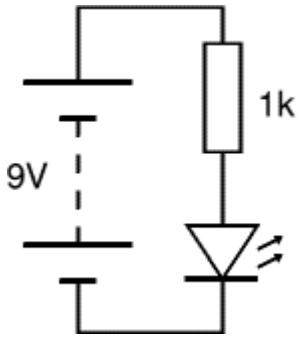
به نام خدا

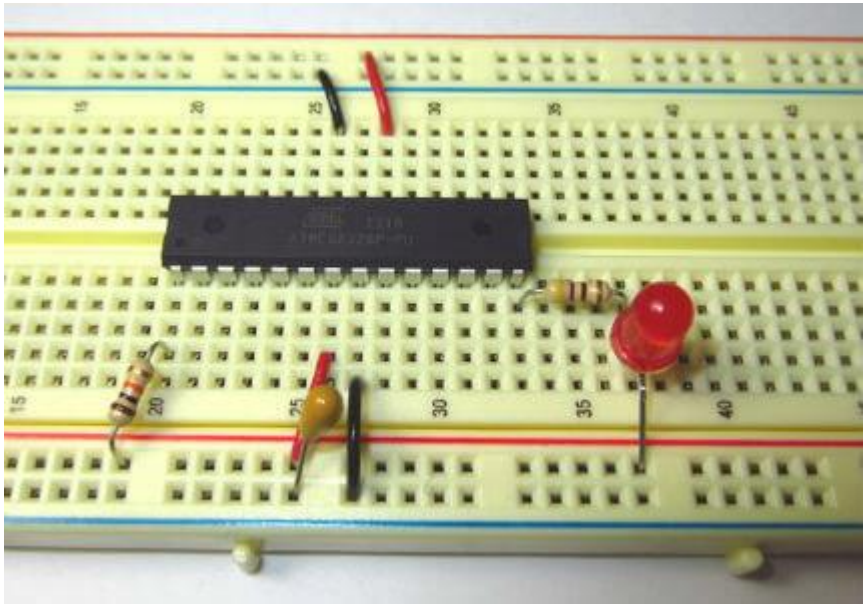
LCD & Keyboard

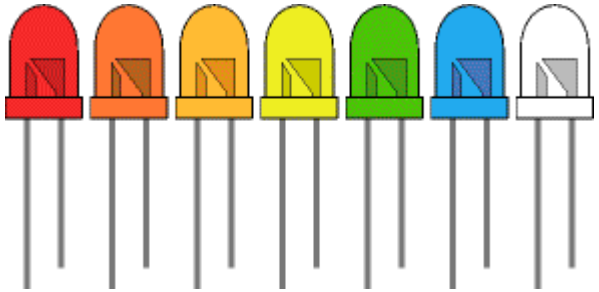
آشنایی با عملکرد و نحوه استفاده

Dr. Aref Karimiafshar
A.karimiafshar@ec.iut.ac.ir

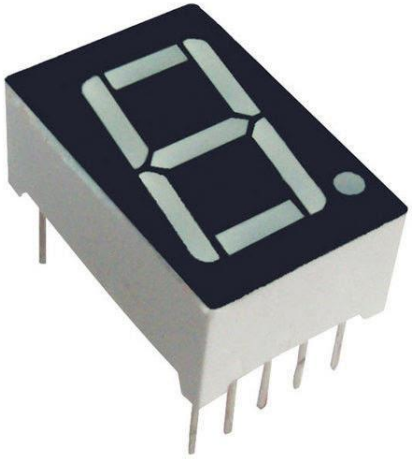
نمایش خروجی

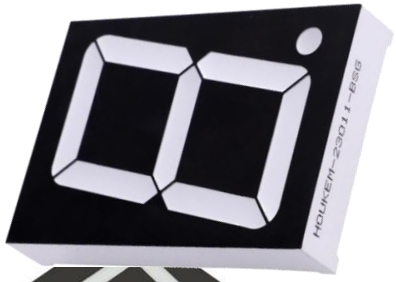


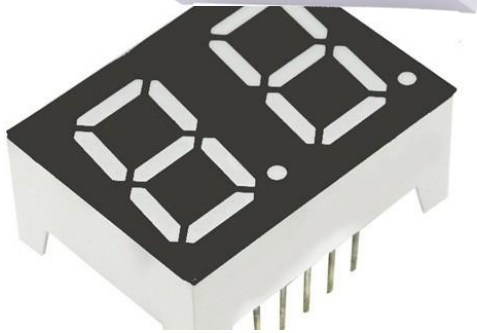


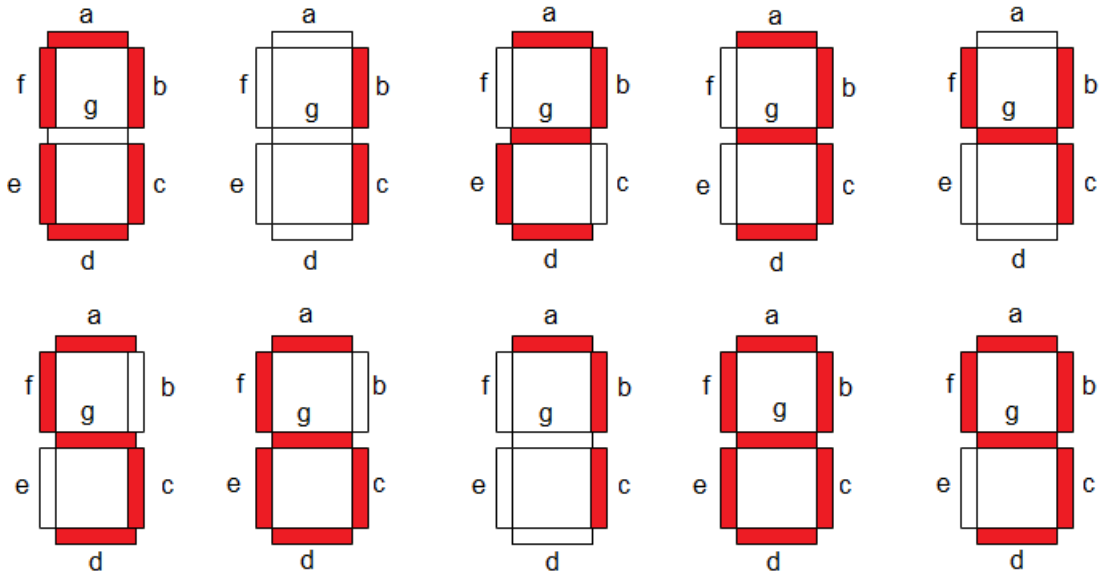


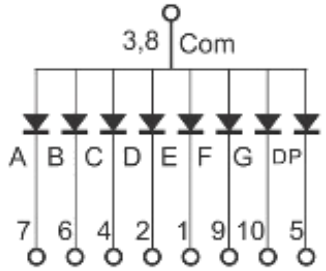
نمایش خروجی (7-Segment)



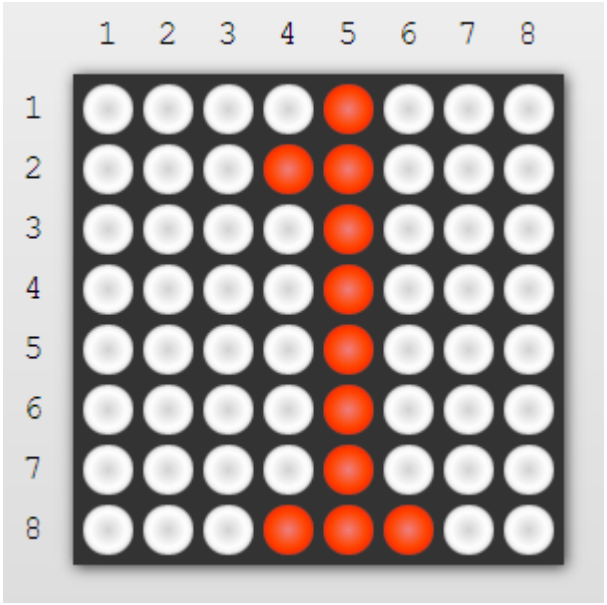


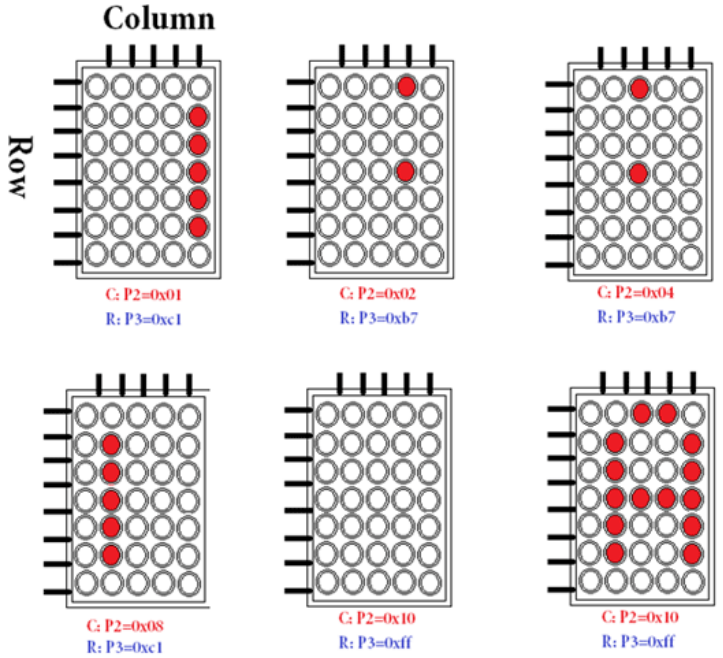


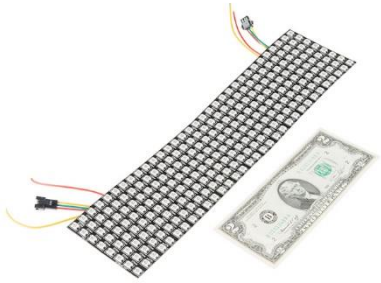


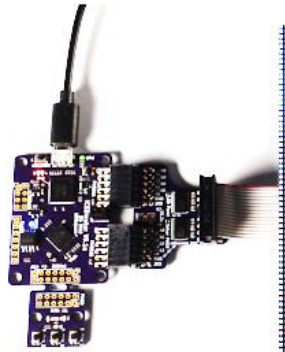


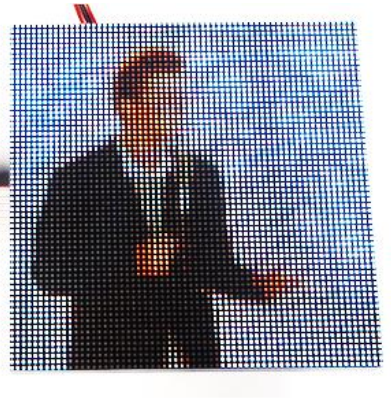
نمایش خروجی (Dot Matrix)



