CHAPTER 12

LCD AND KEYBOARD INTERFACING

OBJECTIVES

Upon completion of this chapter, you will be able to:

- >> **List reasons that LCDs are gaining widespread use, replacing LEDs**
- >> **Describe the functions of the pins of a typical LCD**
- >> **List instruction command codes for programming an LCD**
- >> **Interface an LCD to the AVR**
- >> **Program an LCD in Assembly and C**
- >> **Explain the basic operation of a keyboard**
- >> **Describe the key press and detection mechanisms**
- >> **Interface a 4 × 4 keypad to the AVR using C and Assembly**

This chapter explores some real-world applications of the AVR. We explain how to interface the AVR to devices such as an LCD and a keyboard. In Section 12.1, we show LCD interfacing with the AVR. In Section 12.2, keyboard interfacing with the AVR is shown. We use C and Assembly for both sections.

SECTION 12.1: LCD INTERFACING

This section describes the operation modes of LCDs and then describes how to program and interface an LCD to an AVR using Assembly and C.

LCD operation

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.

LCD pin descriptions

The LCD discussed in this section has 14 pins. The function of each pin is given in Table 12-1. Figure 12-1 shows the pin positions for various LCDs.

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.

E, enable

The enable pin is used by the LCD to latch information presented to its data pins.

Table 12-1: 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

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. Table 12-2 lists the instruction command codes.

In this section you will see how to interface an LCD to the AVR in two different ways. We can use 8-bit data or 4-bit data options. The 8-bit data interfacing is easier to program but uses 4 more pins.

Table 12-2: 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)

Note: This table is extracted from Table 12-4.

Dot matrix character LCDs are available in different packages. Figure 12-1 shows the position of each pin in different packages.

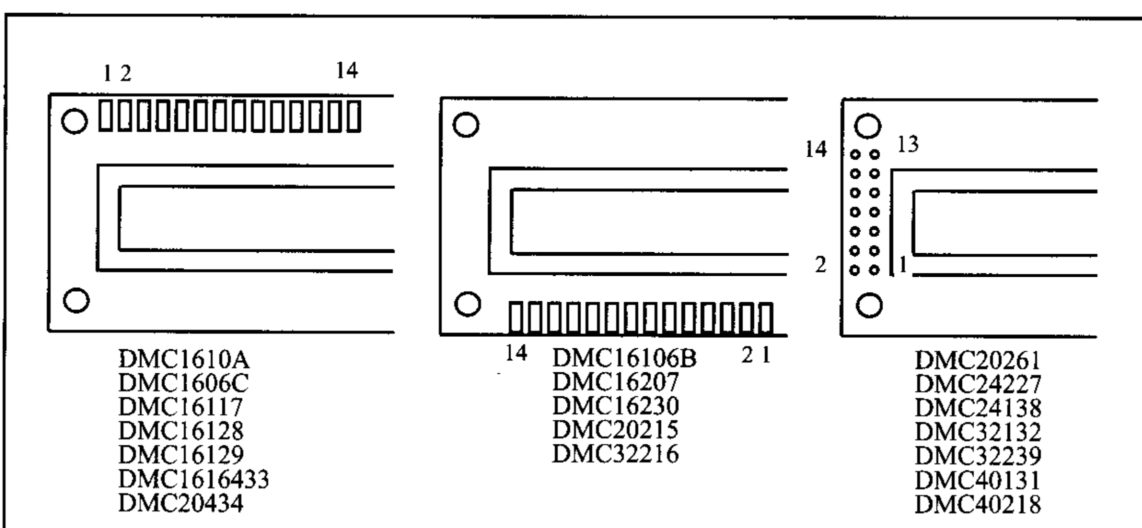


Figure 12-1. Pin Positions for Various LCDs from Optrex

Sending commands and data to LCDs

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 from Table 12-2 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 to the LCD

To send any of the commands from Table 12-2 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. Table 12-3 shows the details of commands and their execution times.

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.

Program 12-1 shows how to write "Hi" on the LCD using 8-bit data. The AVR connection to the LCD for 8-bit data is shown in Figure 12-2.

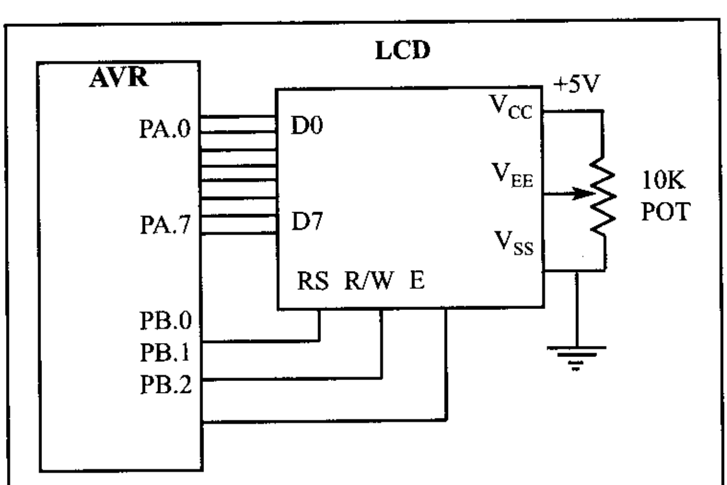


Figure 12-2. LCD Connections for 8-bit Data

```
.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
;-------------------------------------------------------
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
```

Program 12-1: Communicating with LCD *(continued on next page)*

```
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
```

