:**

The STATUS register is an 8-bit Special Function Register (SFR) that shows the current state of the ALU (Arithmetic Logic Unit) and program execution status. It holds flag bits used for arithmetic operations, comparison results, and processor control.

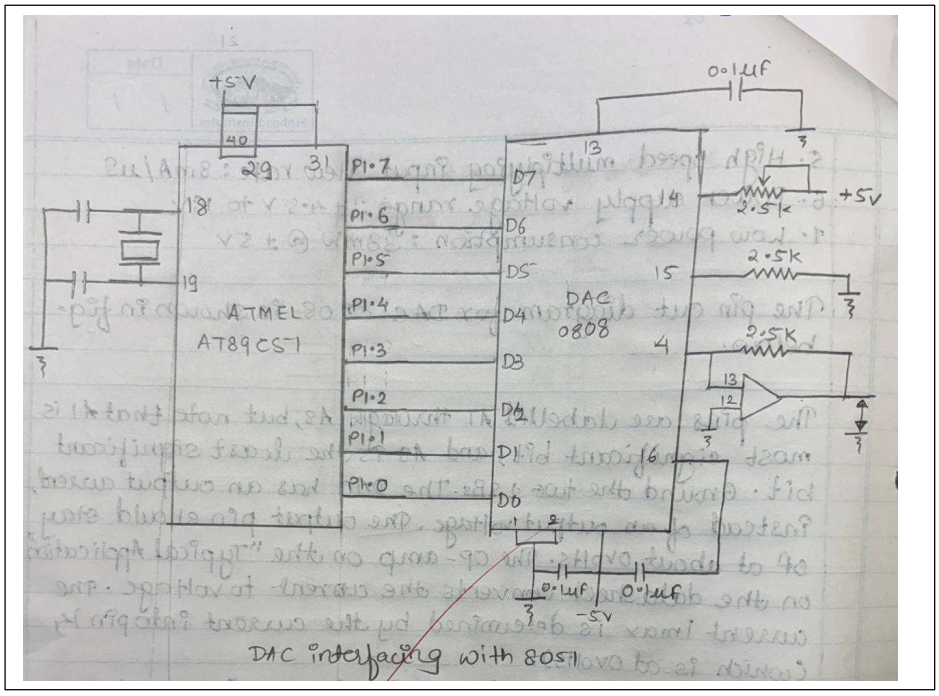
| Bit | Name | Description |
|---|---|---|
| **7** | — | Unimplemented (Read as '0') |
| **6** | **IRP** | Register Bank Select bit (for indirect addressing, selects upper/lower bank). |
| **5** | **RP1** | Register Bank Select bit 1. |
| **4** | **RP0** | Register Bank Select bit 0. |
| **3** | **TO** | Watchdog Time-out bit — 1 = No time-out, 0 = Time-out occurred. |
| **2** | **PD** | Power-down bit — 1 = After power-on, 0 = Woke from sleep. |
| **1** | **Z** | Zero flag — 1 = Result is zero, 0 = Result is non-zero. |
| **0** | **C** | Carry/Borrow flag — 1 = Carry, 0 = No Carry. |

**iv. Write C-program for 8051 uC for Waveform Generation using DAC 0808.**
***Que. Explain DAC 0808 in detail and write down the features of it.***

**Ans:**

**#Program:**

```
#include <reg51.h>
#include <intrins.h>
void main(void){
unsigned char Wavevalu[12] = {128,192,238,255,238,192,128,64,17,0,17,64};
int i;
while(1){
for(i = 0; i < 12; i++){
P1 = Wavevalu[i];
_nop_();
_nop_();
_nop_();
_nop_();
_nop_();
_nop_();
}
}
}
```



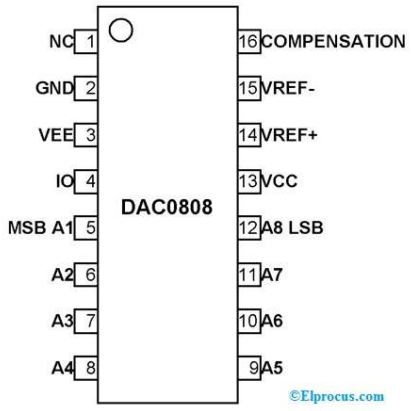

**#Ans:**

DAC0808 is an 8-bit R–2R ladder type DAC that converts digital input (8 bits) into a proportional analog current output. It needs a reference voltage (Vref) and an op-amp to convert current to voltage.