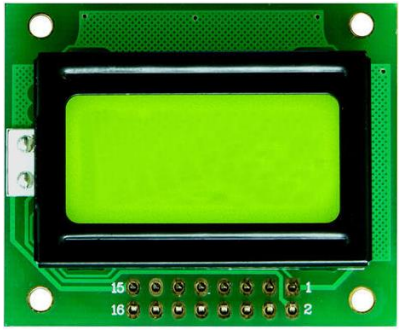




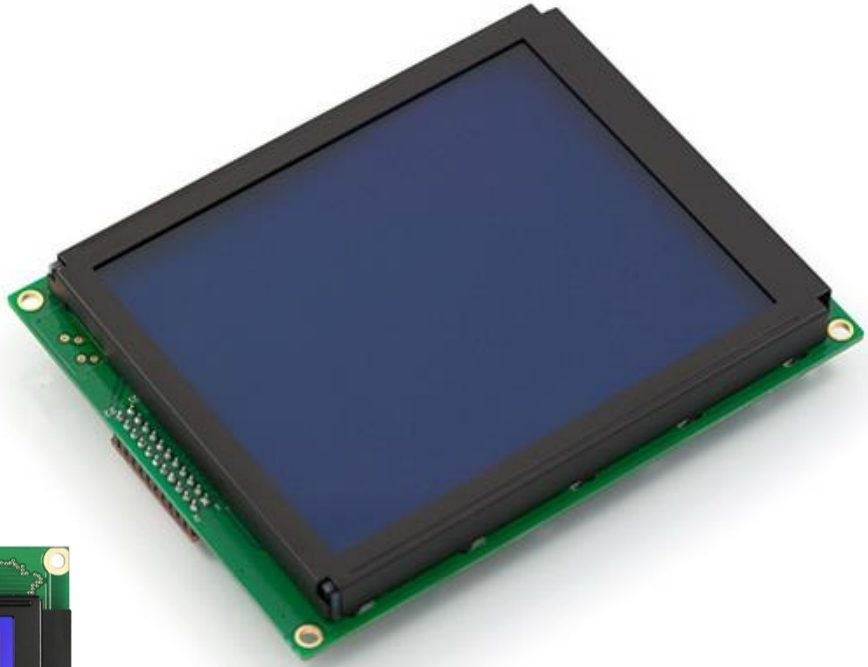




نمایش خروجی (LCD)



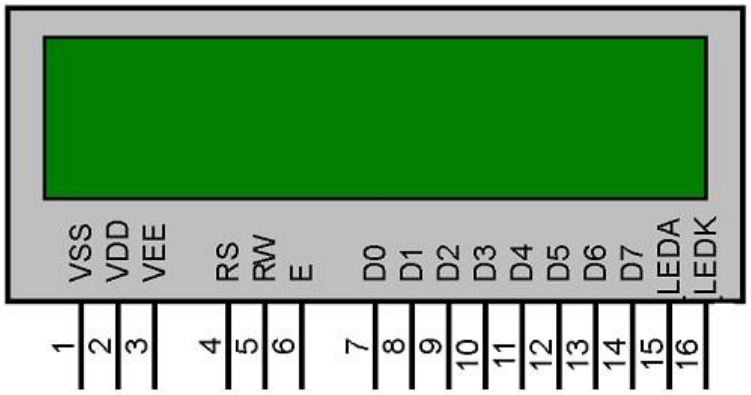
نمایش خروجی (GLCD)



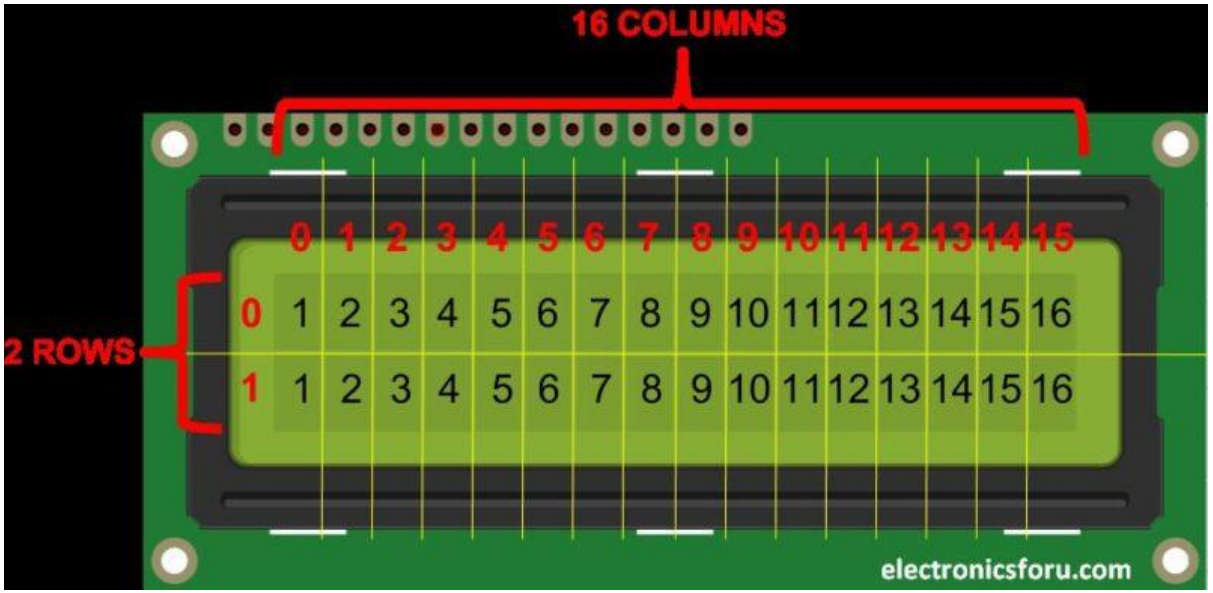
Widespread use of LCD

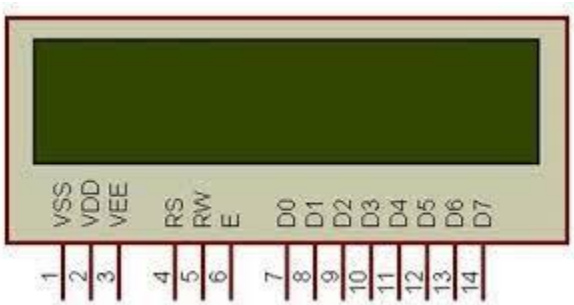
In recent years the LCD is finding widespread use replacing LEDs (seven-segment LEDs or other multisegment LEDs). This is due to the following reasons:

1. The declining prices of LCDs.
2. The ability to display numbers, characters, and graphics. This is in contrast to LEDs, which are limited to numbers and a few characters.
3. Incorporation of a refreshing controller into the LCD, thereby relieving the CPU of the task of refreshing the LCD. In contrast, the LED must be refreshed by the CPU (or in some other way) to keep displaying the data.
4. Ease of programming for characters and graphics.

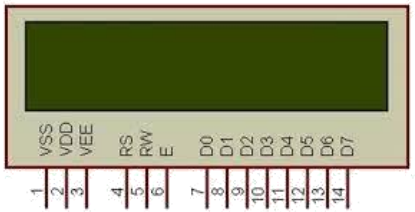


14 Pins LCD


16 COLUMNS
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
2 ROWS
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
1 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
electronicsforu.com


VSS
VDD
VEE
RS
RW
E
D0
D1
D2
D3
D4
D5
D6
D7
1 2 3 4 5 6 7 8 9 10 11 12 13 14

LCD Pin Description



V_{CC}, V_{SS}, and V_{EE}

While V_{CC} and V_{SS} provide +5 V and ground, respectively, V_{EE} is used for controlling LCD contrast.

RS, register select

There are two very important registers inside the LCD. The RS pin is used for their selection as follows. If RS = 0, the instruction command code register is selected, allowing the user to send commands such as clear display, cursor at home, and so on. If RS = 1 the data register is selected, allowing the user to send data to be displayed on the LCD.

R/W, read/write

R/W input allows the user to write information to the LCD or read information from it. R/W = 1 when reading; R/W = 0 when writing.