Program 12-1: Communicating with LCD *(continued from previous page)*

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

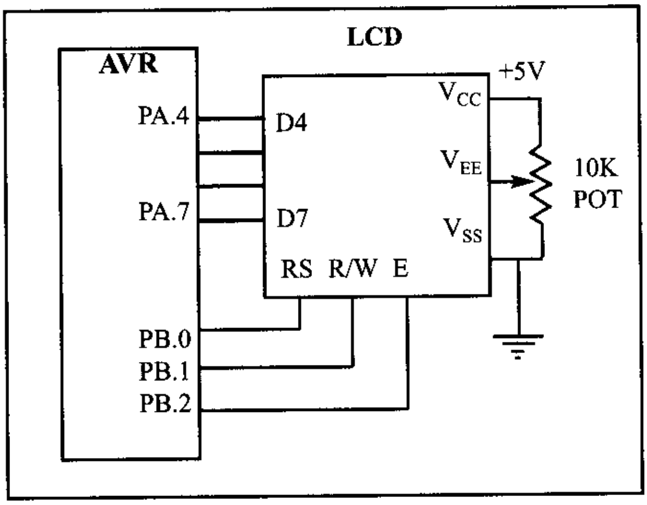


Figure 12-3. LCD Connections Using 4-bit Data

The above code showed how to send commands to the LCD with 8 bits for the data pin. In most cases it is preferred to use 4-bit data to save pins. The LCD may be forced into the 4-bit mode as shown in Program 12-2. Notice that its initialization differs from that of the 8-bit mode and that data is sent out on the high nibble of Port A, high nibble first.

In 4-bit mode, we initialize the LCD with the series 33, 32, and 28 in hex. This represents nibbles 3, 3, 3, and 2, which tells the LCD to go into 4-bit mode. The value \$28 initializes the display for 5 × 7 matrix and 4-bit operation as required by the LCD datasheet. The write routines (CMNDWRT and DATAWRT) send the high nibble first, then swap the low nibble with the high nibble before it is sent to data pins D4–D7. The delay function of the program is the same as in Program 12-1.

```
.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
         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,5x7 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
```

Program 12-2: Communicating with LCD Using 4-bit Mode *(continued on next page)*

```
;-----------------------------------------------------------
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.
;-----------------------------------------------------------
```

Program 12-2: Communicating with LCD Using 4-bit Mode *(continued from previous page)*

Sending code or data to the LCD using a single port

The above code showed how to send commands to the LCD with 4-bit data but we used two different ports for data and commands. In most cases it is preferred to use a single port. Program 12-3 shows Program 12-2 modified to use a single port for LCD interfacing.

Figure 12-4 shows the hardware connection.

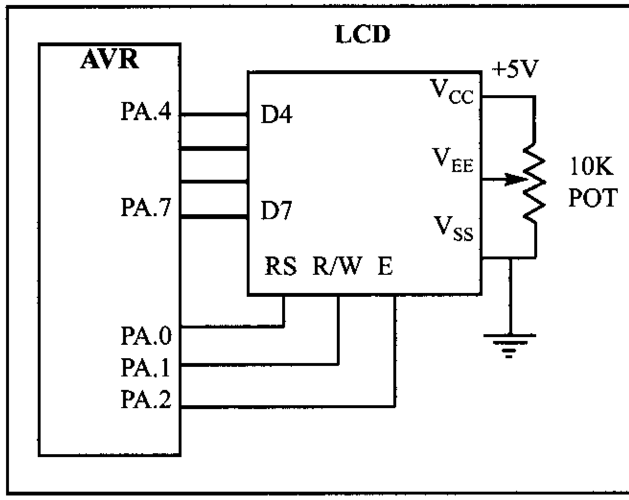


Figure 12-4. LCD Connections Using a Single Port

```
.INCLUDE "M32DEF.INC"

.EQU     LCD_PRT = PORTA        ;LCD DATA PORT
.EQU     LCD_DDR = DDRA         ;LCD DATA DDR
.EQU     LCD_PIN = PINA         ;LCD DATA 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_DDR, R21    ;LCD data port is output
         OUT    LCD_DDR, 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
```

Program 12-3: Communicating with LCD Using a Single Port *(continued on next page)*

```
        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
;-----------------------------------------------------------
CMNDWRT:
        MOV     R27,R16
        ANDI    R27,0xF0
        IN      R26,LCD_PRT
        ANDI    R26,0x0F
        OR      R26,R27
        OUT     LCD_PRT,R26             ;LCD data port = R16
        CBI     LCD_PRT,LCD_RS          ;RS = 0 for command
        CBI     LCD_PRT,LCD_RW          ;RW = 0 for write
        SBI     LCD_PRT,LCD_EN          ;EN = 1 for high pulse
        CALL    SDELAY                  ;make a wide EN pulse
        CBI     LCD_PRT,LCD_EN          ;EN=0 for H-to-L pulse

        CALL    DELAY_100us             ;make a wide EN pulse

        MOV     R27,R16
        SWAP    R27
        ANDI    R27,0xF0
        IN      R26,LCD_PRT
        ANDI    R26,0x0F
        OR      R26,R27
        OUT     LCD_PRT,R26             ;LCD data port = R16
        SBI     LCD_PRT,LCD_EN          ;EN = 1 for high pulse
        CALL    SDELAY                  ;make a wide EN pulse
        CBI     LCD_PRT,LCD_EN          ;EN=0 for H-to-L pulse

        CALL    DELAY_100us             ;wait 100 us
        RET
;-----------------------------------------------------------
DATAWRT:
        MOV     R27,R16
        ANDI    R27,0xF0
        IN      R26,LCD_PRT
        ANDI    R26,0x0F
```