Main Features: 8-bit resolution, R–2R ladder network, Conversion time ≈ 100 ns, Operates on ±5V supply, TTL/CMOS compatible, Low power consumption, Monotonic output

$$I_{OUT} = I_{REF} \times \frac{Digital\ Input}{256}$$

Applications: Waveform generation**,** Audio signal output**,** Control and instrumentation systems



**ix. Write a C program for Interfacing of 16 * 2 LCD Display with 8051 uC to show the message "Welcome" on first line and "NBNSTIC, Pune" on second line of LCD Display.**
***Que. Explain the pins of LCD in Detail.***

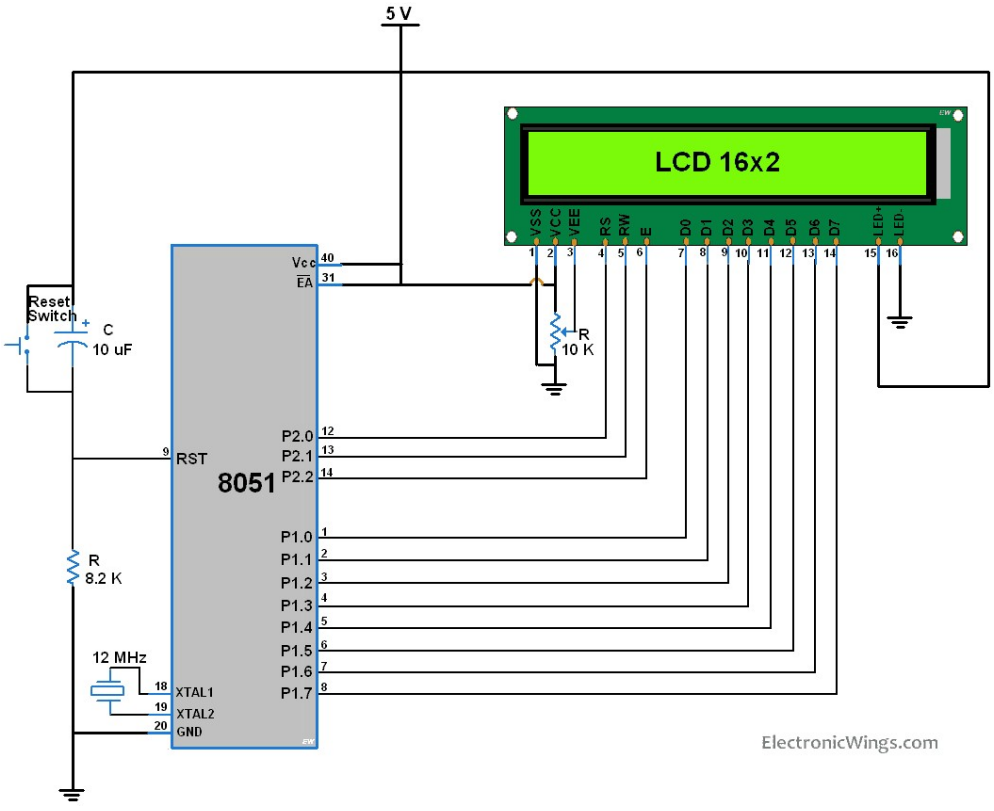
**Ans:**

```
#include <reg51.h>
#define LCD P2
sbit RS = P3^0;
sbit RW = P3^1;
sbit EN = P3^2;
void delay(unsigned int);
void lcd_cmd(unsigned char);
void lcd_data(unsigned char);
void lcd_init(void);
void lcd_string(char *str);
void main(void){
lcd_init();
lcd_cmd(0x80);
lcd_string("Welcome");
lcd_cmd(0xC0);
lcd_string("NBNSTIC, Pune");
while(1);
}
void lcd_init(void){
lcd_cmd(0x38);
lcd_cmd(0x0C);
lcd_cmd(0x06);
lcd_cmd(0x01);
delay(5);
}
void lcd_cmd(unsigned char cmd){
LCD = cmd;
RS = 0;
RW = 0;
EN = 1;
delay(1);
EN = 0;}
```

```
void lcd_data(unsigned char dat){
LCD = dat;
RS = 1;
RW = 0;
EN = 1;
delay(1);
EN = 0;
}
void lcd_string(char *str){
while(*str){
lcd_data(*str++);
}
}
void delay(unsigned int time){
unsigned int i, j;
for(i = 0; i < time; i++)
for(j = 0; j < 1275; j++);
}
```

