Chapter 6. LCD Display and IR Remote Control Applications

This chapter extends the (software enabled) serial communication of Chapter 5 into the applications of data display and IR remote controller which have many additional applications for projects and other designs.

1. LCD Displaying

Alphanumeric LCD display is very popular for many applications because we can quickly and easily display a result of calculation or measurement, or data for debugging purpose. Of course, as we discussed before, a computer monitor is an excellent tool for the same purpose, but when we build an embedded computing system, much smaller LCD is always useful. There also are graphic LCDs are available.

A LCD is different from a LCD module. A LCD is just a medium to display characters or graphics, it itself also cannot display. A LCD module contains, in addition to the display medium, an interface controller/driver for the LCD. A LCD controller/driver displays alphanumerics and symbols. The most popular LCD controller/driver is the Hitachi 44780 based LCD controller chip. A single HD44780 can display up to one 8-character line or two 8-character lines. It can be configured to drive a dot-matrix liquid crystal display under the control of a 4- or 8-bit microprocessor.

<u>LCD Controller/Driver HD44780</u>
Internally HD44780 has a 80x8-bit display data (DD) RAM for maximum 80 characters, and 9,920-bit character generator(CG) ROM for a total of 240 character fonts (208 character fonts with 5x8 dot size and 32 character fonts with 5x10 dot size), and a 64x8-bit character generator RAM for 8 character fonts (5x8 dot) and 4 character fonts (5x10 dot). It also covers Wide range of instruction functions, "HD44780 Standard Control and Command Code," such as display clear, cursor home, display on/off, cursor on/off, display character blink, cursor shift, and display shift. It contains a reset circuit that initializes the controller/driver after power on.

Display data RAM (DDRAM) stores display data represented in 8-bit character codes. Its extended capacity is 80x8 bits, or 80 characters. The area in display data RAM (DDRAM) that is not used for display can be used as general data RAM. The following table shows the relationships between DDRAM addresses and positions on the LCD.

Position	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
First line	00h	01h	02h	03h	04h	05h	06h	07h	08h	09h	0Ah	0Bh	0Ch	0Dh	0Eh	0Fh	10h	11h	12h	13h
Second line	40h	41h	42h	43h	44h	45h	46h	47h	48h	49h	4Ah	4Bh	4Ch	4Dh	4Eh	4Fh	50h	51h	52h	53h
Third line	14h	15h	16h	17h	18h	19h	1Ah	1Bh	1Ch	1Dh	1Eh	1Fh	20h	21h	22h	23h	24h	25h	26h	27h
Fourth Line	54h	55h	56h	57h	58h	59h	5Ah	5Bh	5Ch	5Dh	5Eh	5Fh	60h	61h	62h	63h	64h	65h	66h	67h

In addition to the CGRAM and DDRAM, HD44780 has two 8-bit registers: an instruction register (IR) and a data register (DR). The IR stores instruction codes, such as display clear and cursor shift, and address information for display data RAM (DDRAM) and character generator RAM (CGRAM). The IR can only be written from microprocessor. The DR temporarily stores data to be written into DDRAM or CGRAM and temporarily stores data to be read from DDRAM or CGRAM.

Data written into the DR from the microprocessor is automatically written into DDRAM or CGRAM by an internal operation. The DR is also used for data storage when reading data from DDRAM or CGRAM. When address information is written into the IR, data is read and then stored into the DR from DDRAM or CGRAM by an internal operation. Data transfer to the microprocessor is then completed when the microprocessor reads the DR. After the read, data in DDRAM or CGRAM at the next address is sent to the DR for the next read from the processor. By the register selector (RS) signal, these two registers can be selected. In 16F877 perspective, by controlling the RS line for IR or DR, and sending a DDRAM location for display position and a data for a character to display that position, we can display a character on a desired position.

In addition to the IR and DR, there is Address Counter (AC). The AC assigns addresses to both DDRAM and CGRAM. When an address of an instruction is written into the IR, the address information is sent from the IR to the AC. Selection of either DDRAM or CGRAM is also determined concurrently by the instruction. After writing into (reading from) DDRAM or CGRAM, the AC is automatically incremented by 1 (decremented by 1). The AC contents are then output to DB0 to DB6 when RS = 0 and RW=0.

There are two interfacing method to a microprocessor. The HD44780U can send data in either two 4-bit operations or one 8-bit operation. For 4-bit interface, only four bus lines (DB4 to DB7) are used for transfer: Bus lines DB0 to DB3 are disabled. The data transfer between the HD44780U and the microprocessor is completed only after the 4-bit data has been transferred twice. As for the order of data transfer, the high nibble (DB4 to DB7) are transferred before the low nibble (DB0 toDB3). The busy flag must be checked (one instruction) after the 4-bit data has been transferred twice. Two more 4-bit operations then transfer the busy flag and address counter data. For 8-bit interface, all eight bus lines (DB0 to DB7) are used.

This section will explore the control of a regular LCD module and a serial LCD module. One caution we all have to use is that not all LCD modules are the same: some with different characteristics and pin arrangement, etc. Therefore, before you try to connect a LCD to 16F877, you have to read the data sheet of the module you received or bought. However, once you make yourself familiar with the one presented in this section, on any module of LCD, you can easily change the physical connection and code to adapt to the changing characteristics.

LCD example

A regular LCD module we discuss here is one manufactured by Truly which can display 4 rows, 20 characters per row, with character dot matrix size of 5x8. The exact model number is MTC-C204. So we use 20x4 LCD display with HD44780 controller or equivalent.

The pin arrangement for the LCD module is listed below.

Pin NO.	Symbol	Level	Description
1	V_{SS}	0V	Ground
2	V_{DD}	5.0V	Supply voltage for logic
3	V_O	---	Input voltage for LCD
4	RS	H/L	H : Data, L : Instruction code
5	R/W	H/L	H : Read mode, L : Write mode

6	E	H, H → L	Chip enable signal
7	DB0	H/L	Data bit 0
8	DB1	H/L	Data bit 1
9	DB2	H/L	Data bit 2
10	DB3	H/L	Data bit 3
11	DB4	H/L	Data bit 4
12	DB5	H/L	Data bit 5
13	DB6	H/L	Data bit 6
14	DB7	H/L	Data bit 7
15	BLA	---	For LCD Backlight (Anode)
16	BLK	---	For LCD Backlight (Cathode)

A host microprocessor "talks" to the LCD cotnroller/Driver via the data bus and 3 control lines: Register Select (RS), Read/Write (RW) and Enable (E). This places minimal demands upon the microprocessor. Only when the host microprocessor writes to or reads from the LCD, is intercommunication required.

The Control and Display Command codes for communicating to HD44780 LCD controller/driver are shown below. These codes are good for any LCD module with HD44780 or equivalent processor as the controller/driver of the module.

Control/ Command	**Code** R R B B B B B B B B S W 7 6 5 4 3 2 1 0	**Description**	**Execution Time with f=250Khz**
Clear Display	0 0 0 0 0 0 0 0 0 1	Clears all display and returns the cursor to the home position (Address 0)	1.64ms
Return Home	0 0 0 0 0 0 0 0 1 X	Returns the cursor to the home position (Address 0). Also returns the display being shifted to the original position.	1.64ms
Entry Mode Set	0 0 0 0 0 0 0 1 M S	Set cursor move direction (M=1 for increase, M=0 for decrease) and shift of display (S=1 for shifted and S=0 for not-shifted)	40μs
Display On/Off	0 0 0 0 0 0 1 D C B	Sets On/Off of a Display (D=1 for On and D=0 for Off), Cursor (C=1 for On and C=0 for Off), and Blinking (B=1 for Blink On and B=0 for Blink Off)	40μs
Shift	0 0 0 0 0 1 S R X X	Moves the cursor (S=1 for Shift and S=0 for Cursor Move) and shifts display (R=1 for Right and R=0 for Left Shift).	40μs
Set Function	0 0 0 0 1 L N F X X	Sets interface data length (L=1 for 8-bit and L=0 for 4-bit), number at display lines (N=1 for 2 line display and N=0 for 1 line display), and once character font (F=1 for 5x10 and F=0 for 5x7 dots)	40μs
Set CG RAM Address	0 0 0 1 <--Acg --->	Set the CG (Character Generator) RAM address (i.e., cursor address). CG RAM data is sent and received after this set.	40μs
Set DD RAM Address	0 0 1 <----Add---->	Set the DD (Display Data) RAM address. DD RAM data is sent and received after this set.	40μs
Read Busy Flag & Address	0 1 B <--Acount--->	Reads Bust Flag indicating internal operation is being performed (B=1 for Busy and B=0 for Ready) and read address counter (Acount) contents used for both DD and CG RAMs.	1ms
Write Data	1 0 <----DATA----->	Write DATA to DD or CG RAM	40μs
Read Data	1 1 <----DATA----->	Read DATA from DD or CG RAM	40μs

The LCD Waveform diagram below shows how a data is written to the LCD module. As seen, even though the data is written to the internal data register, it still cannot be displayed on the LCD unless a High-to-Low transition input of E(Enable) signal is provided to the module.

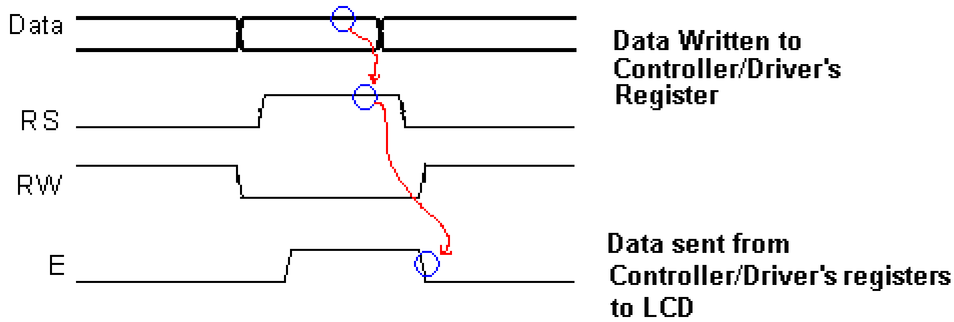


Fig. 24 LCD Waveform diagram

This High-to-Low transition input of E(Enable) signal is also needed when an instruction is written to the instruction register of the LCD controller/Driver. When your interface bit is 4, then we have to send the data twice, higher nibble then lower nibble. For each nibble write, we have to have the transitional E signal.