Pin Descriptions for LCD

Pin	Symbol	I/O	Description
1	V_{SS}	--	Ground
2	V_{CC}	--	+5 V power supply
3	V_{EE}	--	Power supply to control contrast
4	RS	I	RS = 0 to select command register, RS = 1 to select data register
5	R/W	I	R/W = 0 for write, R/W = 1 for read
6	E	I/O	Enable
7	DB0	I/O	The 8-bit data bus
8	DB1	I/O	The 8-bit data bus
9	DB2	I/O	The 8-bit data bus
10	DB3	I/O	The 8-bit data bus
11	DB4	I/O	The 8-bit data bus
12	DB5	I/O	The 8-bit data bus
13	DB6	I/O	The 8-bit data bus
14	DB7	I/O	The 8-bit data bus

LCD Pin Description

E, enable

The enable pin is used by the LCD to latch information presented to its data pins. When data is supplied to data pins, a high-to-low pulse must be applied to this pin in order for the LCD to latch in the data present at the data pins. This pulse must be a minimum of 450 ns wide.

D0–D7

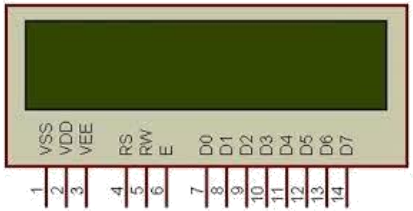
The 8-bit data pins, D0–D7, are used to send information to the LCD or read the contents of the LCD's internal registers.

To display letters and numbers, we send ASCII codes for the letters A–Z, a–z, and numbers 0–9 to these pins while making RS = 1.

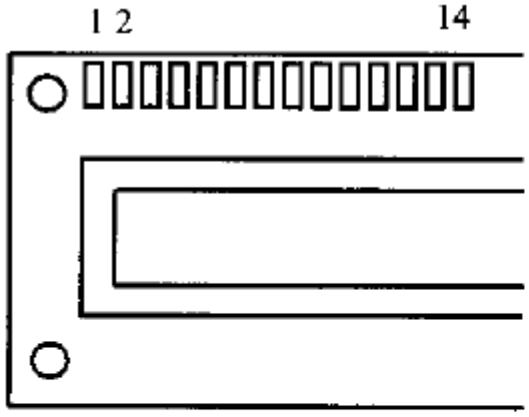
There are also instruction command codes that can be sent to the LCD to clear the display or force the cursor to the home position or blink the cursor.

Pin Descriptions for LCD

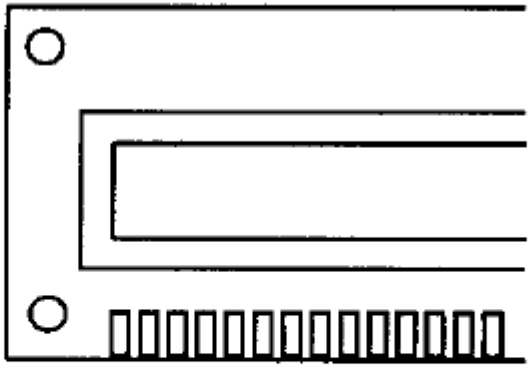
Pin	Symbol	I/O	Description
1	V_{SS}	--	Ground
2	V_{CC}	--	+5 V power supply
3	V_{EE}	--	Power supply to control contrast
4	RS	I	RS = 0 to select command register, RS = 1 to select data register
5	R/W	I	R/W = 0 for write, R/W = 1 for read
6	E	I/O	Enable
7	DB0	I/O	The 8-bit data bus
8	DB1	I/O	The 8-bit data bus
9	DB2	I/O	The 8-bit data bus
10	DB3	I/O	The 8-bit data bus
11	DB4	I/O	The 8-bit data bus
12	DB5	I/O	The 8-bit data bus
13	DB6	I/O	The 8-bit data bus
14	DB7	I/O	The 8-bit data bus



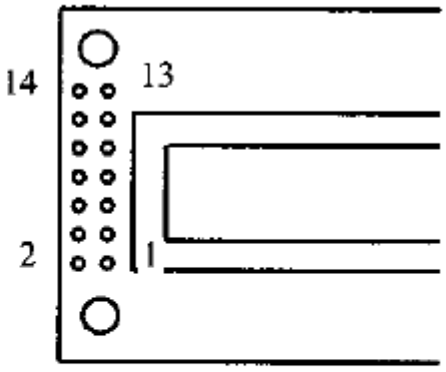
LCD Pin Description



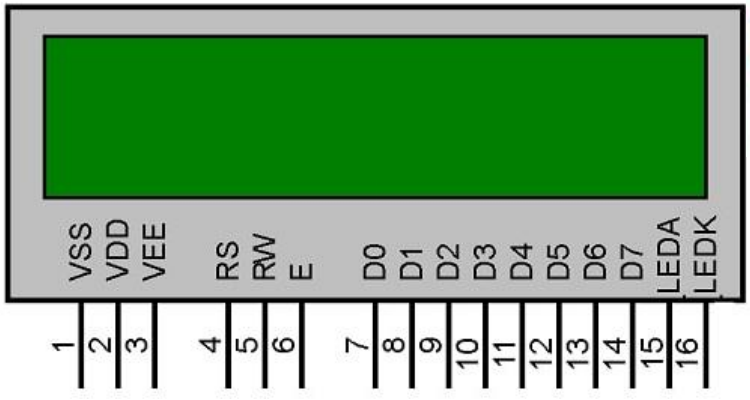
DMC1610A
DMC1606C
DMC16117
DMC16128
DMC16129
DMC1616433
DMC20434

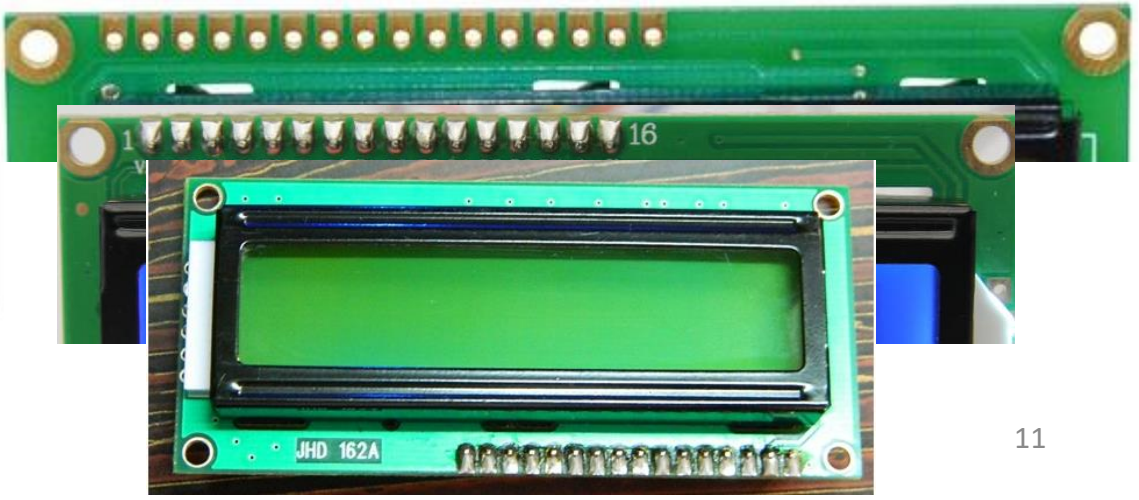


DMC16106B
DMC16207
DMC16230
DMC20215
DMC32216



DMC20261
DMC24227
DMC24138
DMC32132
DMC32239
DMC40131
DMC40218





Sending commands and Data to LCD

To send data and commands to LCDs you should do the following steps. Notice that steps 2 and 3 can be repeated many times:

1. Initialize the LCD.
2. Send any of the commands to the LCD.
3. Send the character to be shown on the LCD.

Initializing the LCD

To initialize the LCD for 5 × 7 matrix and 8-bit operation, the following sequence of commands should be sent to the LCD: 0x38, 0x0E, and 0x01. Next we will show how to send a command to the LCD. After power-up you should wait about 15 ms before sending initializing commands to the LCD. If the LCD initializer function is not the first function in your code you can omit this delay.

Sending commands and Data to LCD

Sending commands to the LCD

To send any of the commands from the Table to the LCD, make pins RS and R/W = 0 and put the command number on the data pins (D0–D7). Then send a high-to-low pulse to the E pin to enable the internal latch of the LCD. Notice that after each command you should wait about 100 µs to let the LCD module run the command. Clear LCD and Return Home commands are exceptions to this rule. After the 0x01 and 0x02 commands you should wait for about 2 ms.

Sending data to the LCD

To send data to the LCD, make pins RS = 1 and R/W = 0. Then put the data on the data pins (D0–D7) and send a high-to-low pulse to the E pin to enable the internal latch of the LCD. Notice that after sending data you should wait about 100 µs to let the LCD module write the data on the screen.