**#Ans:**

| Pin | Pin Name | Description / Function |
|---|---|---|
| 1 | VSS | Connected to system ground (0V). |
| 2 | VCC | Connected to +5V supply. |
| 3 | VEE | Connected to a variable resistor (potentiometer) to adjust display contrast. |
| 4 | RS (Register Select) | Selects register: 0 = Command Register 1 = Data Register. |
| 5 | RW (Read/Write) | Selects mode: 0 = Write to LCD 1 = Read from LCD. |
| 6 | EN (Enable) | Used to latch data into LCD; a high-to-low pulse executes command/data. |
| 7–14 | D0–D7 | 8-bit bidirectional data lines for sending commands or data. Can also use only D4–D7 for 4-bit mode. |
| 15 | LED+ / A | Anode terminal for LED backlight (connected to +5V through resistor). |
| 16 | LED− / K | Cathode terminal for backlight (connected to ground). |



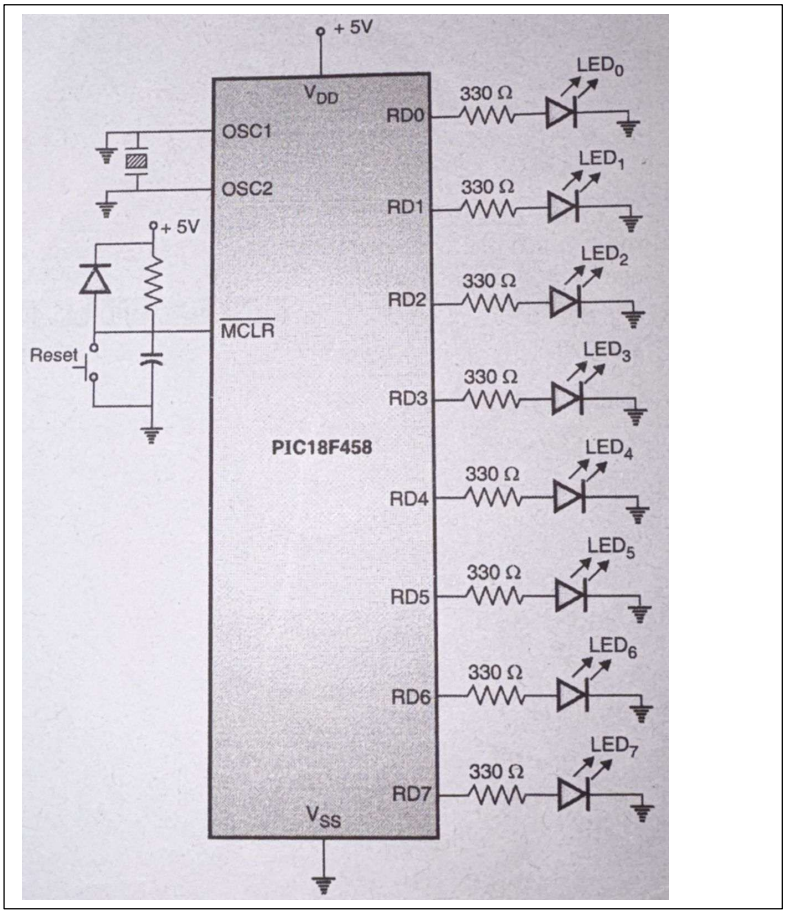
**v. Write a C program for interfacing of LED to PIC uC. Show Flashing LED.**
***Que. Explain the BOD mode of PIC18FXXXX.***