Initialization of LCD module

As mentioned above, HD44780 has an automatic reset circuit when power is on. The following instructions are executed during the initialization. The busy flag (B) is kept in the busy state until the initialization ends (B = 1). The busy state lasts for 10 ms after VCC rises to 4.5 V.

1. Display clear
2. Function set: 8-bit interface, 1-line display, 5x8 dot character font
3. Display on/off control: Display off, Cursor off, Blinking off
4. Entry mode set: Increment by 1, No shift (DDRAM is selected)

If the power supply condition does not reset properly, we have to initialize by instruction. Following is a usual LCD module initialization sequence by instruction.

1. Give power to the LCD module.
2. Wait for 15ms or more so that LCD is warm and ready to respond.
3. Set function for interface data length (i.e., 8 or 4 bits), number of display lines, and character dot matrix size.
4. Wait for 4.5 ms.
5. Check for Busy Flag.
6. Display Off.
7. Display Clear.
8. Set Entry mode.

Operation Example (8-bit interface with 8-digit 2 line display with internal reset)

We have many I/O ports in 16F877, so for this example, we try 4-bit interface and this requires total 11 pins. Assume that PORTB is assigned to the 8 data lines <DB7:DB0>. Since we usually do not read from we connect the RW line to the ground for always-reading status. The busy flag checking, thus cannot be done by this configuration. However, giving enough time delay after writing an instruction or data is does the job. Then, we need two more lines for RS and E signal. Assume that they occupy two pins of PORTD.

LCDEven though the Truly LCD has four lines for display, internally, it is considered as 2 line display. It's done all by the DDRAM address selection as shown in the DDRAM address map. In other words, in 2 line mode, the first line can go from 00h to 40h, but since the LCD module can display only 20 characters, the first line starts from 00 but ends at 13h, then from 14h to 27h will be displayed at the third line. Similarly, the DDRAM addresses of 40h – 53h are displayed at the second line and those of 54-67h are displayed at the fourth line.

Display Position for 2 Lines

DDRAM address (hexadecimal)	1	2	3	4	5		20	21	22		39	40
	00	01	02	03	04	.....		14	15	....	26	27
	40	41	42	43	44	.....		54	55	....	66	67
	1	2					20	21	22		39	40

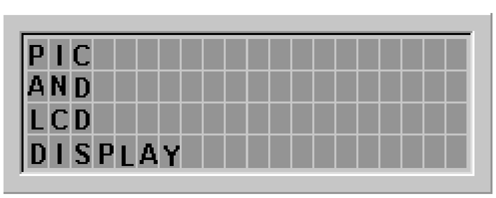
Display Position with 20x2 Format

DDRAM address (hexadecimal)		1	2	3	4	5		20	21	22		39	40
	Line 1	00	01	02	03	04	.....		14	15	....	26	27
	Line 2	40	41	42	43	44	.....		54	55	....	66	67
		1	2					20	21	22		39	40

Display Position with 20x4 Format

DDRAM address (hexadecimal)		1	2				20	1	2		19	20	
	Line 1	00	01	02	03	04	.....	14	15	....	26	27	Line 3
	Line 2	40	41	42	43	44	....	54	55	....	66	67	Line 4
		1	2				20	1	2		19	20	

The following example shows 8-bit interface (<DB7:DB0>) for 20x4 format with 5x8 dot matrix size. Note that RW is tied to ground for ever-writing mode. The steps and accompanying code will eventually display the four lines as shown below.



Step 1: Turn on Power to initialize the LCD. Give enough delay. There is no display.

A 16F877 instruction goes like this:

```
        call  delay10ms
        call  delay10ms   ;delay for 20ms
```

However, if you are not sure the power on reset actually work, you may have to follow the recommended initialization process. See the instructional initialization process.

Step 2: Function set for 8-bit, 2-line display, and 5x8 dot matrix.
RS=0
<DB7:DB0>= 0 0 1 1 1 0 X X

16F877 instruction for this is:

```
        movlw       0x38
        movwf       PORTB
        bcf         PORTD, RS
        bsf         PORTD, E
        bcf         PORTD, E          ;Transitional E signal
        call        delay10ms
```

The above instruction writing can be made into a subroutine.

```
;subroutine instw (instruction write)
;instruction to be written is stored in W before the call
instw movwf       PORTB
      bcf         PORTD, RS
      bsf         PORTD, E
      bcf         PORTD,E
      call        delay10ms
      return
```

Then, the above instruction can be rewritten to:

```
        movlw       0x38
        call        instw
```

Step 3. Display control: Display On, Cursor On, with no blinking are selected.
RS=0
<DB7:DB0>=0 0 0 0 1 1 1 0

Corresponding 16F877 code goes like:

```
        movlw       0x0E
        call        instw
```

Step 4: Entry mode set: Increment the DDRAM address by one and to shift the cursor to the right at the time of write to DDRAM. Display is not shofted
RS=0
<DB7:DB0>= 0 0 0 0 0 1 1 0

Corresponding 16F877 code goes like:

```
        movlw       0x0E
        call        instw
```

Step 5: Write data (i.e., 'P' of 50h in ASCII code) to DDRAM (The initial DDRAM address is set to 00h by the power on initialization.) So the line#1 position 1 is already selected by the reset. After this write, the cursor is incremented by 1 and shifted to the right.

RS=1
<DB7:DB0>= 0 1 0 1 0 0 0 0

Corresponding 16F877 code goes like:

```
movlw       0x50
movwf       PORTB
bsf         PORTD, RS
bsf         PORTD, E
bcf         PORTD, E           ;Transitional E signal
call        delay10ms
```

By changing the above code into a subroutine, we have the following code:

```
      movlw 0x50
      call  dataw

;subroutine dataw (data write)
dataw movwf       PORTB
      bsf         PORTD, RS
      bsf         PORTD, E
      bcf         PORTD, E           ;Transitional E signal
      call        delay10ms
      return
```

So we call `instrw` when RS=0 and `dataw` when RS=1.

Step 6: Write data (i.e., 'I' and 'C' next to 'P' in line #1) to DDRAM. Note that the DDRAM address in automatically incremented by one after each write, therefore, we do not write the DDRAM address (or position).

RS=1
<DB7:DB0>= 0 1 0 0 1 0 0 1 for 'I'
<DB7:DB0>= 0 1 0 0 0 0 1 1 for 'C'

Corresponding 16F877 code goes like:

```
movlw 0x50              ;'I'
call  dataw
movlw 0x43
call  dataw             ;'C'
```

Step 7. Set DDRAM address for the next 3 characters (A, N, and D) in line #2. The DDRAM address starts from 40h for the line #2.

RS=0
<DB7:DB0>= 1 1 0 0 0 0 0 0 for 1000000b

Corresponding 16F877 code goes like:

```
movlw 0xC0              ;B'11000000'
call  instw             ;RS=0
```

Step 8. Write the three characters, 'A', 'N', and 'D' to DDRAM. They are displayed at the line #2 from position 1.

RS=1
<DB7:DB0>= 0 1 0 0 0 0 0 1 for 'A'

<DB7:DB0>= 0 1 0 0 1 1 1 0 for 'N'
<DB7:DB0>= 0 1 0 0 0 1 0 0 for 'D'

Corresponding 16F877 code goes like:

```
movlw 0x41              ;'A'
call  dataw
movlw 0x4E
call  dataw             ;'N'
movlw 0x44
call  dataw             ;'D'
```

Step 9. Set DDRAM address for the next 3 characters (L, C, and D) in line #3. The DDRAM address starts from 14h for the line #3.

RS=0
<DB7:DB0>= 1 0 0 1 0 0 0 0 for 0010000b

Corresponding 16F877 code goes like:

```
movlw 0x94              ;B'10010100'
call  instw             ;RS=0
```

Step 10. Write the three characters, 'L', 'C', and 'D' to DDRAM. They are displayed at the line #3 from position 1.

RS=1
<DB7:DB0>= 0 1 0 0 1 1 0 0 for 'L'
<DB7:DB0>= 0 1 0 0 0 0 1 1 for 'C'
<DB7:DB0>= 0 1 0 0 0 1 0 0 for 'D'

Corresponding 16F877 code goes like:

```
movlw 0x4C              ;'L'
call  dataw
movlw 0x43
call  dataw             ;'C'
movlw 0x44
call  dataw             ;'D'
```

Step 11. Set DDRAM address for the next 7 characters (D, I, S, P, L, A, and Y) in line #4. The DDRAM address starts from 54h for the line #3.

RS=0
<DB7:DB0>= 1 1 0 1 0 1 0 0 for 11010100b

Corresponding 16F877 code goes like:

```
movlw 0xD4
call  instw             ;RS=0
```

Step 12. Write the seven characters, 'D', 'I', 'S', 'P', 'L', 'A', and 'Y' to DDRAM. They are displayed at the line #4 from position 1.

RS=1
<DB7:DB0>= 0 1 0 0 0 1 0 0 for 'D'
<DB7:DB0>= 0 1 0 0 1 0 0 1 for 'I'
<DB7:DB0>= 0 1 0 1 0 0 1 1 for 'S'
<DB7:DB0>= 0 1 0 1 0 0 0 0 for 'P'

<DB7:DB0>= 0 1 0 0 1 1 0 0 for 'L'
<DB7:DB0>= 0 1 0 0 0 0 0 1 for 'A'
<DB7:DB0>= 0 1 0 1 1 0 0 1 for 'Y'

Corresponding 16F877 code goes like:

```
movlw 0x44              ;'D'
call  dataw
movlw 0x49              ;'I'
call  dataw
movlw 0x53              ;'S'
call  dataw             ;
movlw 0x50              ;'P'
call  dataw
movlw 0x4C              ;'L'
call  dataw
movlw 0x41              ;'A'
call  dataw
movlw 0x59              ;'Y'
call  dataw
```

Step 13. Now let's move the cursor to the home position (position 1 of line #1) and set the DDRAM address to 0. This is done by the "return home" instruction.