Program 12-3: Communicating with LCD Using a Single Port *(continued from previous page)*

```
        OR      R26,R27
        OUT     LCD_PRT,R26         ;LCD data port = R16
        SBI     LCD_PRT,LCD_RS      ;RS = 1 for data
        CBI     LCD_PRT,LCD_RW      ;RW = 0 for write
        SBI     LCD_PRT,LCD_EN      ;EN = 1 for high pulse
        CALL    SDELAY              ;make a wide EN pulse
        CBI     LCD_PRT,LCD_EN      ;EN=0 for H-to-L pulse

        MOV     R27,R16
        SWAP    R27
        ANDI    R27,0xF0
        IN      R26,LCD_PRT
        ANDI    R26,0x0F
        OR      R26,R27
        OUT     LCD_PRT,R26         ;LCD data port = R16
        SBI     LCD_PRT,LCD_EN      ;EN = 1 for high pulse
        CALL    SDELAY              ;make a wide EN pulse
        CBI     LCD_PRT,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
```

Program 12-3: Communicating with LCD Using a Single Port *(continued from previous page)*

Sending information to LCD using the LPM instruction

Program 12-4 shows how to use the LPM instruction to send a long string of characters to an LCD. Program 12-4 shows only the main part of the code. The other functions do not change. If you want to use a single port you have to change the port definition in the beginning of the code according to Program 12-2.

```
.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   LDELAY         ;wait for init.
         LDI    R16,0x38       ;init LCD 2 lines, 5x7 matrix
         CALL   CMNDWRT        ;call command function
         CALL   LDELAY         ;init. hold
         LDI    R16,0x0E       ;display on, cursor on
         CALL   CMNDWRT        ;call command function
         LDI    R16,0x01       ;clear LCD
         CALL   CMNDWRT        ;call command function
         LDI    R16,0x06       ;shift cursor right
         CALL   CMNDWRT        ;call command function
         LDI    R16,0x84       ;cursor at line 1 pos. 4
         CALL   CMNDWRT        ;call command function
         LDI    R31,HIGH(MSG<<1)
         LDI    R30,LOW(MSG<<1);Z points to MSG
LOOP:    LPM    R16,Z+
         CPI    R16,0          ;compare R16 with 0
         BREQ   HERE           ;if R16 equals 0 exit
         CALL   DATAWRT        ;call data write function
         RJMP   LOOP           ;jump to loop
HERE:    JMP HERE              ;stay here
MSG:     .DB "Hello World!",0
```

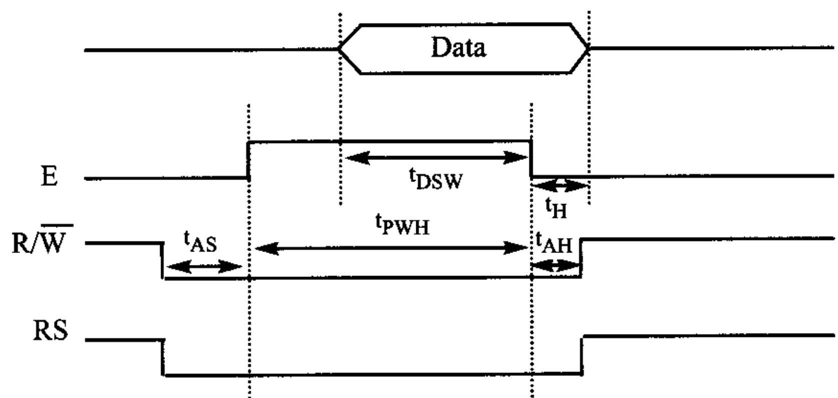
Program 12-4: Communicating with LCD Using the LPM Instruction

LCD data sheet

Here we deepen your understanding of LCDs by concentrating on two important concepts. First we will show you the timing diagram of the LCD; then we will discuss how to put data at any location.

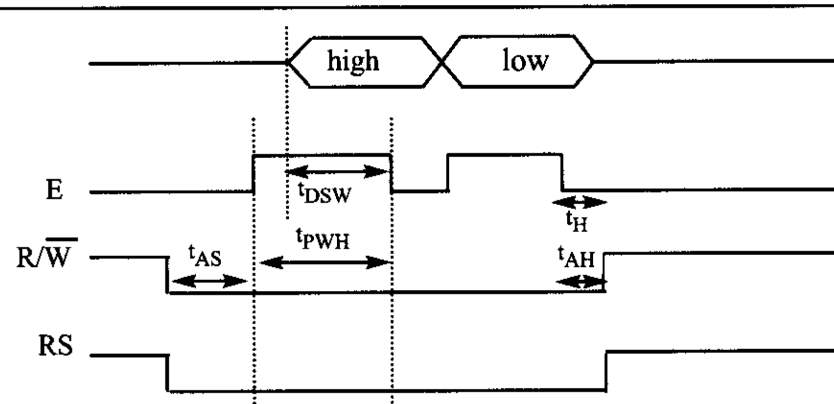
LCD timing diagrams

In Figures 12-5 and 12-6 you can study and contrast the Write timing for the 8-bit and 4-bit modes. Notice that in the 4-bit operating mode, the high nibble is transmitted. Also notice that each nibble is followed by a high-to-low pulse to enable the internal latch of the LCD.