**Ans**.

**#Program:**

#C source code (Generate in MPLABx and copy paste- Continue next right after source code)

```
void msdelay (unsigned int time);
void main(){
INTCON2bits.RBPU=0;
ADCON1=0X0F;
TRISC = 0X00;
while(1){
PORTC= 0XAA;
msdelay (250);
PORTC= 0X55;
msdelay(250);
}
}
void msdelay (unsigned int time) {
unsigned int i,j;
for(i=0;i<time;i++) {
for(j=0;j<710;j++);
}
}
```




**#Ans:**

Brown-Out Detect (BOD) or Brown-Out Reset (BOR) is a protection feature in PIC18F microcontrollers. It resets the MCU automatically when the supply voltage (VDD) drops below a certain threshold — ensuring reliable operation.

Purpose: To prevent erratic behavior or data corruption when power supply voltage falls below the safe operating limit.

Working: When VDD < VBOR (preset threshold, e.g., 4.2V or 2.1V depending on configuration), → The microcontroler resets.

When VDD rises above VBOR + hysteresis, → The MCU resumes normal operation.

Controlled by: BOREN = 1 → Brown-out Reset enabled ; BOREN = 0 → Brown-out Reset disabled.

Indication: The RCON register's BOR bit (RCON<0>) indicates if a brown-out reset has occurred.

BOR = 1 → No BOR occurred ; BOR = 0 → BOR occurred (power dipped below threshold).

Features: Protects from low voltage malfunction, Ensures safe system recovery after power restoration, User can enable/disable via configuration bits, Automatic reset — no external circuit needed.

**vi. Write a C program for Interfacing of 16 * 2 LCD Display with PIC 18FXXXX to show the message "Welcome" on first line and "NBNSTIC, Pune" on second line of LCD Display.**
***Que.* *Explain the Port D and Port E of PIC18FXXXX***

**Ans.**

**#Program:**

```
#include <xc.h>
#pragma config FOSC = HS
#pragma config WDT = OFF
#pragma config LVP = OFF
#pragma config PBADEN = OFF
#define LCD_DATA PORTD
#define ctrl PORTE
#define rs PORTEbits.RE0
#define rw PORTEbits.RE1
#define en PORTEbits.RE2
void init_LCD(void);
void LCD_command(unsigned char cmd);
void LCD_data(unsigned char data);
void LCD_write_string(char *str);
void msdelay(unsigned int time);
void main(void){
char var1[] = "Welcome";
char var2[] = "NBNSTIC, Pune";
ADCON1 = 0x0F;
TRISD = 0x00;
TRISE = 0x00;
init_LCD();
msdelay(50);
LCD_write_string(var1);
msdelay(15);
LCD_command(0xC0);
LCD_write_string(var2);
while(1);
}
void msdelay(unsigned int time){
unsigned int i, j;
for(i = 0; i < time; i++)
for(j = 0; j < 275; j++);
}
void init_LCD(void){
LCD_command(0x38);
 msdelay(15);
LCD_command(0x01);
msdelay(15);
LCD_command(0x0C);
msdelay(15);
LCD_command(0x80);
msdelay(15);
}
void LCD_command(unsigned char cmd){
LCD_DATA = cmd;
rs = 0;
rw = 0;
en = 1;
msdelay(2);
en = 0;
}
void LCD_data(unsigned char data){
LCD_DATA = data;
 rs = 1;
 rw = 0;
en = 1;
msdelay(2);
en = 0;
}
void LCD_write_string(char *str){
int i = 0;
while (str[i] != '\0'){
LCD_data(str[i]);
msdelay(15);
i++;
}}
```

**#Ans:**

1. PORT D :

Pins: RD0–RD7

Functions:

General-Purpose I/O (Input/Output):Each pin can be used as a digital input or output when PSP mode is disabled.

Parallel Slave Port (PSP) Interface (with PORT E): When PSP mode is enabled, PORT D acts as an 8-bit bidirectional data bus. Used to connect PIC to a microprocessor or external memory in parallel slave mode. Controlled by signals from PORT E (RD, WR, CS).