RS=0
<DB7:DB0>= 0 0 0 0 0 0 1 0

Corresponding 16F877 code goes like:

```
movlw 0x02
call  instw             ;RS=0
```

Instructional initialization Process:

Step 1: When power on reset actually work, you have to follow the recommended initialization process and have the following codes at the very first line:

```
call  delay10ms
call  delay10ms
movlw       0x30
call        instw       ;see step 2 below for instw
```

Step 2: Function set for 8-bit, 2-line display, and 5x8 dot matrix. (Still part of initialization. And this step for setting is final and cannot be changed after this step.)

RS=0

<DB7:DB0>= 0 0 1 1 1 0 X X

16F877 instruction for this is:

```
movlw       0x38
call        instw
```

Step 3. Display off. (Still initialization process)

RS=0

<DB7:DB9>= 0 0 0 0 1 0 0 0

16F877 instruction for this step is:

```
movlw       0x08
call        instw
```

Step 4. Display Clear. (Still in the initialization process)

RS=0

<DB7:DB0>= 0 0 0 0 0 0 0 1

16F877 instruction for this step is:

```
movlw       0x01
call        instw
```

Step 5. Entry Mode Set (The last step of initialization) for increment and no shift

RS=0

<DB7:DB0>= 0 0 0 0 0 1 1 0

16F877 instruction for this step is:

```
movlw       0x06
call        instw
```

Hardware connection

Let's connect the 20x4 LCD module as shown below. Eight data bus lines are connected to PORTB, and E and RS are connected to PORTD<5> and PORTD<4>, respectively. RW is connected to PORTD<6>, but, as indicated above, since our main function is to write either command or data to LCD module, RW can be tied to the ground to make "write only" mode.

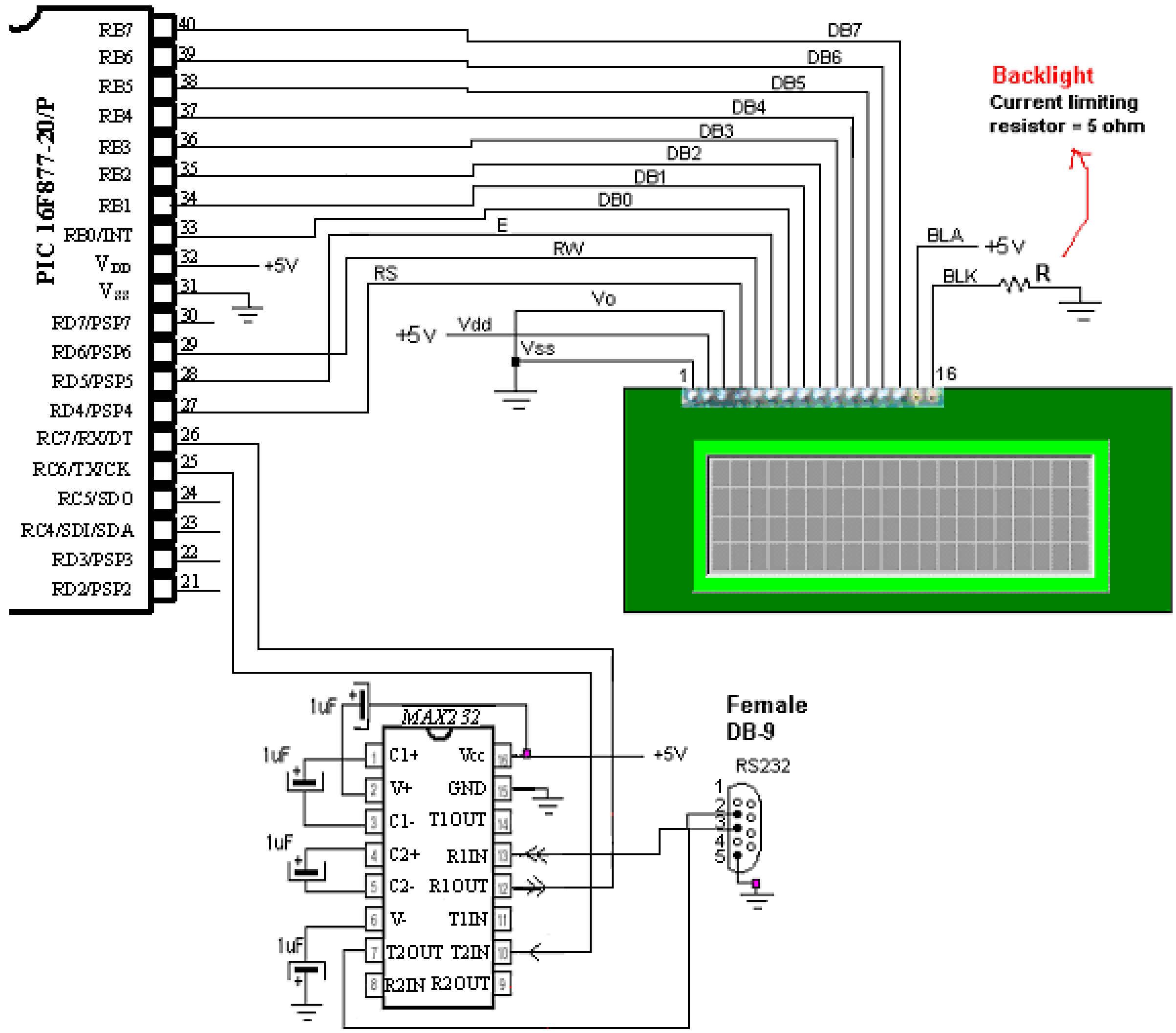


Fig 25. Hardware connection

Code example

Let's have an example code for the 8-bit interface mode control of a 20x4 LCD module. Follow the code carefully for instructions and comments.

```
;LCD-P.asm
;
;This program is to display an 20x4 LCD module
;by Truly (HD44780 compatible)
;
;8-bit interfacing
;
;Pin Connection from LCD to 16F877
;LCD (pin#) 16F877 (pin#)
```

```
;DB7 (14) -----RB7(40)
;DB6 (13) ----RB6(39)
;DB5 (12) ----RB5(38)
;DB4 (11) ----RB4(37)
;DB3 (10) ----RB3(36)
;DB2 (9)---- RB2(35)
;DB1 (8) ----RB1(34)
;DB0 (7) ----RB0(33)
;E (6) ------RD5(28)
;RW (5) -----RD6(29)
;RS (4) -----RD4(27)
;Vo (3) -----+5V
;Vdd (2) ----+5V
;Vss (1) -----GND
;
;Example code to display:
;      PIC
;      AND
;      LCD
;      DISPLAY
;

      list P = 16F877

STATUS       EQU   0x03
PORTB        EQU   0x06
TRISB        EQU   0x86
PORTD        EQU   0x08
TRISD        EQU   0x88
RS           EQU   0x04         ;RD4
E            EQU   0x05         ;RD5
RW           EQU   0x06         ;RW

;RAM arEA

      CBLOCK       0x20
            Kount120us   ;Delay count (number of instr cycles for delay)
            Kount100us
            Kount1ms
            Kount10ms
            Kount1s
            Kount10s
            Kount1m
      ENDC

;
;The Next 5 lines must be here
;because of bootloader arrangement
;Bootloader first execute the first 4 addresses
;then jump to the address what the execution directs
;=========================================================
        org       0x0000            ;line 1
        goto      START       ;line 2 ($0000)
        org       0x05

START
      BANKSEL     TRISD
; 1 for input, 0 for output
      movlw       0x00
      movwf       TRISD
      movwf       TRISB       ;RB<7:0> are all outputs

      banksel     PORTB
      clrf        PORTB
```