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)

Figure 12-5. LCD Timing for Write (H-to-L for E line)



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)

Figure 12-6. LCD Timing for 4-bit Write

LCD detailed commands

Table 12-3 provides a detailed list of LCD commands and instructions.

Table 12-3: List of LCD Instructions

Instruction	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Description	Execution Time (Max)
Clear Display	0	0	0	0	0	0	0	0	0	1	Clears entire display and sets DD RAM address 0 in address counter.	1.64 ms
Return Home	0	0	0	0	0	0	0	0	1	-	Sets DD RAM address 0 as address counter. Also returns display being shifted to original position. DD RAM contents remain unchanged.	1.64 ms
Entry Mode Set	0	0	0	0	0	0	0	1	1/D	S	Sets cursor move direction and specifies shift of display. These operations are performed during data write and read.	40 μs
Display On/ Off Control	0	0	0	0	0	0	1	D	C	B	Sets On/Off of entire display (D), cursor On/Off (C), and blink of cursor position character (B).	40 μs
Cursor or Display Shift	0	0	0	0	0	1	S/C	R/L	-	-	Moves cursor and shifts display without changing DD RAM contents.	40 μs
Function Set	0	0	0	0	1	DL	N	F	-	-	Sets interface data length (DL), number of display lines (L), and character font (F).	40 μs
Set CG RAM Address	0	0	0	1	AGC						Sets CG RAM address. CG RAM data is sent and received after this setting.	40 μs
Set DD RAM Address	0	0	1	ADD							Sets DD RAM address. DD RAM data is sent and received after this setting.	40 μs
Read Busy Flag & Address	0	1	BF	AC							Reads Busy flag (BF) indicating internal operation is being performed and reads address counter contents.	40 μs
Write Data CG or DD RAM	1	0	Write Data								Writes data into DD or CG RAM.	40 μs
Read Data CG or DD RAM	1	1	Read Data								Reads data from DD or CG RAM.	40 μs

Notes:

1. *Execution times are maximum times when fcp or fosc is 250 kHz.*
2. *Execution time changes when frequency changes. Ex: When fcp or fosc is 270 kHz:* 40 μs × 250 / 270 = 37 μs.
3. Abbreviations:

DD RAM	Display data RAM		
CG RAM	Character generator RAM		
ACC	CG RAM address		
ADD	DD RAM address, corresponds to cursor address		
AC	Address counter used for both DD and CG RAM addresses		
1/D = 1	Increment	1/D = 0	Decrement
S = 1	Accompanies display shift		
S/C = 1	Display shift;	S/C = 0	Cursor move
R/L = 1	Shift to the right;	R/L = 0	Shift to the left
DL = 1	8 bits, DL = 0: 4 bits		
N = 1	1 line, N = 0: 1 line		
F = 1	5 × 10 dots, F = 0: 5 × 7 dots		
BF = 1	Internal operation;	BF = 0	Can accept instruction

(Table 12-2 is extracted from this table.) As you see in the eighth row of Table 12-3, you can set the DD RAM address. It lets you put data at any location. The following shows how to set DD RAM address locations.

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	1	A	A	A	A	A	A	A

Where AAAAAAA = 0000000 to 0100111 for line 1 and AAAAAAA = 1000000 to 1100111 for line 2.

The upper address range can go as high as 0100111 for the 40-character-wide LCD, while for the 20-character-wide LCD it goes up to 010011 (19 decimal = 10011 binary). Notice that the upper range 0100111 (binary) = 39 decimal, which corresponds to locations 0 to 39 for the LCDs of 40 × 2 size.

From the above discussion we can get the addresses of cursor positions for various sizes of LCDs. See Table 12-4 for the cursor addresses for common types of LCDs. Notice that all the addresses are in hex. See Example 12-1.

LCD Type	Line	Address Range				
16 × 2 LCD	Line 1:	80	81	82	83	through 8F
	Line 2:	C0	C1	C2	C3	through CF
20 × 1 LCD	Line 1:	80	81	82	83	through 93
20 × 2 LCD	Line 1:	80	81	82	83	through 93
	Line 2:	C0	C1	C2	C3	through D3
20 × 4 LCD	Line 1:	80	81	82	83	through 93
	Line 2:	C0	C1	C2	C3	through D3
	Line 3:	94	95	96	97	through A7
	Line 4:	D4	D5	D6	D7	through E7
40 × 2 LCD	Line 1:	80	81	82	83	through A7
	Line 2:	C0	C1	C2	C3	through E7

Note: All data is in hex.

Table 12-4: Cursor Addresses for Some LCDs

Example 12-1

What is the cursor address for the following positions in a 20 × 4 LCD?
(a) Line 1, Column 1
(b) Line 2, Column 1
(c) Line 3, Column 2
(d) Line 4, Column 3

Solution:

(a) 80
(b) C0
(c) 95
(d) D6

LCD programming in C

Programs 12-5, 12-6, and 12-7 show how to interface an LCD to the AVR using C programming. The codes are modular to improve code clarity.