Registers:

TRISD: Data Direction Register (1 → Input, 0 → Output)

PORTD: Reads logic levels on RD0–RD7

LATD: Latch register (stores output values)

2. PORT E:

Pins: RE0, RE1, RE2 (and sometimes RE3)

Functions:

| Pin | Digital Function | Alternate Function |
|---|---|---|
| RE0 | I/O | **/RD** – Read control (used in PSP mode) |
| RE1 | I/O | **/WR** – Write control |
| RE2 | I/O | **/CS** – Chip Select control |
| RE3 | Input only | **/MCLR** (Master Clear / Reset input) or **Vpp** (programming voltage) |

Registers:

TRISE: Data Direction Register

PORTE: Reads logic levels

LATE: Latch for outputs

ADCON1 / ADCON0: Control whether RE0–RE2 function as analog inputs or digital I/O.

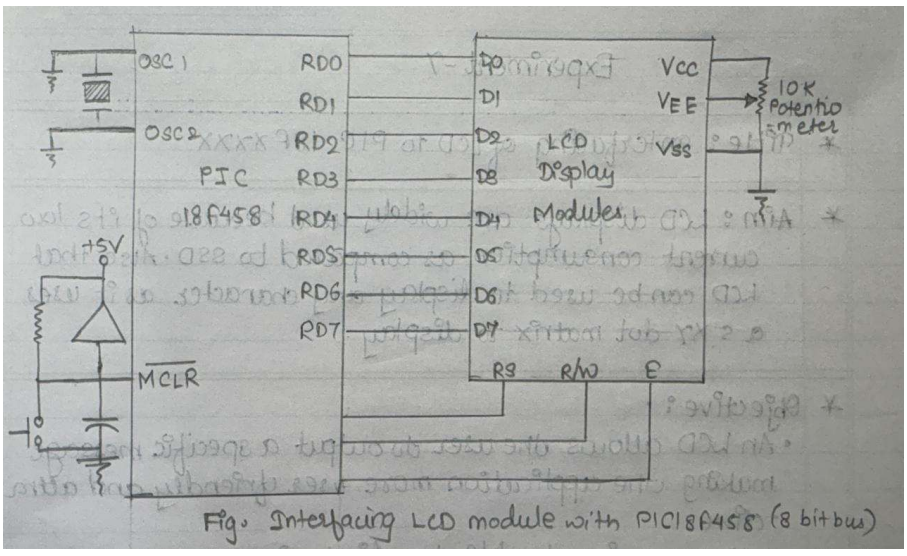


Fig. Interfacing LCD module with PIC18F458 (8 bit bus)

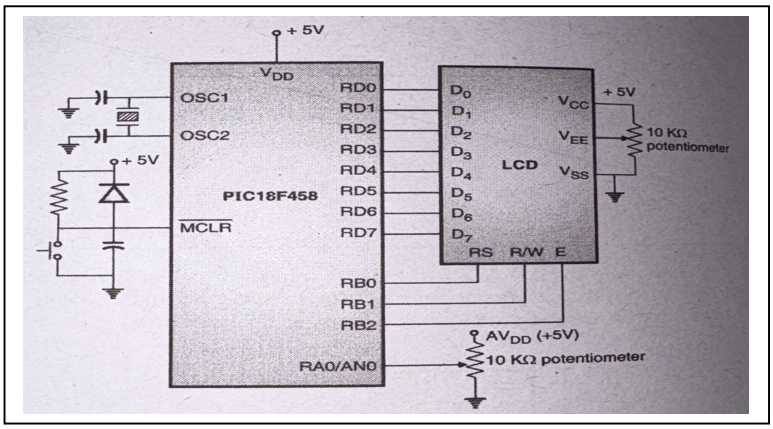
**vii. Write a C program for Interfacing of PIC uC for analog voltage 0-5V to internal ADC and display value on LCD Display.**
***Que.** Explain Port A of PIC18FXXXX*