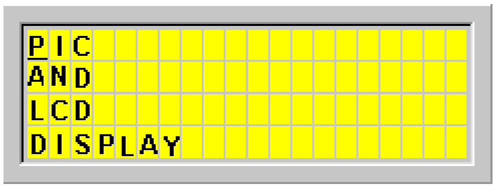
```
      clrf        PORTD       ;Here RW is pulled down to ground
;LCD routine starts
      call        delay10ms
      call        delay10ms
                  ;give LCD module to reset automatically
;Fundtion for 8-bit, 2-line display, and 5x8 dot matrix
      movlw       0x38
      call        instw
;Display On, CUrsor On, No blinking
      movlw       0x0E        ;0F would blink
      call        instw
;DDRAM address increment by one & cursor shift to right
      movlw       0x06
      call        instw
;DISPLAY CLEAR
      movlw       0x01
      call        instw

;Set DDRAM ADDRES
      movlw       0x80        ;00
      call  instw
;WRITE DATA in the 1st position of line 1
      movlw       0x50        ;P
      call        dataw

      movlw       0x49        ;I
      call        dataw

      movlw       0x43        ;C
      call        dataw

;Set DDRAM address for the 1st position of line 2 (40h)

      movlw       0xC0              ;B'11000000'
      call        instw             ;RS=0

;Write A, N, D

      movlw       0x41              ;A
      call        dataw
      movlw       0x4E
      call        dataw             ;N
      movlw       0x44
      call        dataw             ;D

;Set DDRAM address for the next 3 characters (L, C, and D) in line #3. (14h)
;The DDRAM address starts from 14h for the line #3.

      movlw       0x94              ;B'10010000'
      call        instw             ;RS=0

;Write the three characters, 'L', 'C', and 'D' to DDRAM.
;They are displayed at the line #3 from position 1.

      movlw       0x4C              ;L
      call        dataw
      movlw       0x43
      call        dataw             ;C
      movlw       0x44
      call        dataw             ;D

;Set DDRAM address for the next 7 characters (D, I, S, P, L, A, and Y) in line
#4.
;The DDRAM address starts from the line #4. (54h)
```

```
      movlw       0xD4
      call        instw             ;RS=0

;Write the seven characters, 'D', 'I', 'S', 'P', 'L', 'A', and 'Y' to DDRAM.
;They are displayed at the line #4 from position 1.

      movlw       0x44              ;D
      call        dataw
      movlw       0x49              ;I
      call        dataw
      movlw       0x53              ;S
      call        dataw             ;
      movlw       0x50              ;P
      call        dataw
      movlw       0x4C              ;L
      call        dataw
      movlw       0x41              ;A
      call        dataw
      movlw       0x59              ;Y
      call        dataw

;Now let's move the cursor to the home position (position 1 of line #1)
;and set the DDRAM address to 0.  This is done by the "return home"
instruction.

      movlw       0x02
      call        instw

IDLE  nop
      goto        IDLE

;====SUBROUTINES =====
;subroutine instw (instruction write)
;instruction to be written is stored in W before the call
instw movwf       PORTB
      call        delay1ms    ;delay may not be needed
      bcf         PORTD, RS
      call        delay1ms
      bsf         PORTD, E
      call        delay1ms
      bcf         PORTD,E
      call        delay10ms
      return

;subroutine dataw (data write)
dataw movwf       PORTB
      call        delay1ms    ;delay may not be needed
      bsf         PORTD, RS
      call        delay1ms
      bsf         PORTD, E
      call        delay1ms
      bcf         PORTD, E          ;Transitional E signal
      call        delay10ms
      return

;
;===========================================================
;DELAY SUBROUTINES

Delay120us
      banksel     Kount120us
      movlw       H'C5'       ;D'197'
      movwf       Kount120us
```

```
R120us
      decfsz      Kount120us
      goto        R120us
      return
;
Delay100us
      banksel     Kount100us
      movlw       H'A4'
      movwf       Kount100us
R100us
      decfsz      Kount100us
      goto        R100us
      return
;
;1ms delay
Delay1ms
      banksel     Kount1ms
      movlw       0x0A  ;10
      movwf       Kount1ms
R1ms  call        delay100us
      decfsz      Kount1ms
      goto        R1ms
      return
;
;10ms delay
; call 100 times of 100 us delay  (with some time discrepancy)
Delay10ms
      banksel     Kount10ms
      movlw       H'64' ;100
      movwf       Kount10ms
R10ms call        delay100us
      decfsz      Kount10ms
      goto        R10ms
      return
;
;

;1 sec delay
;call 100 times of 10ms delay
Delay1s
      banksel     Kount1s
      movlw       H'64'
      movwf       Kount1s
R1s   call        Delay10ms
      decfsz      Kount1s
      goto        R1s
      return
;
;
;10 s delay
;call 10 tiems of 1 s delay
Delay10s
      banksel     Kount10s
      movlw       H'0A'       ;10
      movwf       Kount10s
R10s  call        Delay1s
      decfsz      Kount10s
      goto        R10s
      return
;
;1 min delay
;call 60 times of 1 sec delay
Delay1m
      banksel     Kount1m
```

```
      movlw       H'3C' ;60
      movwf       Kount1m
R1m   call        Delay1s
      decfsz      Kount1m
      goto        R1m
      return
;=======================================================
      END
```

Run your program and see if you have the following display with an underscore cursor under 'P' of the fist line with lit backlight as shown below.



2. LCD Displaying: 4-bit Interface Example

Even though 16F877 has an ample amount of I/O pins, it's always wise to save a few pins for future use. Also, if we can achieve with fewer number of I/O pins the same function, there is no reason not to try the economical method. The 4-bit interface method is different from 8-bit interface only how we send the 8-bit data over 8 data lines or 4 data lines.
In 4-bit interface, we separate the 8-bit data by nibbles and send each nibble at a time. Therefore, for coding perspective, the only difference is the change in the subroutines of `instw` and `dataw`. Of course, we have to instruct the LCD module for 4-bit interface instead of 8-bit.

However, there is a slight odd step you have to have before setting the 4-bit interface. The HD44780 requires, for 4-bit interface only, to send the only the high nibble at the first step, and to send the high and low nibbles at the second step. In other words, the setting up for 4-bit interface has, unlike in 8-bit interface, an additional weird step. This is very important. If you miss this first step, you would some weird behavior from the LCD module such as one reset would show proper display and another would not.

The first step for function set for 4-bit interface:
RS=0
<DB7:DB4>=0 0 1 0
Then, the above instruction can be rewritten as:

```
      movlw 0x28
      call  hnibble4
```

with subroutine hnibble4;

```
hnibble4
      movwf       Temp        ;Temp storage
      movf        Temp,0            ;Now W also holds the data
      andlw       0xF0            ; get upper nibble
      movwf       PORTB          ; send data to lcd
```

```
      bcf         PORTB, RS
      bsf         PORTB, E
      call        delay1ms
      bcf         PORTB, E
      call        delay10ms          ;end of high nibble for 4-bit setup
      return
```

The second step for 4-bit interface now can set for for 4-bit, 2-line display, and 5x8 dot matrix:

RS=0

<DB7:DB0>= 0 0 1 0 1 0 X X

Then, the above instruction can be rewritten as (with X=0):

```
            movlw       0x28
            call        instw4
```

However, since we have to separate the byte into two nibbles and send each nibble separately, we have to change the `instw` subroutine to `instw4` subroutine.

```
;subroutine instw4 (4-bit interface instruction write)
;instruction to be written is stored in W before the call
instw4
      movwf       Temp        ;Temp storage
      movf        Temp,0      ;Now W also holds the data
      andlw       0xf0            ; get upper nibble
      movwf       PORTB          ; send data to lcd
      bcf         PORTB, RS
      bsf         PORTB, E
      call        delay1ms
      bcf         PORTB, E
      call        delay10ms          ;end of higher nibble
      swapf       Temp,0             ;get lower nibble to W
      andlw       0xf0
      movwf       PORTB              ;Write to LCD
      bcf         PORTB, RS
      bsf         PORTB, E
      call        delay1ms
      bcf         PORTB, E           ;end of lower nibble
      call        delay10ms
      return
```

Similarly, the data write subroutine `dataw` must also be changed to `dataw4` to reflect the change in data transmission.

```
dataw4
      movwf       Temp        ;Temp storage
      movf        Temp,0      ;Now W also holds the data
      andlw       0xf0            ; get upper nibble
      movwf       PORTB          ; send data to lcd
      bsf         PORTB, RS
      bsf         PORTB, E
      call        delay1ms
      bcf         PORTB, E
      call        delay10ms          ;end of higher nibble
      swapf       Temp,0             ;get lower nibble to W
      andlw       0xf0
      movwf       PORTB              ;Write to LCD
      bsf         PORTB, RS
      bsf         PORTB, E
      call        delay1ms
```

```
        bcf         PORTB, E            ;end of lower nibble
        call        delay10ms
        return
```

Additional change you have to bring to the code is to correctly assign the pins of RW, RS, and E to PORTB. As you see the following 4-bit interface illustration, we use only PORTB for a LCD module.

4-bit Interface

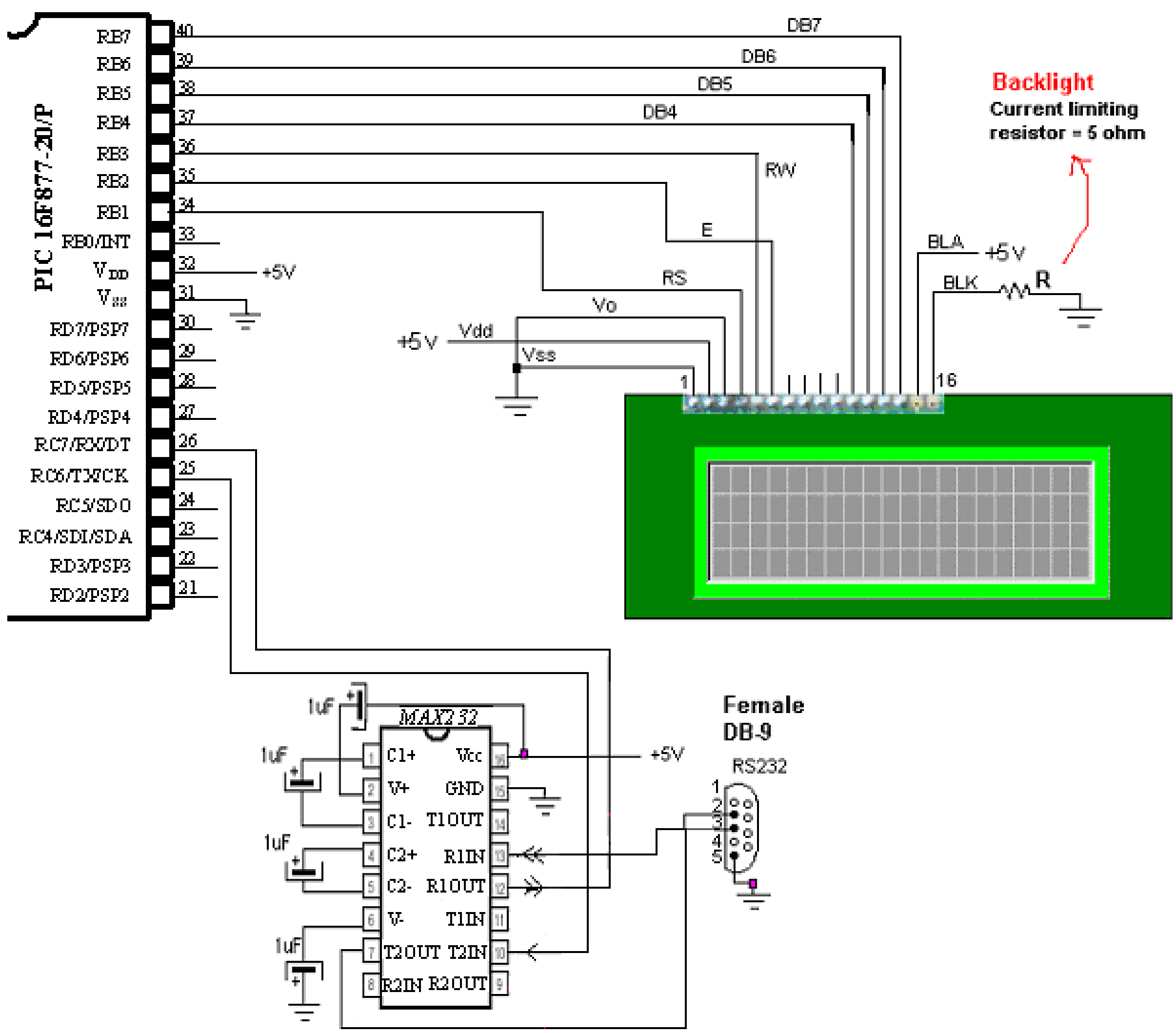
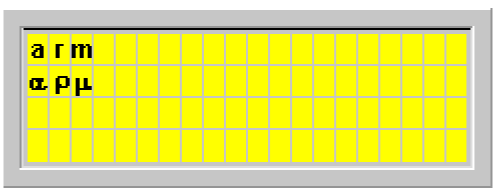