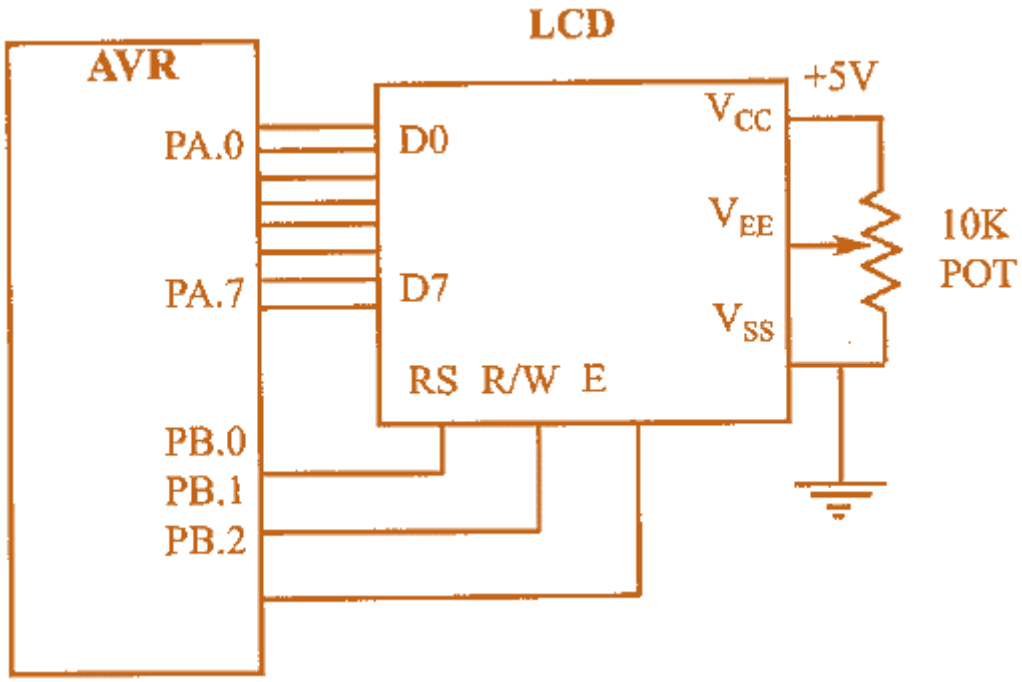
LCD Command Codes

LCD Command Codes

Code (Hex)	Command to LCD Instruction Register
1	Clear display screen
2	Return home
4	Decrement cursor (shift cursor to left)
6	Increment cursor (shift cursor to right)
5	Shift display right
7	Shift display left
8	Display off, cursor off
A	Display off, cursor on
C	Display on, cursor off
E	Display on, cursor blinking
F	Display on, cursor blinking
10	Shift cursor position to left
14	Shift cursor position to right
18	Shift the entire display to the left
1C	Shift the entire display to the right
80	Force cursor to beginning of 1st line
C0	Force cursor to beginning of 2nd line
28	2 lines and 5 × 7 matrix (D4–D7, 4-bit)
38	2 lines and 5 × 7 matrix (D0–D7, 8-bit)

Example

write “Hi” on the LCD using 8-bit data.



Example cnt.

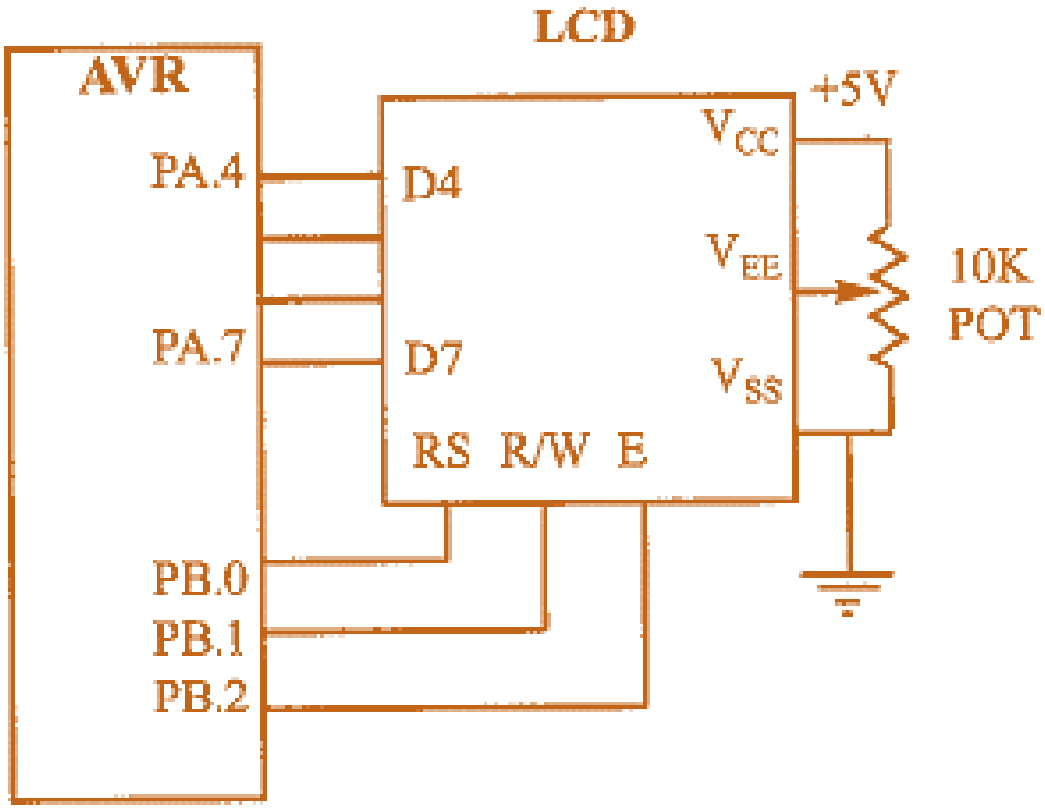
```
.INCLUDE "M32DEF.INC"
.EQU     LCD_DPRT = PORTA      ;LCD DATA PORT
.EQU     LCD_DDDR = DDRA       ;LCD DATA DDR
.EQU     LCD_DPIN = PINA       ;LCD DATA PIN
.EQU     LCD_CPRT = PORTB      ;LCD COMMANDS PORT
.EQU     LCD_CDDR = DDRB       ;LCD COMMANDS DDR
.EQU     LCD_CPIN = PINB       ;LCD COMMANDS PIN
.EQU     LCD_RS = 0            ;LCD RS
.EQU     LCD_RW = 1            ;LCD RW
.EQU     LCD_EN = 2            ;LCD EN

         LDI   R21,HIGH(RAMEND)
         OUT   SPH,R21         ;set up stack
         LDI   R21,LOW(RAMEND)
         OUT   SPL,R21

         LDI   R21,0xFF;
         OUT   LCD_DDDR, R21   ;LCD data port is output
         OUT   LCD_CDDR, R21   ;LCD command port is output
         CBI   LCD_CPRT,LCD_EN;LCD_EN = 0
         CALL  DELAY_2ms       ;wait for power on
         LDI   R16,0x38        ;init LCD 2 lines,5x7 matrix
         CALL  CMNDWRT         ;call command function
         CALL  DELAY_2ms       ;wait 2 ms
         LDI   R16,0x0E        ;display on, cursor on
         CALL  CMNDWRT         ;call command function
         LDI   R16,0x01        ;clear LCD
         CALL  CMNDWRT         ;call command function
         CALL  DELAY_2ms       ;wait 2 ms
         LDI   R16,0x06        ;shift cursor right
         CALL  CMNDWRT         ;call command function
         LDI   R16,'H'         ;display letter 'H'
         CALL  DATAWRT         ;call data write function
         LDI   R16,'i'         ;display letter 'i'
         CALL  DATAWRT         ;call data write function
HERE:    JMP HERE              ;stay here
```

Example cnt.

```
CMNDWRT:
        OUT   LCD_DPRT,R16        ;LCD data port = R16
        CBI   LCD_CPRT,LCD_RS     ;RS = 0 for command
        CBI   LCD_CPRT,LCD_RW     ;RW = 0 for write
        SBI   LCD_CPRT,LCD_EN     ;EN = 1
        CALL  SDELAY              ;make a wide EN pulse
        CBI   LCD_CPRT,LCD_EN     ;EN=0 for H-to-L pulse
        CALL  DELAY_100us         ;wait 100 us
        RET
DATAWRT:
        OUT   LCD_DPRT,R16        ;LCD data port = R16
        SBI   LCD_CPRT,LCD_RS     ;RS = 1 for data
        CBI   LCD_CPRT,LCD_RW     ;RW = 0 for write
        SBI   LCD_CPRT,LCD_EN     ;EN = 1
        CALL  SDELAY              ;make a wide EN pulse
        CBI   LCD_CPRT,LCD_EN     ;EN=0 for H-to-L pulse
        CALL  DELAY_100us         ;wait 100 us
        RET
;-----------------------------------------------------
SDELAY: NOP
        NOP
        RET
;-----------------------------------------------------
DELAY_100us:
        PUSH  R17
        LDI   R17,60
DR0:    CALL  SDELAY
        DEC   R17
        BRNE  DR0
        POP   R17
        RET
;-----------------------------------------------------
DELAY_2ms:
        PUSH  R17
        LDI   R17,20
LDR0:   CALL  DELAY_100US
        DEC   R17
        BRNE  LDR0
        POP   R17
        RET
```