**Ans.**

**#Program:**

```
#include <xc.h>
#pragma config OSC = HS
#pragma config WDT = OFF
#pragma config LVP = OFF
#pragma config PBADEN = OFF
#define LCD_DATA PORTD
#define rs PORTEbits.RE0
#define rw PORTEbits.RE1
#define en PORTEbits.RE2
void msdelay(unsigned int);
void init_LCD(void);
void LCD_command(unsigned char);
void LCD_data(unsigned char);
void LCD_write_string(char *str);
void ADC_Init(void);
void Start_Conversion(void);
unsigned int Get_ADC_Result(void);
void main(void){
char msg1[] = "On-chip ADC Prog.";
char msg2[] = "ADC VOLT:";
unsigned int adc_val;
unsigned long voltage;
unsigned char thousands, hundreds, tens, ones;
ADCON1 = 0x0F;
TRISD = 0x00;
TRISE = 0x00;
ADC_Init();
init_LCD();
LCD_write_string(msg1);
msdelay(1000);
LCD_command(0x01);
LCD_write_string(msg2);
while(1){
Start_Conversion();
adc_val = Get_ADC_Result();
voltage = ((unsigned long)adc_val * 5000) /
1023;
LCD_command(0xC9);
thousands = (voltage / 1000) % 10;
hundreds = (voltage / 100) % 10;
tens = (voltage / 10) % 10;
ones = voltage % 10;
LCD_data(thousands + '0');
LCD_data('.');
LCD_data(hundreds + '0');
LCD_data(tens + '0');
LCD_data(ones + '0');
msdelay(500);
}
}
void ADC_Init(void){
ADCON0 = 0x01;
ADCON1 = 0x0E;
ADCON2 = 0b10001110;
}
void Start_Conversion(void){
 ADCON0bits.GO = 1;
}
unsigned int Get_ADC_Result(void){
while(ADCON0bits.GO);
return ((ADRESH << 8) + ADRESL);
}
void msdelay(unsigned int time){
unsigned int i, j;
for(i = 0; i < time; i++)
for(j = 0; j < 710; j++);
}
void init_LCD(void){
LCD_command(0x38);
```

```
msdelay(15);
LCD_command(0x01);
msdelay(15);
LCD_command(0x0C);
msdelay(15);
LCD_command(0x80);
msdelay(15);
}
void LCD_command(unsigned char cmd){
LCD_DATA = cmd;
rs = 0;
rw = 0;
en = 1;
msdelay(2);
en = 0;
}
void LCD_data(unsigned char data){
LCD_DATA = data;
rs = 1;
rw = 0;
en = 1;
msdelay(2);
en = 0;
}
void LCD_write_string(char *str){
while(*str){
LCD_data(*str);
str++;
}
}
```




| Pin | Description / Use |
|---|---|
| RA0/AN0 | ADC input or digital I/O |
| RA1/AN1 | ADC input or digital I/O |
| RA2/AN2/VREF– | Selectable analog/digital pin |
| RA3/AN3/VREF+ | Selectable analog/digital pin |
| RA4/T0CKI | Open-drain digital pin or external clock for Timer0 |
| RA5/AN4/SS | Multipurpose analog/digital pin |
| RA6/OSC2/CLKO | Used in crystal mode or as I/O (in some variants) |
| RA7/OSC1/CLKI | Used for connecting external clock source |

**#Ans:**

PORTA is an 8-bit bidirectional I/O port. It can be used for general-purpose digital input/output as well as analog input for the ADC (Analog-to-Digital Converter) module.
Registers Used: TRISA → Direction control register (1 = Input, 0 = Output), PORTA → Reads the current logic levels on pins, LATA → Output latch register (used to write output values), ADCON1 → Determines which pins act as analog or digital.
Key Features: Supports both digital I/O and analog inputs, Can provide reference voltages to ADC (VREF+ / VREF–), RA4 is open-drain — needs an external pull-up resistor, Configurable using TRISA and ADCON1 registers.

viii. **Write a C program for Interfacing of PIC uC with DC Motor and show the generation of PWM signal for controlling DC Motor.**
***Que.*** ***Explain the Interrupt structure of PIC18FXXXX.***

Ans.