Fig. 26 4-bit Interface Illustration

Special Character Display using Character Generator ROM (CGROM)

The character generator ROM generates 5x8 dot or 5x10 dot character patterns from 8-bit character codes (See the CGROM character codes of HD44780 manual). It can generate 208 5x8 dot character patterns and 32 5x10 dot character patterns. User-defined character patterns are also available by mask-programmed ROM. So we can display even some weird characters. Let's add a few lines of instructions, then, to write a line of Alphabet and a line of symbol (or Greek) equivalent. From the CGROM map, we found that α, ρ, and μ are at E0, E6, and E4, respectively. So by the following instruction should display the example display illustrated after the code.

```
;display a, r, m at line 1
;alpha, rho, and mu at line 2
;Set DDRAM ADDRESS for line 1
      movlw 0x80        ;00
      call  instw4
      movlw 'a'
      call  dataw4
      movlw 'r'
      call  dataw4
      movlw 'm'
      call  dataw4
;Set DDRAM ADDRES for line 2
;CGROM address for alpha, rho, and mu are E0, E6, and E4, respectively
      movlw 0xC0        ;00
      call  instw4
      movlw 0xE0
      call  dataw4
      movlw 0xE6
      call  dataw4
      movlw 0xE4
      call  dataw4
```



3. LCD Displaying -Serial LCD

As discussed above, we know that a LCD module with internal controller/driver provides all the functions such as display RAM, character generator, and liquid crystal driver, required for driving a dot-matrix liquid crystal display, and either 11 lines or 7 lines of processor are needed to interface with the controller/driver of the LCD module. However, to many a hobbyist and students, the control of the controller/driver following the timing diagram suggested in the manual of the module or the controller/driver seems to be a lot of trouble. Also, the requirement of many pins causes some burden for certain processors with fewer I/O pins.

Because either of many pins required for connection or of rather a complex control scheme (at least, by just reading a multi-page control instruction provided by the manufacturer of the LCD module, or by the lack of such instruction), many sought an easier alternative approach. A popular solution to this search is a so-called serial LCD module which requires only one pin (actually three, including +5V and GND connections). A serial LCD module has, in addition to the LCD controller/driver, a convert chip which coverts serial data into a parallel data and signals necessary for the controller/driver. The converter is actually a serial-in/parallel-out shift register, which uses the synchronous serial data pin to load a serial stream of data. Of course, the shift register and accessory circuit can be replaced by a microcontroller for better and simpler control of the LCD module. For example, Scott Edward (Seetron.com)'s Serial Backpack® adopts PIC 16C622. Similarly, Peter Anderson (phasnderson.com)'s cheaper Basic Serial LCD kit employs a PIC processor, PIC16C554.

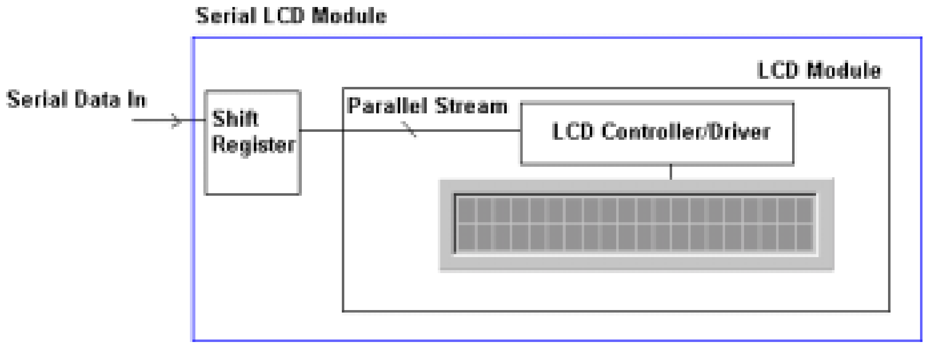


Fig. 27 Serial LCD Module

The discussion and example code follows will be centered on the Serial LCD BPP420 by Scott Edward. For other serial LCD module, closely follow the manual for the module. The BPP 420 package consists of the LCD Backpack and a LCD module by Truly which we thoroughly examined before. So this may give you a stark contrast of controlling the same module by two different method. According to the manual of BPP420, by toggling, we can get 2400 or 9600 bps serial communication speed. The change of the selection is effective when power is on. In other words, we have to select before applying power to the module. The dip switch for speed selection is at the back side of the Backpack. If you do not touch the dip switch, the selected speed is 2400 bps. At the back also is a 5-pin header. However, we need only three pins: +5V, GND, and SER. SER is the single line from 16F877 for instruction/data write to the LCD module. The serial communication format is with the normal 8N1: 8-bit data, no-parity, with 1 stop bit.

Now let's check how to operate this serial LCD by examining the manual of the module. Here goes some precaution that must be exercised. The BPP420 used in the example may be somewhat different from what one gets. The serial convert (Serial Backpack) attached to the LCD module is so-called "old version" made in later 1990s. The control is a little complex than the current version. The apparent difference in hardware is that the old version uses 4-bit interface while the current version uses 8-bit interface. The easiest way to know is to check if all 8 data pins (pin No. 7 – 14) are all connected to the processor chip of the board of the Serial Backpack. If all 8 pins are connected to the chip, you are holding a new version. The old version connects only 4 data pins out of 8 (pin No. 11 – 14). So if you have acquired a new version, follows what the manual (the manual on BPP420 available from seetron.com is good for the new version) indicates. It is assured that the control is much easier. For example, there is no prefix code need to indicate that a following code is an instruction for new version. However, in old version, you have to send a hex number FE before any instruction code. The good manual for old version is the manual for the original serial Backpack. Check seetron.com for the manual. This example follows the manual for the serial Backpack.

Among many control functions provided in the "old version" provided, the following functions are most relevant for normal use of LCD:

Function	**Code (Hex)**
Clear LCD	01
Cursor Home (line 1 and position 1)	02
Show Underline Cursor	0E
Show Blinking Block Cursor	0D
Hide Cursor	0C
Move Cursor one character left	10
Move cursor one character right	14
Scroll display one character left (all characters)	18
Scroll display one character right (all characters)	1C
DDRAM Address (Cursor Position) Set	Addr
CGROM Address set	Addr

The DDRAM map (cursor location) for the old version is shown below.

Position	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
First line	80h	11h	82h	83h	84h	85h	86h	87h	88h	89h	8Ah	8Bh	8Ch	8Dh	8Eh	8Fh	90h	91h	92h	93h
Second line	C0h	C1h	C2h	C3h	C4h	C5h	C6h	C7h	C8h	C9h	CAh	CBh	CCh	CDh	CEh	CFh	D0h	D1h	D2h	D3h
Third line	94h	95h	96h	97h	98h	99h	9Ah	9Bh	9Ch	9Dh	9Eh	9Fh	A0h	A1h	A2h	A3h	A4h	A5h	A6h	A7h
Fourth Line	D4h	D5h	D6h	D7h	D8h	D9h	DAh	DBh	DCh	DDh	DEh	DFh	E0h	E1h	E2h	E3h	E4h	E5h	E6h	E7h

Therefore, when we want to type 'A' at the column 1 and line 2, the following command must be called:
1. Write FEh as an instruction prefix
2. Write C0h to mover the cursor (or DDRAM address) the position of line 3 and column 1.
3. Write 'A' for the character

Before we proceed further, let's make our code for 2400 bps serial communication routine. I hope we remember our discussion and example code in Chapter 5 for software-built serial communication.

The biggest and troubles some thing about the serial Backpack is using "inverted" serial communication mode. In other words, usually the TX line in asynchronous mode is high to indicate the idling state and it goes to low to start a communication. The data "1' is represented to High bit and "0" to Low bit. But in "inverted" mode, everything has to be inverted: idling should be Low, and Start bit should go to High to initiate communication. Also, Low for "1" and High for "0". Therefore, the software built serial communication program we had in Chapter 5 should be changed to reflect the "inverted" mode of the serial Backpack. The reason for this unusual approach is because its main application is for Basic Stamp, which can choose "inverted" or "non-inverted" mode, the latter for direct RS-232 connection

From Start bit we send data through a pin (any I/O pin of 16F877) to the SER pin of the serial LCD module. The pulse width for the bit is 1 Baud cycle. 1 Baud cycle for 2400 bps = 417 µs. Since we already made out 100 µs and 120 µs delay routines, with minor error, 417 µs can be made by calling 100µs delay 3 times followed by calling 120µs delay once. Let's call the subroutine for 417 µs pulse width `bps2400`.