Program 12-5 shows how to use 8-bit data to interface an LCD to the AVR in C language.

```
// YOU HAVE TO SET THE CPU FREQUENCY IN AVR STUDIO
// BECAUSE YOU ARE USING PREDEFINED DELAY FUNCTION

#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
```

Program 12-5: Communicating with LCD Using 8-bit Data in C *(continued on next page)*

```
  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 × 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};//Table 12-5
 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;
}
```

Program 12-5: Communicating with LCD Using 8-bit Data in C

Program 12-6 shows how to use 4-bit data to interface an LCD to the AVR in C language.

```
#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
```

Program 12-6: Communicating with LCD Using 4-bit Data in C *(continued on next page)*

```
    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,5×7 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;
}
```

Program 12-6: Communicating with LCD Using 4-bit Data in C

Program 12-7 shows how to use 4-bit data to interface an LCD to the AVR in C language. It uses only a single port. Also there are some useful functions to print a string (array of chars) or to move the cursor to a specific location.

```
#include <avr/io.h>                     //standard AVR header
#include <util/delay.h>                 //delay header
#define  LCD_PRT  PORTA                 //LCD DATA PORT
#define  LCD_DDR  DDRA                  //LCD DATA DDR
#define  LCD_PIN  PINA                  //LCD DATA 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 delay_ms(int d)
{
  _delay_ms(d);
}

void lcdCommand( unsigned char cmnd ){
  LCD_PRT = (LCD_PRT & 0x0F) | (cmnd & 0xF0);
  LCD_PRT &= ~ (1<<LCD_RS);             //RS = 0 for command
  LCD_PRT &= ~ (1<<LCD_RW);             //RW = 0 for write
  LCD_PRT |= (1<<LCD_EN);               //EN = 1 for H-to-L
  delay_us(1);                          //wait to make EN wider
  LCD_PRT &= ~ (1<<LCD_EN);             //EN = 0 for H-to-L

  delay_us(20);                         //wait

  LCD_PRT = (LCD_PRT & 0x0F) | (cmnd << 4);
  LCD_PRT |= (1<<LCD_EN);               //EN = 1 for H-to-L
  delay_us(1);                          //wait to make EN wider
  LCD_PRT &= ~ (1<<LCD_EN);             //EN = 0 for H-to-L
}

void lcdData( unsigned char data ){
  LCD_PRT = (LCD_PRT & 0x0F) | (data & 0xF0);
  LCD_PRT |= (1<<LCD_RS);               //RS = 1 for data
  LCD_PRT &= ~ (1<<LCD_RW);             //RW = 0 for write
  LCD_PRT |= (1<<LCD_EN);               //EN = 1 for H-to-L
```

Program 12-7: Communicating with LCD Using 4-bit Data in C *(continued on next page)*

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

  LCD_PRT = (LCD_PRT & 0x0F) | (data << 4);
  LCD_PRT |= (1<<LCD_EN);           //EN = 1 for H-to-L
  delay_us(1);                      //wait to make EN wider
  LCD_PRT &= ~ (1<<LCD_EN);         //EN = 0 for H-to-L
}

void lcd_init(){
  LCD_DDR = 0xFF;                   //LCD port is output

  LCD_PRT &=~(1<<LCD_EN);           //LCD_EN = 0
  delay_us(2000);                   //wait for stable power
  lcdCommand(0x33);                 //$33 for 4-bit mode
  delay_us(100);                    //wait
  lcdCommand(0x32);                 //$32 for 4-bit mode
  delay_us(100);                    //wait
  lcdCommand(0x28);                 //$28 for 4-bit mode
  delay_us(100);                    //wait
  lcdCommand(0x0e);                 //display on, cursor on
  delay_us(100);                    //wait
  lcdCommand(0x01);                 //clear LCD
  delay_us(2000);                   //wait
  lcdCommand(0x06);                 //shift cursor right
  delay_us(100);
}

void lcd_gotoxy(unsigned char x, unsigned char y)
{ //Table 12-5
  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++;
  }
}
```

Program 12-7: Communicating with LCD Using 4-bit Data in C