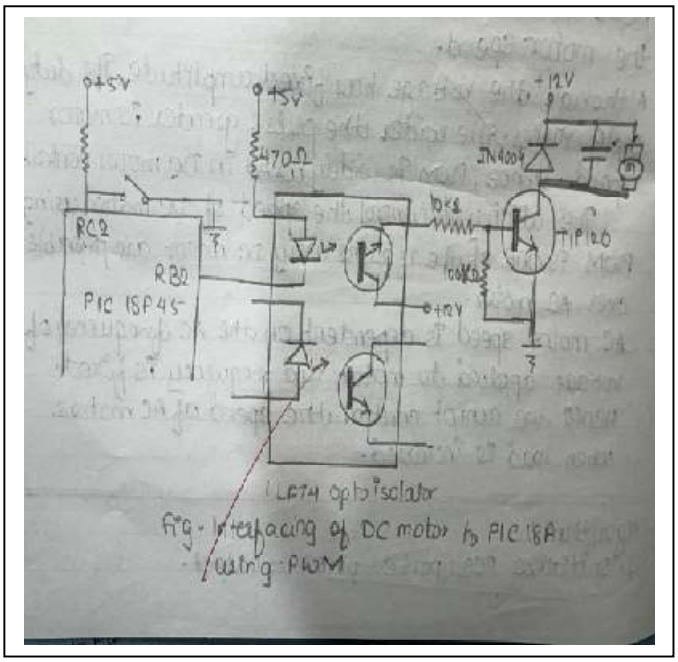


fig - Interfacing of DC motor to PIC 18F using PWM

```
#Program:
#include <xc.h>
#pragma config OSC = HS
#pragma config WDT = OFF
#pragma config LVP = OFF
#pragma config PBADEN = OFF
void myMsDelay(unsigned int time);
void main(void){
TRISCbits.TRISC0 = 0;
TRISCbits.TRISC1 = 0;
TRISCbits.TRISC2 = 0;
PR2 = 0x4E;
CCP1CON = 0x0C;
T2CON = 0x07;
PORTCbits.RC0 = 1;
PORTCbits.RC1 = 0;
while (1){
CCP1CONbits.DC1B1 = 1;
CCP1CONbits.DC1B0 = 0;
CCPR1L = 0x3E;
myMsDelay(2000);
CCP1CONbits.DC1B1 = 1;
CCP1CONbits.DC1B0 = 1;
CCPR1L = 0x2E;
myMsDelay(2000);
CCP1CONbits.DC1B1 = 0;
CCP1CONbits.DC1B0 = 1;
CCPR1L = 0x1F;
myMsDelay(2000);
CCP1CONbits.DC1B1 = 0;
CCP1CONbits.DC1B0 = 0;
CCPR1L = 0x0F;
myMsDelay(2000);
```

```
}
}
void myMsDelay(unsigned int time){
unsigned int i, j;
for (i = 0; i < time; i++)
for (j = 0; j < 275; j++);
}
```

**#Ans:**

An interrupt is an event that temporarily halts the normal program execution, allowing the CPU to execute a special Interrupt Service Routine (ISR) to handle that event. After servicing, the CPU returns to the main program.
Types:
Hardware Interrupts – Generated by internal or external peripherals (Timer, ADC, UART, etc.).

Software Interrupts – Generated by software instructions.

Interrupt Priority Levels:

High Priority- Urgent tasks (executed immediately)

Low Priority- Less urgent tasks (executed only when no high-priority interrupt is active)

Interrupt Registers:

| Register | Function |
|---|---|
| INTCON | Controls global and peripheral interrupt enable bits |
| INTCON2 | Configures edge selection and priority |
| INTCON3 | Manages external interrupts (INT1, INT2) |
| PIRx | Holds interrupt flags (indicates if event occurred) |
| PIEx | Holds interrupt enable bits |
| IPx | Defines interrupt priority (High/Low) |

Steps to Enable an Interrupt:

- Set the interrupt enable bit (PIE register).
- Clear the interrupt flag (PIR register).
- Set GIE (Global Interrupt Enable) in INTCON.
- Write the ISR to handle the event.