Since we have to send LSB first, we have to do the similar rotation, which includes the carry bit. The idea is to move the LSB of the file register where the data is stored to Carry bit, and check the status of the bit. If the carry bit is 1, then we send 1 to SER, for 0, then 0 to SER, for 1 BC seconds. See below for a code section of one bit (LSB first) transmission:

```
;START BIT
      bsf         PORTD, SER        ;Start Bit in Inverted Mode
      call        bps2400           ;430us-long
;Data Bits  (8 bit transmission)
      movlw       0x08        ;8 --->W
      movwf       Bitcount    ;8 data bits
TXNEXT
      bcf         STATUS, CARRY
      rrf         Treg        ;LSB first mode (normal)
      btfsc       STATUS,CARRY
      bcf         PORTD, SER        ;inverted Mode
      btfss       STATUS,CARRY
      bsf         PORTD, SER        ;Inverted Mode
      call        bps2400
      decfsz      Bitcount
      goto        TXNEXT
;STOP BIT
Stop Bit
      bcf         PORTD, SER        ;Inverted Mode
      call        bps2400           ;STOP bit
```

Using the above code we make two subroutines: one for instruction write and the other for data write. Since the above routine can be directly converted to data write (named as `LCDOUT`) because data write does not need a prefix code.

```
;LCD write subroutine (Note: Inverted Mode) ====================
;The 8-bit data to be sent to LCD module is stored in W
LCDOUT
      banksel     Tchr
      movwf       Tchr        ;W --->Treg
      movlw       0x08        ;8-bit
      movwf       Bitcount    ;8 data bits
;send a START bit
      bsf         PORTD, SER
      call        bps2400

TXNEXT
      bcf         STATUS, CARRY
      rrf         Tchr        ;LSB first mode (normal)
      btfsc       STATUS,CARRY
      bcf         PORTD, SER
      btfss       STATUS,CARRY
      bsf         PORTD, SER
      call        bps2400
      decfsz      Bitcount
      goto        TXNEXT
;send STOP bit
      bcf         PORTD, SER
      call        bps2400
      return
```

Since instruction write needs a prefix write and a code write, it involves two writes. In the `LCDcom` subroutine, the prefix is sent out using the `LCDOUT` subroutine followed by actual code (stored in `Tcom` register) write using the same `LCDOUT` subroutine.

```
;LCDCOM subroutine === to send command prefix + command code
LCDcom
      movwf        Tcom  ;command code here
      movlw        0xFE  ;command prefix
      call         LCDOUT
      movf         Tcom, W
      call         LCDOUT
      return
```

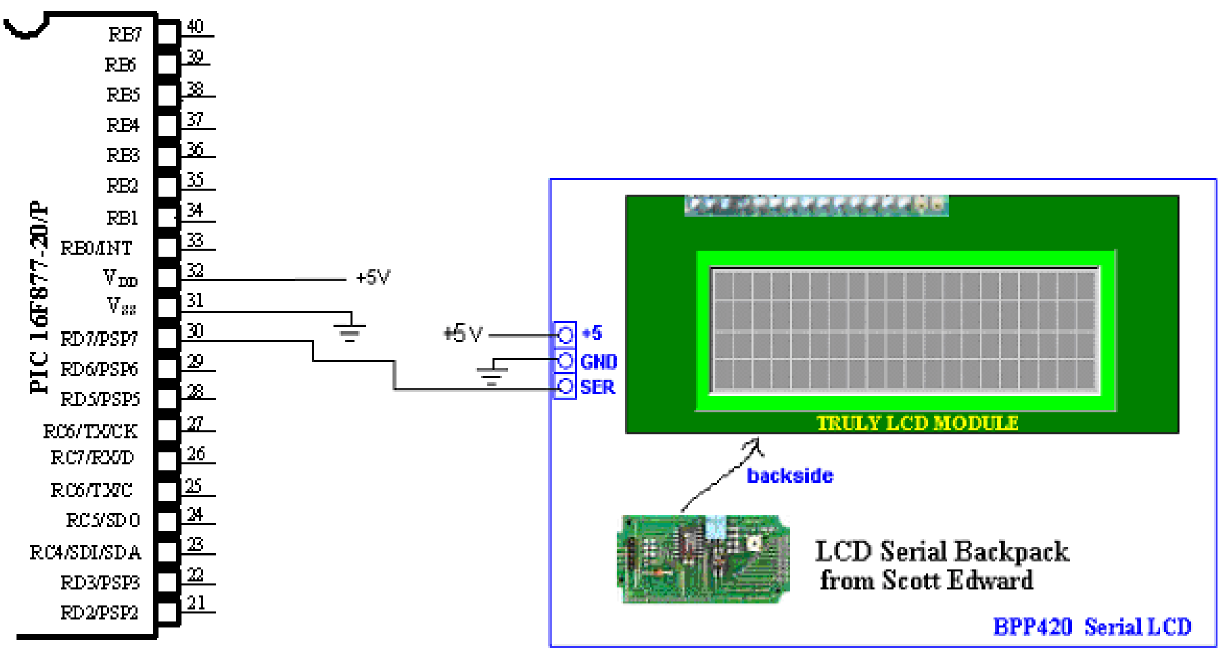


Fig. 28 PIC 16F877-20P connection to BPP420 Serial LCD

Now let's have the following connection which uses PORTD<7> as SER pin for bit transmission. In the example code, we want to display the same format we displayed with regular LCD module. Since all the subroutines are already discussed, only the main part is listed and explained here.

```
;16LCD-s.asm
;
;Serial LCD control Program
;Seetron's BPP420 LCD module (20x4) (OLD MODE: I guess 4-bit interface)
;Software-built Serial communication used
;
; 2400 bps with 8N1 format
; 1 Baud Cycle is then 417 us --->420 us pulse width
;
;F = 20 MHz
;
;SER pin = RD7

        list P = 16F877
```

```
STATUS      EQU   0x03
CARRY       EQU   0x00
TRISD       EQU   0x88
PORTD       EQU   0x08
SER         EQU   0x07        ;RDr for SER out
MSB         EQU   0x07
;
;
      CBLOCK      0x20
            Kount120us   ;Delay count (number of instr cycles for delay)
            Kount100us
            Kount1ms
            Kount10ms
            Kount1s
            Kount10s
            Kount1m
            BitCount
            Tchr        ;temp storage
            Tcom
      ENDC
;

;program should start from 0005h
;0004h is allocated to interrupt handler

      org         0x0000
      goto        START


      org         0x05
START
      banksel     TRISD
; Port setting (1 for input and 0 for output)
; 0000 0000

      movlw       0x00
      movwf       TRISD       ;all outputs

      banksel     PORTD
      clrf        PORTD
      bcf         PORTD, SER  ;(no signal) Inverted Mode

      banksel     Tchr
      clrf        Tchr
      clrf        Tcom

;LCD display started here
      call        delay1s           ;warm-up

      movlw       0x01        ;Clear LCD
      call        LCDcom      ;Usually no time delay required for 2400bps
                        ;when in 9600 apply 1ms time delay after each
                        ;write

      movlw       0x0E        ;Show Underline Cursor
      call        LCDcom
;Position cursor to Line 1 Column 1 ($80)

      movlw       0x80
      call        LCDcom
      movlw       'P'
      call        LCDOUT
      movlw       'I'
      call        LCDOUT
```

```
      movlw       'C'
      call        LCDOUT
;Change the DDRAM address for line 2 and Column 1 ($C0)

      movlw       0xC0        ;DDRAM ADDRESS SET
      call        LCDcom
      movlw       'A'
      call        LCDOUT            ;A
      movlw       'N'
      call        LCDOUT            ;N
      movlw       'D'
      call        LCDOUT            ;D
;Change the DDRAM address (cursor position) to line 3 and column 1 ($94)
      movlw       0x94
      call        LCDcom
      movlw       'L'
      call        LCDOUT
      movlw       'C'
      call        LCDOUT
      movlw       'D'
      call        LCDOUT

;Change the DDRAM address (cursor position) to line 4 and column 1 ($D4)
      movlw       0xD4
      call        LCDcom
      movlw       'D'
      call        LCDOUT
      movlw       'I'
      call        LCDOUT
      movlw       'S'
      call        LCDOUT
      movlw       'P'
      call        LCDOUT
      movlw       'L'
      call        LCDOUT
      movlw       'A'
      call        LCDOUT
      movlw       'Y'
      call        LCDOUT
```

Also, we can display special characters stored in the CGROM. Since the LCD module is the same, the location of Greek characters α, ρ, and μ are the same: E0, E6, and E4, respectively. Then, the following code with display the same display format as we did with the regular LCD module: `arm` at the first line and αρμ at the second line.

```
;Clear
      movlw       0x01
      call        LCDcom
;Hide cursor
      movlw       0x0C
      call        LCDcom
;Line 1 column 1
      movlw       0x80
      call        LCDcom
;Write arm in English
      movlw       'a'
      call        LCDOUT
      movlw       'r'
      call        LCDOUT
      movlw       'm'
      call        LCDOUT
;move to line 2 column1
```

```
      movlw       0xC0
      call        LCDcom
;get the special character
;alpha (E0), rho (E6), mu (E4)
;CGROM access
      movlw       0xE0
      call        LCDOUT
      movlw       0xE6
      call        LCDOUT
      movlw       0xE4
      call        LCDOUT
```

Now we examined LCD modules and serial LCD modules, and programmed example codes. Now it is up to you whether you go with the regular LCD module and a series LCD module depending upon your budget (the serial one costs much more) or your I/O pin availability. For programming perspective, there is not much difference between two modules.

4. Decoding IR Remote Controller

IR may be the cheapest way to remotely control a device within a visible range. Almost all audio and video equipment are controlled this way nowadays. Due to this wide spread use, the required components are quite cheap.

Let's extend our interest of serial communication, especially software enabled one, to decode TV or VCR Infrared (IR) remote controller. We cannot directly use the code in Chapter 5 since remote controllers use different protocols. However, the protocols are all based on serial communication, the principle of the operation is the same. In the application, we will read the IR information, sent by a remote controller, using a IR receiver module (that means it is not just an IR detector but a receiver with 40KHz demodulation circuit inside the module. Details on this follows.)