Example

Sending code or data to the LCD 4 bits at a time



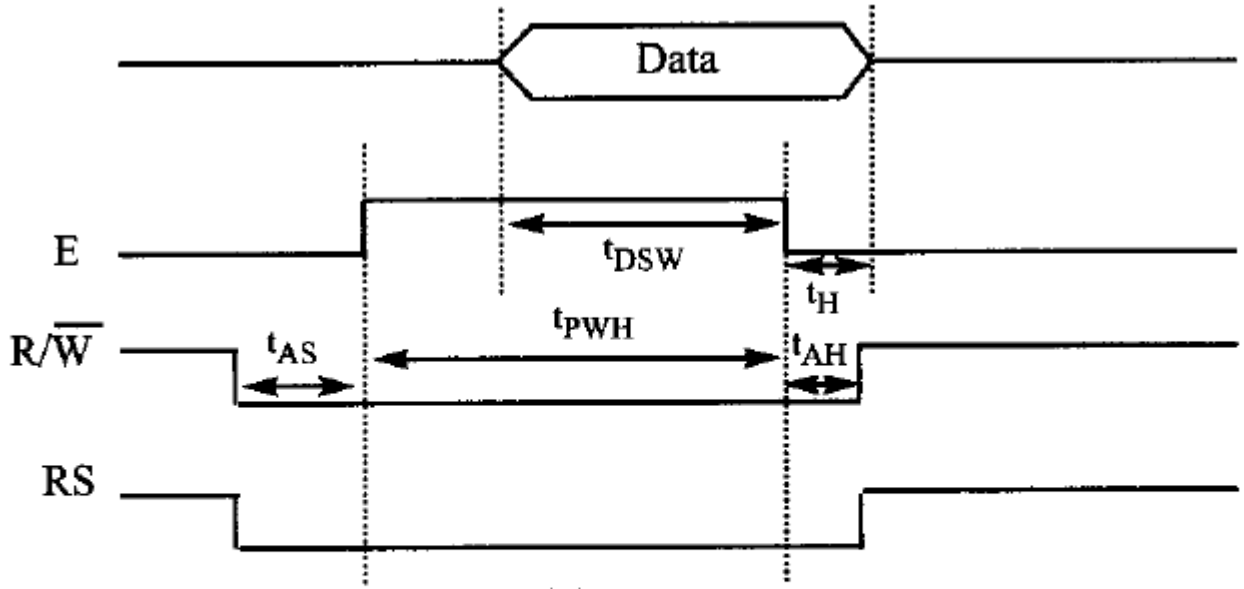
Example cnt.

```
.EQU     LCD_DPRT = PORTA      ;LCD DATA PORT
.EQU     LCD_DDDR = DDRA       ;LCD DATA DDR
.EQU     LCD_DPIN = PINA       ;LCD DATA PIN
.EQU     LCD_CPRT = PORTB      ;LCD COMMANDS PORT
.EQU     LCD_CDDR = DDRB       ;LCD COMMANDS DDR
.EQU     LCD_CPIN = PINB       ;LCD COMMANDS PIN
.EQU     LCD_RS = 0            ;LCD RS
.EQU     LCD_RW = 1            ;LCD RW
.EQU     LCD_EN = 2            ;LCD EN

         LDI   R21,HIGH(RAMEND)
         OUT   SPH,R21         ;set up stack
         LDI   R21,LOW(RAMEND)
         OUT   SPL,R21
         LDI   R21,0xFF;
         OUT   LCD_DDDR, R21   ;LCD data port is output
         OUT   LCD_CDDR, R21   ;LCD command port is output
         LDI   R16,0x33        ;init. LCD for 4-bit data
         CALL  CMNDWRT         ;call command function
         CALL  DELAY_2ms       ;init. hold
         LDI   R16,0x32        ;init. LCD for 4-bit data
         CALL  CMNDWRT         ;call command function
         CALL  DELAY_2ms       ;init. hold
         LDI   R16,0x28        ;init. LCD 2 lines,5×7 matrix
         CALL  CMNDWRT         ;call command function
         CALL  DELAY_2ms       ;init. hold
         LDI   R16,0x0E        ;display on, cursor on
         CALL  CMNDWRT         ;call command function
         LDI   R16,0x01        ;clear LCD
         CALL  CMNDWRT         ;call command function
         CALL  DELAY_2ms       ;delay 2 ms for clear LCD
         LDI   R16,0x06        ;shift cursor right
         CALL  CMNDWRT         ;call command function
         LDI   R16,'H'         ;display letter 'H'
         CALL  DATAWRT         ;call data write function
         LDI   R16,'i'         ;display letter 'i'
         CALL  DATAWRT         ;call data write function
HERE:    JMP HERE              ;stay here
```

Example cnt.

```
CMNDWRT:
        MOV   R27,R16
        ANDI  R27,0xF0
        OUT   LCD_DPRT,R27        ;send the high nibble
        CBI   LCD_CPRT,LCD_RS     ;RS = 0 for command
        CBI   LCD_CPRT,LCD_RW     ;RW = 0 for write
        SBI   LCD_CPRT,LCD_EN     ;EN = 1 for high pulse
        CALL  SDELAY              ;make a wide EN pulse
        CBI   LCD_CPRT,LCD_EN     ;EN=0 for H-to-L pulse
        CALL  DELAY_100us         ;make a wide EN pulse

        MOV   R27,R16
        SWAP  R27                 ;swap the nibbles
        ANDI  R27,0xF0            ;mask D0-D3
        OUT   LCD_DPRT,R27        ;send the low nibble
        SBI   LCD_CPRT,LCD_EN     ;EN = 1 for high pulse
        CALL  SDELAY              ;make a wide EN pulse
        CBI   LCD_CPRT,LCD_EN     ;EN=0 for H-to-L pulse
        CALL  DELAY_100us         ;wait 100 us
        RET
;-----------------------------------------------------
DATAWRT:
        MOV   R27,R16
        ANDI  R27,0xF0
        OUT   LCD_DPRT,R27        ;;send the high nibble
        SBI   LCD_CPRT,LCD_RS     ;RS = 1 for data
        CBI   LCD_CPRT,LCD_RW     ;RW = 0 for write
        SBI   LCD_CPRT,LCD_EN     ;EN = 1 for high pulse
        CALL  SDELAY              ;make a wide EN pulse
        CBI   LCD_CPRT,LCD_EN     ;EN=0 for H-to-L pulse

        MOV   R27,R16
        SWAP  R27                 ;swap the nibbles
        ANDI  R27,0xF0            ;mask D0-D3
        OUT   LCD_DPRT,R27        ;send the low nibble
        SBI   LCD_CPRT,LCD_EN     ;EN = 1 for high pulse
        CALL  SDELAY              ;make a wide EN pulse
        CBI   LCD_CPRT,LCD_EN     ;EN=0 for H-to-L pulse

        CALL  DELAY_100us         ;wait 100 us
        RET
;-----------------------------------------------------
;delay functions are the same as last program and should
;be placed here.
```

LCD Timing Diagram



t_{PWH} = Enable pulse width = 450 ns (minimum)

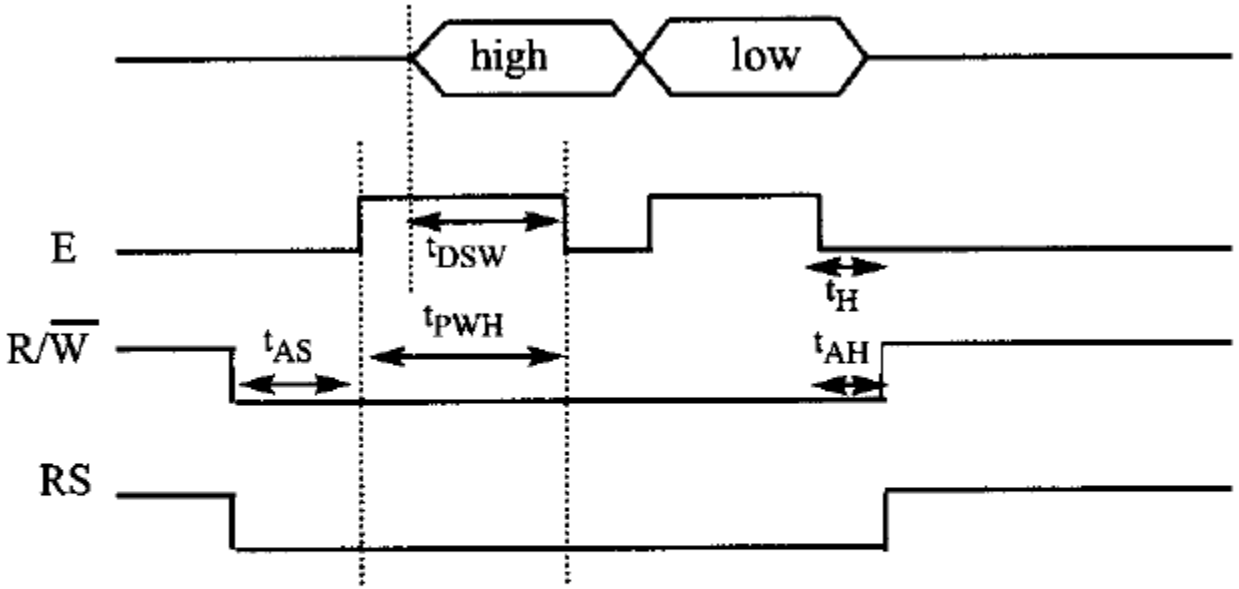
t_{DSW} = Data setup time = 195 ns (minimum)

t_H = Data hold time = 10 ns (minimum)

t_{AS} = Setup time prior to E (going high) for both RS and R/W = 140 ns (minimum)

t_{AH} = Hold time after E has come down for both RS and R/W = 10 ns (minimum)