```
int main(void)
{
   lcd_init();
   while(1)
   {                          //stay here forever
        lcd_gotoxy(1,1);
        lcd_print("The world is but");
        lcd_gotoxy(1,2);
        lcd_print("one country      ");
        delay_ms(1000);
        lcd_gotoxy(1,1);
        lcd_print("and mankind its ");
        lcd_gotoxy(1,2);
        lcd_print("citizens         ");
        delay_ms(1000);
   }
   return 0;
}
```

Program 12-7: Communicating with LCD Using 4-bit Data in C *(cont. from previous page)*

You can purchase the LCD expansion board of the MDE AVR trainer from the following websites:

www.digilentinc.com
www.MicroDigitalEd.com

The LCDs can be purchased from the following websites:

www.digikey.com
www.jameco.com
www.elexp.com

Review Questions

1. The RS pin is an _______ (input, output) pin for the LCD.
2. The E pin is an ________ (input, output) pin for the LCD.
3. The E pin requires an _______ (H-to-L, L-to-H) pulse to latch in information at the data pins of the LCD.
4. For the LCD to recognize information at the data pins as data, RS must be set to _____ (high, low).
5. What is the 0x06 command ?
6. Which of the following commands takes more than 100 microseconds to run?
   (a) Shift cursor left
   (b) Shift cursor right
   (c) Set address location of DDRAM
   (d) Clear screen
7. Which of the following initialization commands initializes an LCD for 5 × 7 matrix characters in 8-bit operating mode?
   (a) 0x38, 0x0E, 0x0, 0x06
   (b) 0x0E, 0x0, 0x06
   (c) 0x33, 0x32, 0x28, 0x0E, 0x01, 0x06
   (d) 0x01, 0x06
8. Which of the following initialization commands initializes an LCD for 5 × 7 matrix characters in 4-bit operating mode?
   (a) 0x38, 0x0E, 0x0, 0x06
   (b) 0x0E, 0x0, 0x06
   (c) 0x33, 0x32, 0x28, 0x0E, 0x01, 0x06
   (d) 0x01, 0x06
9. Which of the following is the address of the second column of the second row in a 2 × 20 LCD?
   (a) 0x80
   (b) 0x81
   (c) 0xC0
   (d) 0xC1
10. Which of the following is the address of the second column of the second row in a 4 × 20 LCD?
    (a) 0x80
    (b) 0x81
    (c) 0xC0
    (d) 0xC1
11. Which of the following is the address of the first column of the second row in a 4 × 20 LCD?
    (a) 0x80
    (b) 0x81
    (c) 0xC0
    (d) 0xC1

SECTION 12.2: KEYBOARD INTERFACING

Keyboards and LCDs are the most widely used input/output devices in microcontrollers such as the AVR and a basic understanding of them is essential. In the previous section, we discussed how to interface an LCD with an AVR using some examples. In this section, we first discuss keyboard fundamentals, along with key press and key detection mechanisms. Then we show how a keyboard is interfaced to an AVR.