Modulation is a way to make signal stand out above noise. With modulation , IR light source blinks in a particular frequency, say 40KHz. The IR receiver should be tuned to that frequency, so it can ignore everything else.

Infrared remote controls are using a 32-40 kHz modulated square wave for communication. These circuits are used to transmit a 1-4 kHz digital signal through infra light (also, this is the maximum attainable speed, 1000-4000 bits per sec). The transmitter oscillator which is driving the infrared transmitter LED can be turned on/off by applying a logic level voltage. For us, the remote controller is the transmitter. Therefore our attention is toward more on the IR receiver.

On the receiver side a photodiode takes up the signal. The integrated circuit inside the chip is sensitive only around a specific frequency in the 32-40 kHz range. The output is the demodulated digital input. All these element in a case form an IR receiver module. The output of the module is High when there is no IR signal, Low when there is IR signal.

As illustrated below, there are several IR receiver modules available in very cheap price.

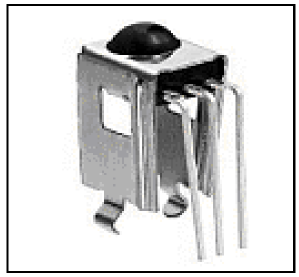


Fig. 29(a) Sharp GP1U581Y IR Receiver Module Fig.29(b) Radio Shack's IR receiver.

Sharp GP1U581Y IR receiver module is the most popular IR receiver. It is designed for use with 38khz modulated IR sources. It incorporates an amplifier, limiter, band pass filter, demodulator, integrator and comparator. Radio shack's IR receiver (Catalog #: 276-640) is also good for experimental projects and building remote control. It works with voltage in 2.4 – 5.5V. It's elliptical lens helps to block light noise from above and below the center frequency of 38kHz.

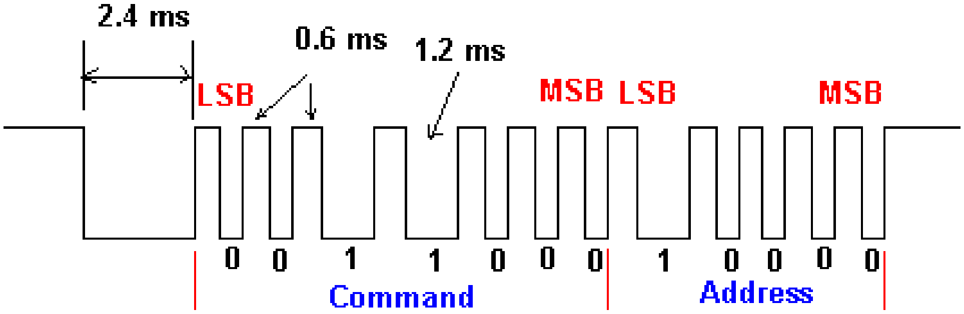
To detect IR signal from a remote controller is to know how different remote controllers send information. And this is the subject of IR protocol. Basically there are three types of IR protocols: pulse coded protocol, space coded protocol, and shift coded protocol.

Pulse coded protocol is to use the varying length of a pulse to represent either 0 or 1. Sony protocol is one of the pulse coded protocols. Space coded protocol uses the length of a space between pulses to represent either 0 or 1. Sharp TV/VCR remote control uses this space coded protocol. In shit coded protocol, the direction of transitions represent either 0 or 1, and the all the bits have a constant time period. Philips remote controller uses this shift coded protocol.

We will consider here only for Sony and Sharp protocols.

<u>Sony Protocol</u>

Sony protocol is consistent of pulse coded 12-bit information with carrier frequency of 40 KHz. The code starts from a 2.4ms start bit. Out of 12-bit information, 5 bits are assigned for address to indicate different device such as TV, VCR, or DVD and the other 7 bits are assigned for command to indicate the buttons on the remote controller. The pulse widths (or space) are 1.2 ms for "1" and 0.6 ms for "0". Commands are repeatedly transmitted from the remote controller every 45 ms as long as a key is held down. As in normal serial communication, LSB is sent first and the MSB last for both address and command.

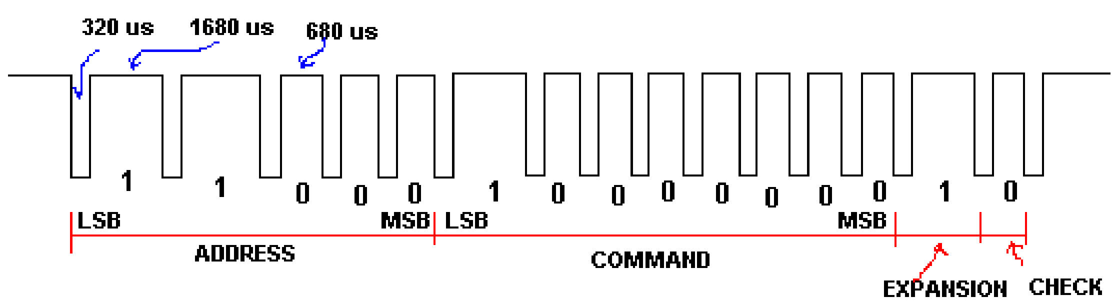


The table below lists some messages sent by Sony remote controls in the 12-bit protocol.

ADDRESS		**COMMAND**	
1	TV	6	Key "7"
2	VCR1	7	Key "8"
3	VCR	8	Key "9"
6	LDP	9	Key "0"
COMMAND		16(10h)	Channel +
0	Key "1"	17(11h)	Channel -
1	Key "2"	18(12h)	Volume +
2	Key "3"	19(13h)	Volume -
3	Key "4"	20(14h)	Mute
4	Key "5"	21(15h)	Power
5	Key "6"	22(16h)	Reset

<u>Sharp Protocol</u>

Sharp protocol for Sharp VCR uses 13-bit protocol with carrier frequency of 38KHz. There are two trailing additional bits for expansion and check. These two bits are not used in decoding. The first 5 bits are for address and the second 8 bits are for command. The "1" and "0" representation is done by the length of a distance between two pulses, the pulse distance: pulse distance of 0.68 ms is for "0" and distance of 1.68ms for "1". The pulses which separate the distances are 0.32 ms long. One key press sends the code twice separated by 40 ms time delay.



The table below lists some messages sent by Sharp VCR remote controller.

ADDRESS		**COMMAND**	
3	VCR	7	Key "7"
		8	Key "8"
		9	Key "9"
		10(0Ah)	Key "0"
COMMAND		17(11h)	Channel +
1	Key "1"	18(12h)	Channel -
2	Key "2"	34(22h)	Play
3	Key "3"	39(27h)	Stop
4	Key "4"	33(21h)	Fast Forward
5	Key "5"	35(23h)	Rewind
6	Key "6"	40(28h)	Recording

Hardware Implementation

Let's connect a Sharp IR receiver module (this works both for Sony and Sharp) at the RB7 port (Pin #40) of 16F877. The pin arrangement is common to most IR receiver modules: Ground and Signal Out pins are separated by the Vcc pin. You provide +5V source to the Vcc pin to activate the module. The Ground pin can be found easily since the Ground pin is internally connected to the metal case. So a pin connected to the metal part of the case is the Ground pin. The output, in hexadecimal number, will be displayed on a monitor and will be compared with the command/address list tables for Sony and Sharp.