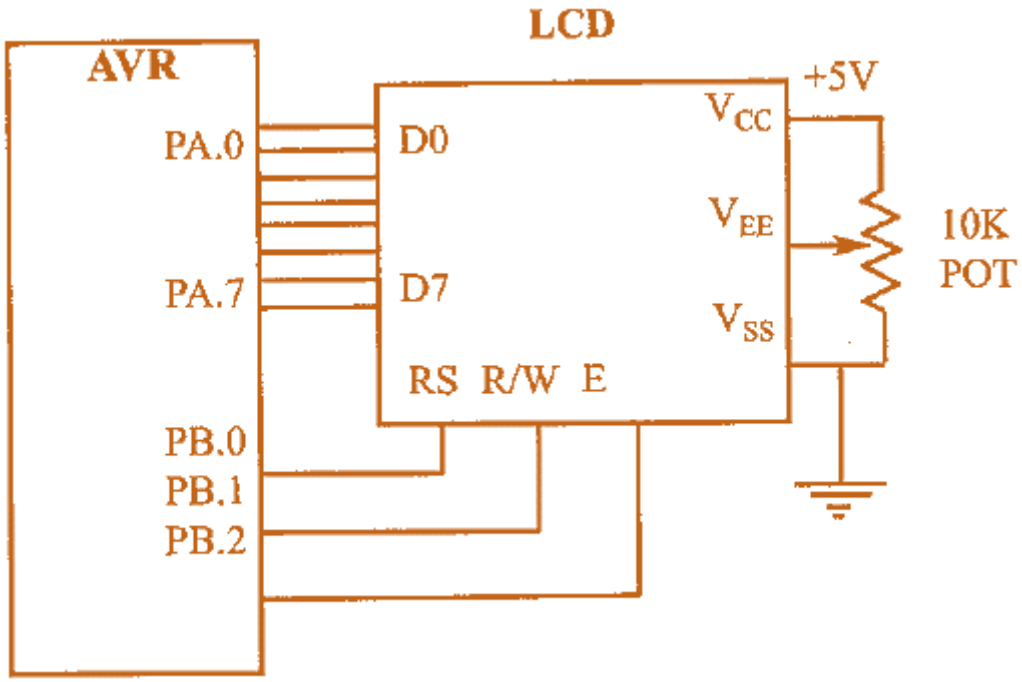
LCD Timing Diagram



t_{PWH} = Enable pulse width = 450 ns (minimum)
t_{DSW} = Data setup time = 195 ns (minimum)
t_H = Data hold time = 10 ns (minimum)
t_{AS} = Setup time prior to E (going high) for both RS and R/W = 140 ns (minimum)
t_{AH} = Hold time after E has come down for both RS and R/W = 10 ns (minimum)

Example (LCD Programming in C)

write on the LCD using 8-bit data.



Example cnt.

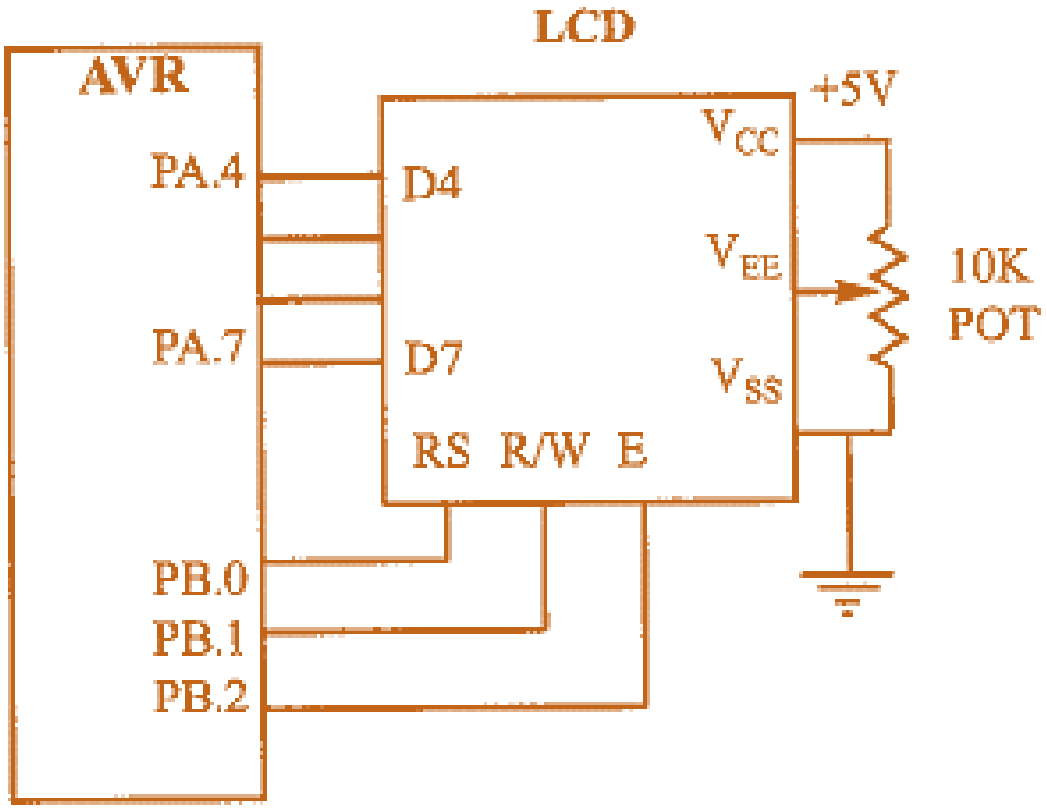
```
#include <avr/io.h>                 //standard AVR header
#include <util/delay.h>             //delay header

#define   LCD_DPRT   PORTA          //LCD DATA PORT
#define   LCD_DDDR   DDRA           //LCD DATA DDR
#define   LCD_DPIN   PINA           //LCD DATA PIN
#define   LCD_CPRT   PORTB          //LCD COMMANDS PORT
#define   LCD_CDDR   DDRB           //LCD COMMANDS DDR
#define   LCD_CPIN   PINB           //LCD COMMANDS PIN
#define   LCD_RS   0                //LCD RS
#define   LCD_RW   1                //LCD RW
#define   LCD_EN   2                //LCD EN

//*************************************************************
void delay_us(unsigned int d)
{
   _delay_us(d);
}

//*************************************************************
void lcdCommand( unsigned char cmnd )
{
   LCD_DPRT = cmnd;                 //send cmnd to data port
   LCD_CPRT &= ~ (1<<LCD_RS);       //RS = 0 for command
   LCD_CPRT &= ~ (1<<LCD_RW);       //RW = 0 for write
   LCD_CPRT |= (1<<LCD_EN);         //EN = 1 for H-to-L pulse
   delay_us(1);                     //wait to make enable wide
   LCD_CPRT &= ~ (1<<LCD_EN);       //EN = 0 for H-to-L pulse
   delay_us(100);                   //wait to make enable wide
}

//*************************************************************
void lcdData( unsigned char data )
{
   LCD_DPRT = data;                 //send data to data port
   LCD_CPRT |= (1<<LCD_RS);         //RS = 1 for data
   LCD_CPRT &= ~ (1<<LCD_RW);       //RW = 0 for write
   LCD_CPRT |= (1<<LCD_EN);         //EN = 1 for H-to-L pulse
   delay_us(1);                     //wait to make enable wide
```

Example cnt.

```
  LCD_CPRT &= ~ (1<<LCD_EN);      //EN = 0 for H-to-L pulse
  delay_us(100);                  //wait to make enable wide
}

//*********************************************************
void lcd_init()
{
  LCD_DDDR = 0xFF;
  LCD_CDDR = 0xFF;

  LCD_CPRT &=~(1<<LCD_EN);        //LCD_EN = 0
  delay_us(2000);                 //wait for init.
  lcdCommand(0x38);               //init. LCD 2 line, 5 x 7 matrix
  lcdCommand(0x0E);               //display on, cursor on
  lcdCommand(0x01);               //clear LCD
  delay_us(2000);                 //wait
  lcdCommand(0x06);               //shift cursor right
}

//*********************************************************
void lcd_gotoxy(unsigned char x, unsigned char y)
{
 unsigned char firstCharAdr[]={ 0x80,0xC0,0x94,0xD4};
 lcdCommand(firstCharAdr[ y-1]  + x - 1);
 delay_us(100);
}

//*********************************************************
void lcd_print( char * str )
{
  unsigned char i = 0;
  while(str[ i] !=0)
  {
    lcdData(str[ i] );
    i++ ;
  }
}

//*********************************************************
int main(void)
{
        lcd_init();
        lcd_gotoxy(1,1);
        lcd_print("The world is but");
        lcd_gotoxy(1,2);
        lcd_print("one country");

        while(1);                       //stay here forever
        return 0;
}
```

Example (LCD Programming in C)

Sending code or data to the LCD 4 bits at a time



Example cnt.