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. In x86 PC keyboards, a single microcontroller takes care of hardware and software interfacing of the keyboard. In such systems, it is the function of programs stored in the Flash of the microcontroller to scan the keys continuously, identify which one has been activated, and present it to the motherboard. In this section we look at the mechanism by which the AVR scans and identifies the key.

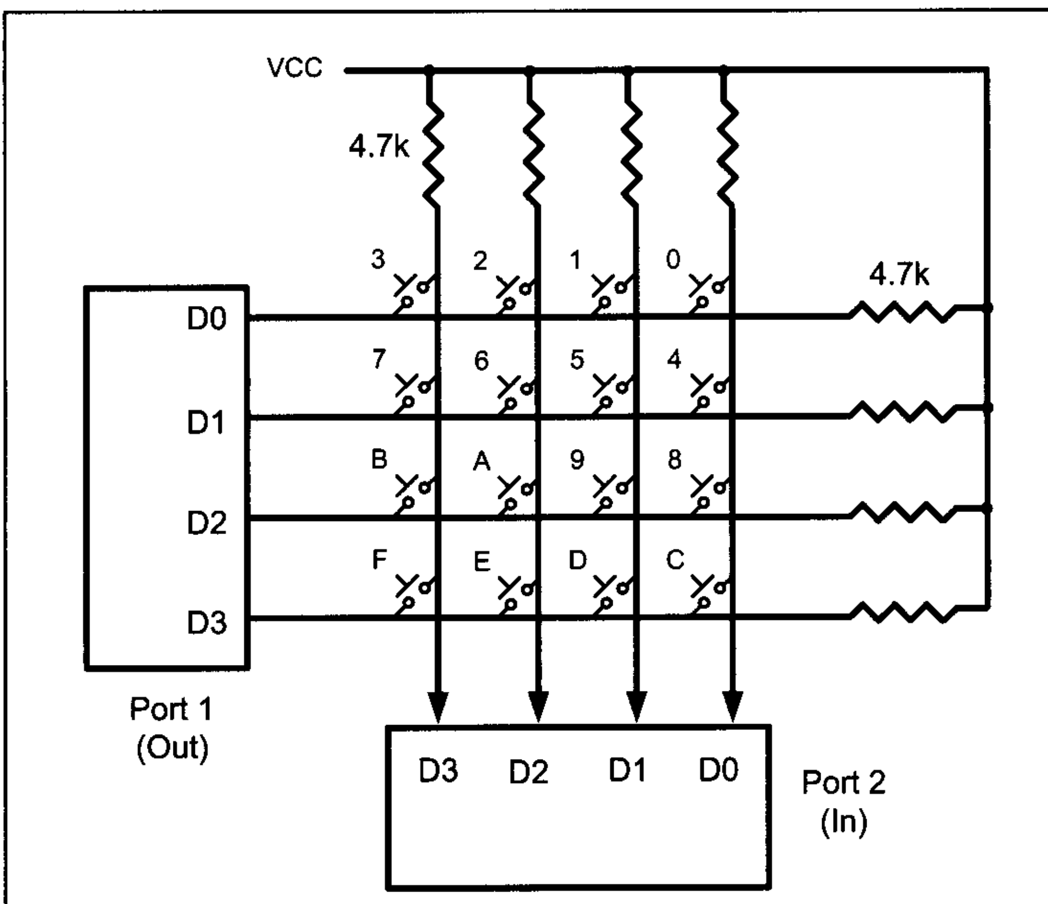


Figure 12-7. Matrix Keyboard Connection to Ports

Scanning and identifying the key

Figure 12-7 shows a 4 × 4 matrix connected to two ports. The rows are connected to an output port and the columns are connected to an input port. If no key has been pressed, reading the input port will yield 1s for all columns since they are all connected to high (VCC). If all the rows are grounded and a key is pressed, one of the columns will have 0 since the key pressed provides the path to ground. It is the function of the microcontroller to scan the keyboard continuously to detect and identify the key pressed. How this is done is explained next.

Grounding rows and reading the columns

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. Look at Example 12-2.

Example 12-2

From Figure 12-7 identify the row and column of the pressed key for each of the following.
(a) D3–D0 = 1110 for the row, D3–D0 = 1011 for the column
(b) D3–D0 = 1101 for the row, D3–D0 = 0111 for the column

Solution:

From Figure 12-7 the row and column can be used to identify the key.
(a) The row belongs to D0 and the column belongs to D2; therefore, key number 2 was pressed.
(b) The row belongs to D1 and the column belongs to D3; therefore, key number 7 was pressed.

Program 12-8 is 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.

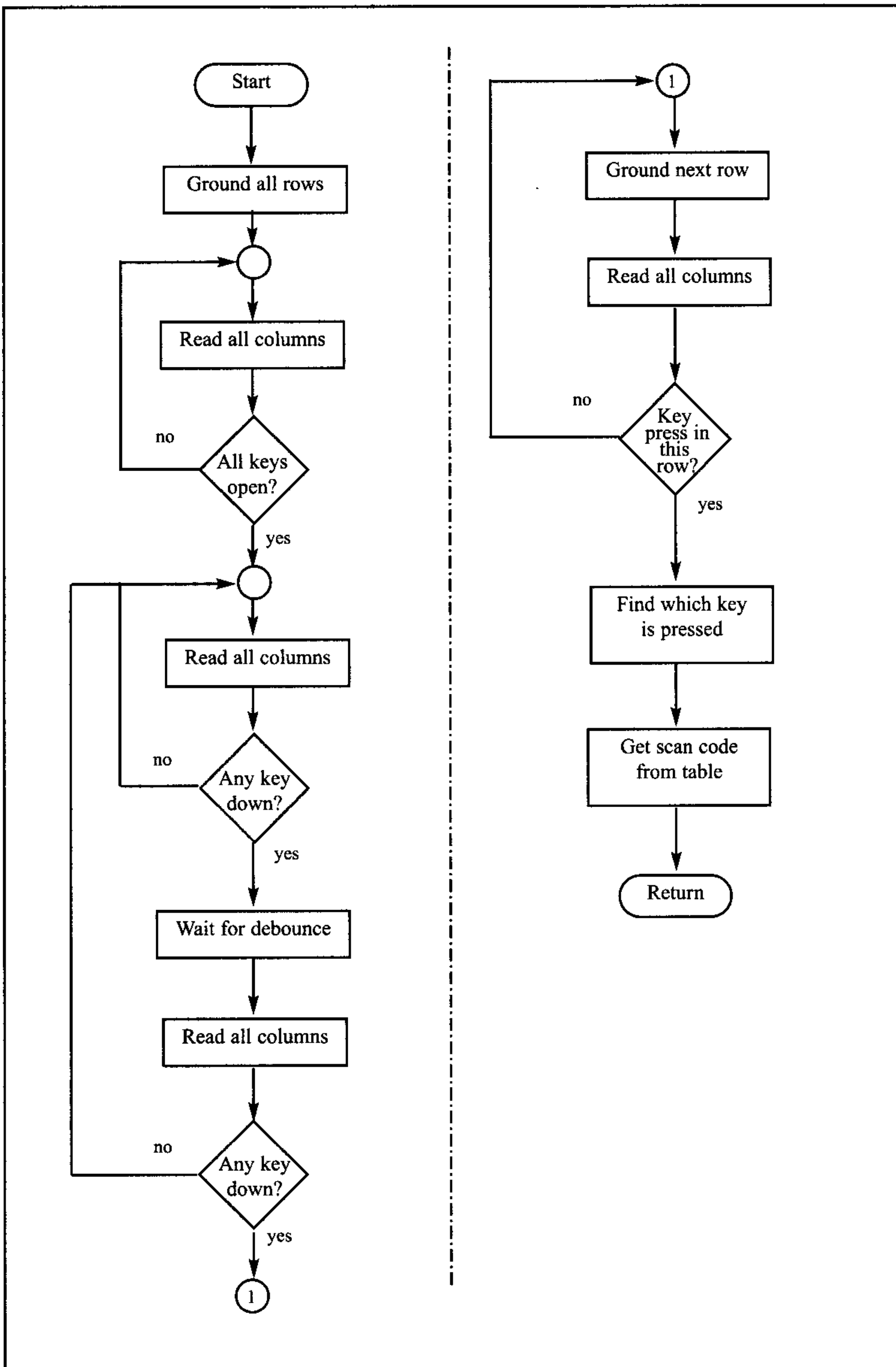


Figure 12-8. Flowchart for Program 12-8

Program 12-8 goes through the following four major stages (Figure 12-8 flowcharts this process):

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 Figure 12-9. 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.
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.

While the key press detection is standard for all keyboards, the process for determining which key is pressed varies. The look-up table method shown in Program 12-8 can be modified to work with any matrix up to 8 × 8. Example 12-3 shows keypad programming in C.

There are IC chips such as National Semiconductor's MM74C923 that incorporate keyboard scanning and decoding all in one chip. Such chips use combinations of counters and logic gates (no microcontroller) to implement the underlying concepts presented in Program 12-8.

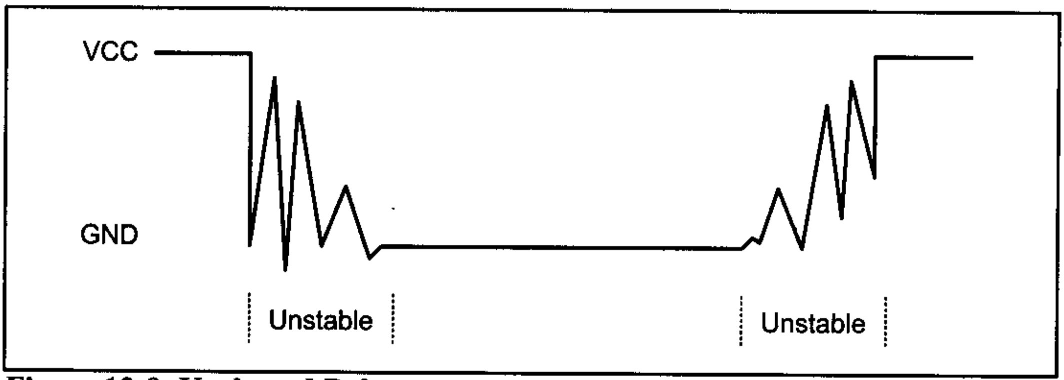


Figure 12-9. Keyboard Debounce