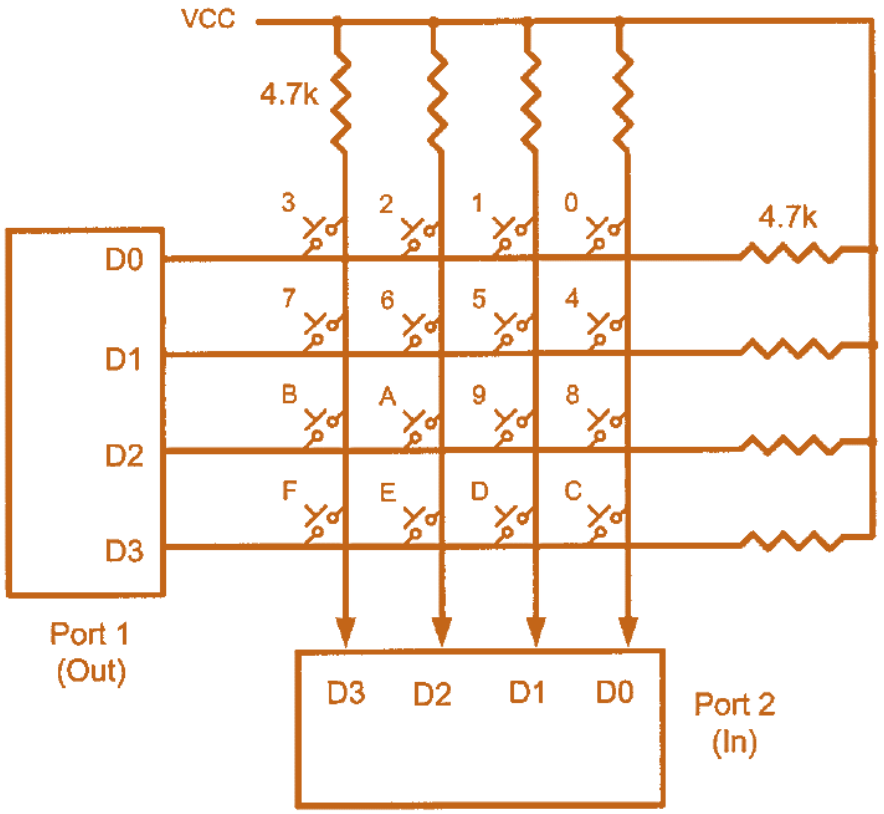
```
#include <avr/io.h>             //standard AVR header
#include <util/delay.h>         //delay header
#define  LCD_DPRT   PORTA       //LCD DATA PORT
#define  LCD_DDDR   DDRA        //LCD DATA DDR
#define  LCD_DPIN   PINA        //LCD DATA PIN
#define  LCD_CPRT   PORTB       //LCD COMMANDS PORT
#define  LCD_CDDR   DDRB        //LCD COMMANDS DDR
#define  LCD_CPIN   PINB        //LCD COMMANDS PIN
#define  LCD_RS   0             //LCD RS
#define  LCD_RW   1             //LCD RW
#define  LCD_EN   2             //LCD EN

void delay_us(int d)
{
   _delay_us(d);
}

void lcdCommand( unsigned char cmnd )
{
   LCD_DPRT = cmnd & 0xF0;      //send high nibble to D4-D7
   LCD_CPRT &= ~ (1<<LCD_RS);   //RS = 0 for command
   LCD_CPRT &= ~ (1<<LCD_RW);   //RW = 0 for write
   LCD_CPRT |= (1<<LCD_EN);     //EN = 1 for H-to-L pulse
   delay_us(1);                 //make EN pulse wider
   LCD_CPRT &= ~ (1<<LCD_EN);   //EN = 0 for H-to-L pulse
   delay_us(100);               //wait
   LCD_DPRT = cmnd<<4;          //send low nibble to D4-D7
   LCD_CPRT |= (1<<LCD_EN);     //EN = 1 for H-to-L pulse
   delay_us(1);                 //make EN pulse wider
   LCD_CPRT &= ~ (1<<LCD_EN);   //EN = 0 for H-to-L pulse
   delay_us(100);               //wait
}

void lcdData( unsigned char data )
{
   LCD_DPRT = data & 0xF0;      //send high nibble to D4-D7
   LCD_CPRT |= (1<<LCD_RS);     //RS = 1 for data
   LCD_CPRT &= ~ (1<<LCD_RW);   //RW = 0 for write
   LCD_CPRT |= (1<<LCD_EN);     //EN = 1 for H-to-L pulse
   delay_us(1);                 //make EN pulse wider
   LCD_CPRT &= ~ (1<<LCD_EN);   //EN = 0 for H-to-L pulse
   LCD_DPRT = data<<4;          //send low nibble to D4-D7
   LCD_CPRT |= (1<<LCD_EN);     //EN = 1 for H-to-L pulse
```

Example cnt.

```
  delay_us(1);                  //make EN pulse wider
  LCD_CPRT &= ~ (1<<LCD_EN);    //EN = 0 for H-to-L pulse
  delay_us(100);                //wait
}

void lcd_init()
{
  LCD_DDDR = 0xFF;
  LCD_CDDR = 0xFF;
  LCD_CPRT &=~(1<<LCD_EN);      //LCD_EN = 0
  lcdCommand(0x33);             //send $33 for init.
  lcdCommand(0x32);             //send $32 for init.
  lcdCommand(0x28);             //init. LCD 2 line,5x7 matrix
  lcdCommand(0x0e);             //display on, cursor on
  lcdCommand(0x01);             //clear LCD
  delay_us(2000);
  lcdCommand(0x06);             //shift cursor right
}

void lcd_gotoxy(unsigned char x, unsigned char y)
{
  unsigned char firstCharAdr[ ]={ 0x80,0xC0,0x94,0xD4};
  lcdCommand(firstCharAdr[ y-1]  + x - 1);
  delay_us(100);
}

void lcd_print(char * str )
{
  unsigned char i = 0;
  while(str[ i] !=0)
  {
    lcdData(str[ i] );
    i++ ;
  }
}

int main(void)
{
  lcd_init();
  lcd_gotoxy(1,1);
  lcd_print("The world is but");
  lcd_gotoxy(1,2);
  lcd_print("one country");
  while(1);                     //stay here forever
  return 0;
}
```

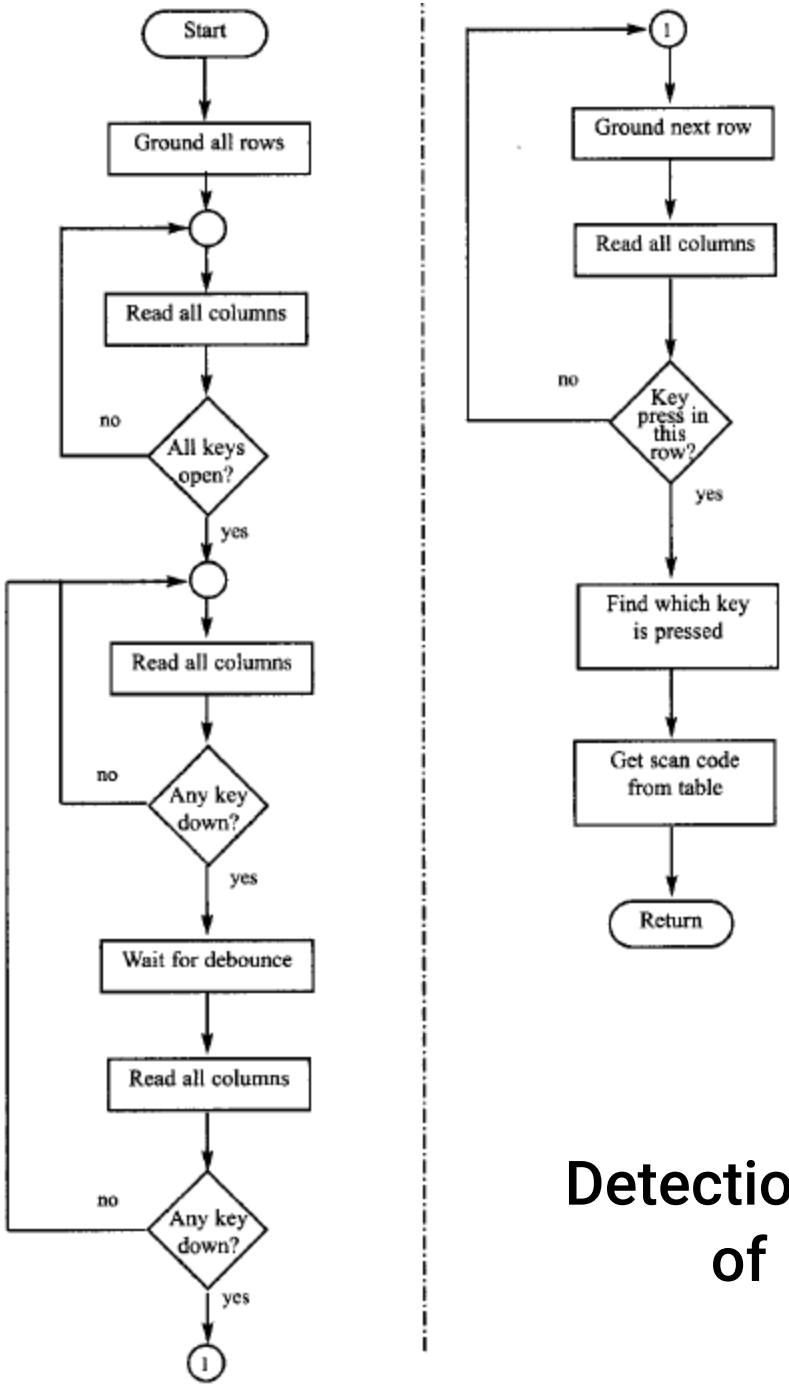
Interfacing the Keyboard to the AVR

At the lowest level, keyboards are organized in a matrix of rows and columns. The CPU accesses both rows and columns through ports; therefore, with two 8-bit ports, an 8 × 8 matrix of keys can be connected to a microcontroller. When a key is pressed, a row and a column make a contact; otherwise, there is no connection between rows and columns.



Interfacing the Keyboard to the AVR

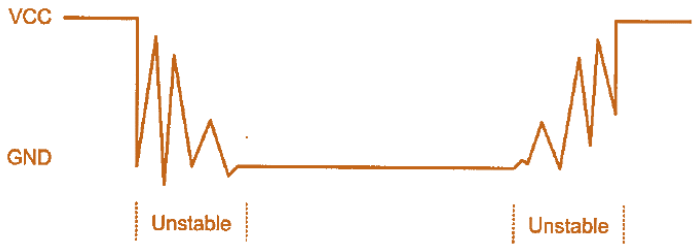
To detect a pressed key, the microcontroller grounds all rows by providing 0 to the output latch, and then it reads the columns. If the data read from the columns is D3–D0 = 1111, no key has been pressed and the process continues until a key press is detected. However, if one of the column bits has a zero, this means that a key press has occurred. For example, if D3–D0 = 1101, this means that a key in the D1 column has been pressed. After a key press is detected, the microcontroller will go through the process of identifying the key. Starting with the top row, the microcontroller grounds it by providing a low to row D0 only; then it reads the columns. If the data read is all 1s, no key in that row is activated and the process is moved to the next row. It grounds the next row, reads the columns, and checks for any zero. This process continues until the row is identified. After identification of the row in which the key has been pressed, the next task is to find out which column the pressed key belongs to. This should be easy since the microcontroller knows at any time which row and column are being accessed.