```
;Keyboard Program. This program sends the ASCII code
;for pressed key to Port D
;PC0-PC3 connected to rows PC4-PC7 connected to 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
```

Program 12-8: Keyboard Interfacing Program *(continued on next page)*

```
        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
```

Program 12-8. Keyboard Interfacing Program *(continued from previous page)*

Example 12-3

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

Solution:

```
#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 12-3 *(continued from previous page)*

```
        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 ;

}
```

Review Questions

1. True or false. To see if any key is pressed, all rows are grounded.
2. If D3–D0 = 0111 is the data read from the columns, which column does the pressed key belong to?
3. True or false. Key press detection and key identification require two different processes.
4. In Figure 12-7, if the rows are D3–D0 = 1110 and the columns are D3–D0 = 1110, which key is pressed?
5. True or false. To identify the pressed key, one row at a time is grounded.

SUMMARY

This chapter showed how to interface real-world devices such as LCDs and keypads to the AVR. First, we described the operation modes of LCDs, and then described how to program the LCD by sending data or commands to it via its interface to the AVR.

Keyboards are one of the most widely used input devices for AVR projects. This chapter also described the operation of keyboards, including key press and detection mechanisms. Then the AVR was shown interfacing with a keyboard. AVR programs were written to return the ASCII code for the pressed key.

PROBLEMS

SECTION 12.1: LCD INTERFACING

1. The LCD discussed in this section has ____ pins.
2. Describe the function of pins E, R/W, and RS in the LCD.
3. What is the difference between the V_{CC} and V_{EE} pins on the LCD?
4. "Clear LCD" is a __________ (command code, data item) and its value is ___ hex.
5. What is the hex value of the command code for "display on, cursor on"?
6. Give the state of RS, E, and R/W when sending a command code to the LCD.
7. Give the state of RS, E, and R/W when sending data character 'Z' to the LCD.
8. Which of the following is needed on the E pin in order for a command code (or data) to be latched in by the LCD?
   (a) H-to-L pulse (b) L-to-H pulse
9. True or false. For the above to work, the value of the command code (data) must already be at the D0–D7 pins.
10. There are two methods of sending commands and data to the LCD: (1) 4-bit mode or (2) 8-bit mode. Explain the difference and the advantages and disadvantages of each method.
11. For a 16 × 2 LCD, the location of the last character of line 1 is 8FH (its command code). Show how this value was calculated.
12. For a 16 × 2 LCD, the location of the first character of line 2 is C0H (its command code). Show how this value was calculated.
13. For a 20 × 2 LCD, the location of the last character of line 2 is 93H (its command code). Show how this value was calculated.
14. For a 20 × 2 LCD, the location of the third character of line 2 is C2H (its command code). Show how this value was calculated.
15. For a 40 × 2 LCD, the location of the last character of line 1 is A7H (its command code). Show how this value was calculated.
16. For a 40 × 2 LCD, the location of the last character of line 2 is E7H (its command code). Show how this value was calculated.
17. Show the value (in hex) for the command code for the 10th location, line 1 on a 20 × 2 LCD. Show how you got your value.
18. Show the value (in hex) for the command code for the 20th location, line 2 on

a 40×2 LCD. Show how you got your value.

SECTION 12.2: KEYBOARD INTERFACING

19. In reading the columns of a keyboard matrix, if no key is pressed we should get all ______ (1s, 0s).
20. In the 4×4 keyboard interfacing, to detect the key press, which of the following is grounded?
    (a) all rows (b) one row at time (c) both (a) and (b)
21. In the 4×4 keyboard interfacing, to identify the key pressed, which of the following is grounded?
    (a) all rows (b) one row at time (c) both (a) and (b)
22. For the 4×4 keyboard interfacing (Figure 12-7), indicate the column and row for each of the following.
    (a) D3–D0 = 0111 (b) D3–D0 = 1110
23. Indicate the steps to detect the key press.
24. Indicate the steps to identify the key pressed.
25. Indicate an advantage and a disadvantage of using an IC chip for keyboard scanning and decoding instead of using a microcontroller.
26. What is the best compromise for the answer to Problem 25?

ANSWERS TO REVIEW QUESTIONS

SECTION 12.1: LCD INTERFACING

1. Input
2. Input
3. H-to-L
4. High
5. Shift cursor to right
6. d
7. a
8. c
9. d
10. d
11. c

SECTION 12.2: KEYBOARD INTERFACING

1. True
2. Column 3
3. True
4. 0
5. True