Detection and Identification of Key Activation

Detection and Identification of Key Activation

1. To make sure that the preceding key has been released, 0s are output to all rows at once, and the columns are read and checked repeatedly until all the columns are high. When all columns are found to be high, the program waits for a short amount of time before it goes to the next stage of waiting for a key to be pressed.
2. To see if any key is pressed, the columns are scanned over and over in an infinite loop until one of them has a 0 on it. Remember that the output latches connected to rows still have their initial zeros (provided in stage 1), making them grounded. After the key press detection, the microcontroller waits 20 ms for the bounce and then scans the columns again. This serves two functions: (a) it ensures that the first key press detection was not an erroneous one due to a spike noise, and (b) the 20-ms delay prevents the same key press from being interpreted as a multiple key press. Look at the Figure . If after the 20-ms delay the key is still pressed, it goes to the next stage to detect which row it belongs to; otherwise, it goes back into the loop to detect a real key press.



Detection and Identification of Key Activation

3. To detect which row the key press belongs to, the microcontroller grounds one row at a time, reading the columns each time. If it finds that all columns are high, this means that the key press cannot belong to that row; therefore, it grounds the next row and continues until it finds the row the key press belongs to. Upon finding the row that the key press belongs to, it sets up the starting address for the look-up table holding the scan codes (or the ASCII value) for that row and goes to the next stage to identify the key.

4. To identify the key press, the microcontroller rotates the column bits, one bit at a time, into the carry flag and checks to see if it is low. Upon finding the zero, it pulls out the ASCII code for that key from the look-up table; otherwise, it increments the pointer to point to the next element of the look-up table.

Example

the AVR Assembly language program for detection and identification of key activation. In this program, it is assumed that PC0–PC3 are connected to the rows and PC4–PC7 are connected to the columns.

```
.INCLUDE "M32DEF.INC"
.EQU  KEY_PORT = PORTC
.EQU  KEY_PIN = PINC
.EQU  KEY_DDR = DDRC
      LDI   R20,HIGH(RAMEND)
      OUT   SPH,R20
      LDI   R20,LOW(RAMEND)       ;init. stack pointer
      OUT   SPL,R20
      LDI   R21,0xFF
      OUT   DDRD,R21
      LDI   R20,0xF0
      OUT   KEY_DDR,R20
GROUND_ALL_ROWS:
      LDI   R20,0x0F
      OUT   KEY_PORT,R20
WAIT_FOR_RELEASE:
      NOP
      IN    R21,KEY_PIN           ;read key pins
      ANDI  R21,0x0F              ;mask unused bits
      CPI   R21,0x0F              ;(equal if no key)
      BRNE  WAIT_FOR_RELEASE      ;do again until keys released
WAIT_FOR_KEY:
      NOP                         ;wait for sync. circuit
      IN    R21,KEY_PIN           ;read key pins
      ANDI  R21,0x0F              ;mask unused bits
      CPI   R21,0x0F              ;(equal if no key)
      BREQ  WAIT_FOR_KEY          ;do again until a key pressed
      CALL  WAIT15MS              ;wait 15 ms
      IN    R21,KEY_PIN           ;read key pins
      ANDI  R21,0x0F              ;mask unused bits
      CPI   R21,0x0F              ;(equal if no key)
      BREQ  WAIT_FOR_KEY          ;do again until a key pressed
      LDI   R21,0b01111111        ;ground row 0
      OUT   KEY_PORT,R21
      NOP                         ;wait for sync. circuit
      IN    R21,KEY_PIN           ;read all columns
      ANDI  R21,0x0F              ;mask unused bits
      CPI   R21,0x0F              ;(equal if no key)
      BRNE  COL1                  ;row 0, find the colum
      LDI   R21,0b10111111        ;ground row 1
      OUT   KEY_PORT,R21
      NOP                         ;wait for sync. circuit
      IN    R21,KEY_PIN           ;read all columns
      ANDI  R21,0x0F              ;mask unused bits
      CPI   R21,0x0F              ;(equal if no key)
      BRNE  COL2                  ;row 1, find the colum
```

Example cnt.

```
      LDI   R21,0b11011111       ;ground row 2
      OUT   KEY_PORT,R21
      NOP                        ;wait for sync. circuit
      IN    R21,KEY_PIN          ;read all columns
      ANDI  R21,0x0F             ;mask unused bits
      CPI   R21,0x0F             ;(equal if no key)
      BRNE  COL3                 ;row 2, find the colum
      LDI   R21,0b11101111       ;ground row 3
      OUT   KEY_PORT,R21
      NOP                        ;wait for sync. circuit
      IN    R21,KEY_PIN          ;read all columns
      ANDI  R21,0x0F             ;mask unused bits
      CPI   R21,0x0F             ;(equal if no key)
      BRNE  COL4                 ;row 3, find the colum
COL1:
      LDI   R30,LOW(KCODE0<<1)
      LDI   R31,HIGH(KCODE0<<1)
      RJMP  FIND
COL2:
      LDI   R30,LOW(KCODE1<<1)
      LDI   R31,HIGH(KCODE1<<1)
      RJMP  FIND
COL3:
      LDI   R30,LOW(KCODE2<<1)
      LDI   R31,HIGH(KCODE2<<1)
      RJMP  FIND
COL4:
      LDI   R30,LOW(KCODE3<<1)
      LDI   R31,HIGH(KCODE3<<1)
      RJMP  FIND
FIND:
      LSR   R21
      BRCC  MATCH                ;if Carry is low go to match
      LPM   R20,Z+               ;INC Z
      RJMP  FIND
MATCH:
      LPM   R20,Z
      OUT   PORTD,R20
      RJMP  GROUND_ALL_ROWS
WAIT15MS:                        ;place a code to wait 15 ms
                                 ;here
      RET

.ORG 0x300

KCODE0:    .DB '0','1','2','3'   ;ROW 0
KCODE1:    .DB '4','5','6','3'   ;ROW 1
KCODE2:    .DB '8','9','A','B'   ;ROW 2
KCODE3:    .DB 'C','D','E','F'   ;ROW 3
```

Example

Write a C program to read the keypad and send the result to Port D.
PC0–PC3 connected to columns
PC4–PC7 connected to rows

```
#include <avr/io.h>                    //standard AVR header
#include <util/delay.h>                //delay header


#define     KEY_PRT   PORTC            //keyboard PORT
#define     KEY_DDR   DDRC             //keyboard DDR
#define     KEY_PIN   PINC             //keyboard PIN

void delay_ms(unsigned int d)
{
   _delay_ms(d);
}

unsigned char keypad[4][4] ={ '0','1','2','3',
                              '4','5','6','7',
                              '8','9','A','B',
                              'C','D','E','F'};

int main(void)
{
  unsigned char colloc, rowloc;

  //keyboard routine. This sends the ASCII
  //code for pressed key to  port c
  DDRD = 0xFF;
  KEY_DDR = 0xF0;                      //
  KEY_PRT = 0xFF;
  while(1)                             //repeat forever
  {
    do
    {
      KEY_PRT &= 0x0F;                 //ground all rows at once
      colloc = (KEY_PIN & 0x0F);       //read the columns
    } while(colloc != 0x0F);           //check until all keys released

    do
    {
      do
      {
        delay_ms(20);                  //call delay
        colloc =(KEY_PIN&0x0F);        //see if any key is pressed
      } while(colloc == 0x0F);         //keep checking for key press

      delay_ms(20);                    //call delay for debounce
      colloc = (KEY_PIN & 0x0F);       //read columns
    }while(colloc == 0x0F);            //wait for key press

    while(1)
    {
      KEY_PRT = 0xEF;                  //ground row 0
      colloc = (KEY_PIN & 0x0F);       //read the columns
```

Example cnt.

```
        if(colloc != 0x0F)              //column detected
        {
          rowloc = 0;                   //save row location
          break;                        //exit while loop
        }

        KEY_PRT = 0xDF;                 //ground row 1
        colloc = (KEY_PIN & 0x0F);      //read the columns

        if(colloc != 0x0F)              //column detected
        {
          rowloc = 1;                   //save row location
          break;                        //exit while loop
        }

        KEY_PRT =  0xBF;                //ground row 2
         colloc = (KEY_PIN & 0x0F);     //read the columns
        if(colloc != 0x0F)              //column detected
        {
          rowloc = 2;                   //save row location
          break;                        //exit while loop
        }

        KEY_PRT = 0x7F;                 //ground row 3
        colloc = (KEY_PIN & 0x0F);      //read the columns
        rowloc = 3;                     //save row location
        break;                          //exit while loop
      }

      //check column and send result to Port D
      if(colloc == 0x0E)
        PORTD = (keypad[rowloc][0]);
      else if(colloc == 0x0D)
        PORTD = (keypad[rowloc][1]);
      else if(colloc == 0x0B)
        PORTD = (keypad[rowloc][2]);
      else
        PORTD = (keypad[rowloc][3]);
    }
    return 0 ;

}
```

پایان

موفق و پیروز باشید