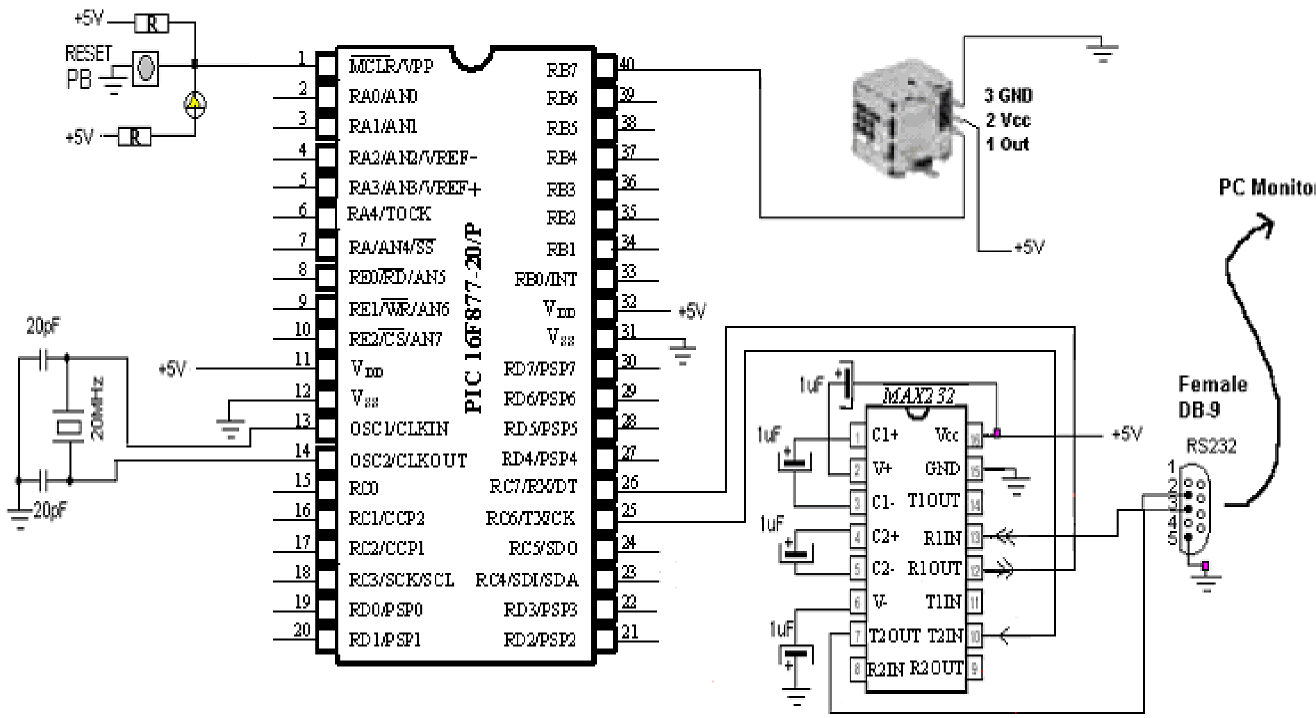


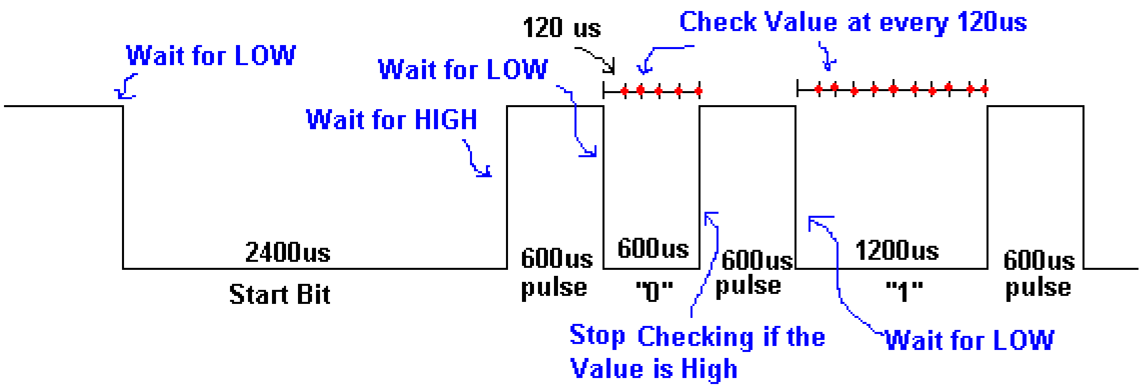
Fig. 30 PIC 16F877/20-P connection to Sharp IR receiver module

Programming for Sony Remote Controller

As the Sony protocol and the code, we see that a digit of "1" is represented by a 1200µs space and "0" by 600µs space. Also, we should remember that the code starts with a Start space for a length of 2400µs. Therefore, we have to follow a sequence of reading the RB7 pin (the output of the IR receiver module). The detection of the Start big goes like the following sequence:

1. Check the RB7 pin.
2. If the RB7 is High. Go to 1. IF RB7 is Low, proceed.
3. Wait until RB7 goes back to high.
4. IR reading begins from here.

Once the Start bit is detected, as listed above, we wait for the first pulse of 600 µs. If pulse goes to Low, we have to measure the space until the next pulse. How do we measure the space in time? The approach we choose here is to use and extend the time delay (using only instructions not timer module of 16F877) we studied before. In other words, how many time delays of 120µs are in the space will determine the space length. We do not need exact length of time. What we need is just a comparison figure. The figure for comparison is, say, Pcount which is the number of 120µs time span in a space. For "0" (600 µs), Pcount must be less than 7, and for "1" (1200 µs), it must be bigger than 8. Since the number 7 is lower than 8, we could finally have the following comparison of Pcount for "1" and "0" determination: A space with Pcount less than 8 is "0" and a space with Pcount of 8 or more is "1".



The actual reason of choosing the number 8 comes from the easiness of the comparison. We know that the maximum possible Pcount is 1200÷120=10 (0A). Therefore, for "1" the bit-3 (B3) of the Pcount must be 1. For "0" , since the count is less than 8, the B3 of Pcount would be 0. In other words, by checking the third bit of the Pcount, we can easily determine its representation, as illustrated in the code below.

```
            btfsc       Pcount,0x03
            bsf         COMreg, MSB          ;assuming that the MSB
                                             ;is already cleared above.
                                             ;if B3=1, it is "1"
            bcf         STATUS, CARRY        ;other wise, keep the previous value
            rrf         COMreg               ;Rotate right with value 0
```

The delay building block for IR decoding program is the 120 µs time delay. As we discussed before, since 1 instruction cycle in 20MHz clock takes 0.2 µs, for 120 µs, there must be 600 instruction cycles. Therefore, from 600 =197*3 +9, the repetition count for 120 µs, `Kount120us`, is selected as 197(C5h). The subroutine goes like this:

```
;subroutine delay120us
Delay120us
            banksel     Kount120us
            movlw       0XC5                 ;197d
            movwf       Kount120us
R120us      decfsz      Kount120us
            goto        R120us
            return
;end of subroutine
```

As we did before, 100µs delay can be calculated, as we need 500 instruction cycles, from 500 =164*3 +8. So the repetition count, `Kount100us`, is selected as 164(A4h). The subroutine goes like below:

```
Delay100us
            banksel     Kount100us
            movlw       0xA4        ;164d
            movwf       Kount100us
R100us      decfsz      Kount100us
            goto        R100us
            return
```

Similarly, but very conveniently, other time delays can be made from the building block. For example, a 10ms delay can be generated by calling the 100µs delay subroutine for 100 times. Here goes the subroutine for 10ms delay.

```
;10ms delay subroutine
; call 100 times of 100 us delay
Delay10ms
            banksel     Kount10ms
            movlw       0x64              ;100d
            movwf       Kount10ms
R10ms       call        delay100us
            decfsz      Kount10ms
            goto        R10ms
            return
```

So this is a pseudo-code for Sony remote controller decoding program:

1. Begin
2. If RB7 is LOW (this means an IR transmission is already undergoing), give enough delay time, say 200ms, not to read on-going data stream or the second command/address stream.
3. When RB7 is back to High, wait for Start bit.
4. After Start bit detection, wait for a pulse to arrive.
5. After each pulse, count number of 120 µs delays at a space.
6. Determine the bit value (0 or 1) and rotate to the right a bit. (Remember that the LSB arrives first)
7. Repeat 5-6 for seven times for 7-bit Command. Rotate to the right one last time for an 8-bit result.
8. Repeat 5-6 five times for 5-bit Address. Rotate to the right 3 times for an 8-bit result.

Now, let's have an example code. Read each line of instruction and comments to follow the logic of IR decoding. In addition to the decoding, the decoded contents in two hex numbers (one for Address and the other for Command) in two digits are displayed on a monitor.

```
;4IR-sony.asm
;
;This program is to:
;1. Read IR data from a SONY IR Receiver module
;   sent from a Sony VCR remote controller (12 bit protocol)
;2. Display the data in ASCII format on a PC screen
;
;
; Sony IR remote protocol (12-bit version):
;0. When no button is pressed, the output from the IR receiver
;   is kept HIGH
;1. Pulse Width Encoding Method
;2. When button is pressed, a 2400 uS LOW starts the serial communication
;3. 1/0 code is separated by 600 uS long HIGH pulse separator
```

```
;   "1": 1200 uS long LOW followed by a pulse separator
;   "0": 600 uS long LOW followed by a pulse separator
;4. Encoding Order
;    7-bit command followed by 5-bit Address
;5. The end is marked by HIGH
;6. LSB first mode
;
;IR-RX pin(IRX) is connected to RB7 port
;
;
;IR-RECEPTION AND DECODING
;Here's the way to read and decode the IR
;1. Detect the IRX for LOW
;2. Wait until IRX goes to HIGH
;3. Wait for 120uS
;4. Check IRX (Add IRCounter if IRX=HIGH)
;   IF IRX=LOW goto 2.

; if IRCOUNT <8 : "0"
; if IRCOUNT >8 : "1"
;
; Repeat 7 times for Command --->COMreg is the result register
; Repeat 5 times for ADDRESS --->ADDRreg is the result register
;
;Terminal set up: 8N1 19200
;

        list P = 16F877

STATUS      EQU   0x03
CARRY       EQU   0x00
ZERO        EQU   0x02
TRISB       EQU   0x86
PORTB       EQU   0x06
TXSTA       EQU   0x98        ;TX status and control
RCSTA       EQU   0x18        ;RX status and control
SPBRG       EQU   0x99        ;Baud Rate assignment
TXREG       EQU   0x19        ;USART TX Register
RCREG       EQU   0x1A        ;USART RX Register
PIR1        EQU   0x0C        ;USART RX/TX buffer status (empty or full)
RCIF        EQU   0x05        ;PIR1<5>: RX Buffer 1-Full  0-Empty
TXIF        EQU   0x04        ;PIR1<4>: TX Buffer 1-empty 0-full
TXMODE      EQU   0x20        ;TXSTA=00100000 : 8-bit, Async mode
RXMODE      EQU   0x90        ;RCSTA=10010000 : 8-bit, enable port, enable RX
BAUD        EQU   0x0F        ;19200 bps
MSB         EQU   0x07
IRX         EQU   0x07        ;RB7 for IR receiver
;

;RAM Area for file registes

      CBLOCK      0x20

            Kount120us   ;Delay count for 120us delay
            Kount100us
            Kount1ms
            Kount10ms
            Kount1s
            Kount10s
            Kount1m
            first
            second
            third
            Bitcount          ;data bit count
```

```
            Kount               ;
            ADDRreg             ;IR ADDRESS register
            COMreg              ;IR Command register
            Pcount              ;HIGH duration count
            Adcount             ;count for ADDRESS
            Cmcount             ;count for COMMAND
            Tcount
            ADDRtemp            ;for ASCII conversion of ADDRESS
            COMtemp             ;for ASCII conversion of COMMAND
            ADDR1               ;First hex digit for ADDRESS reg
            ADDR2               ;Second hex digit
            COM1                ;First hex digit for COMMAND reg
            COM2                ;Second hex digit
            ASCIIreg            ;Temporary register for H-to-A conversion
      ENDC

;bootloader accommodation
      org         0x0000            ;line 1
      goto        START             ;line 2 ($0000)
;
      org         0x05
START
      banksel     TRISB
      movlw       0x80        ; 1000 0000  (RB7 [IRX] as input)
      movwf       TRISB

BEGIN
      banksel     ADDRreg     ;clear all file registers
      clrf        ADDRreg
      clrf        COMreg
      clrf        Pcount      ;pulse count for space measurement
; CHECK IF THE IRX is HIGH at least for 200 mS
; to make sure it does not read on-going or the second stream
; For 200 mS delay, call 10ms delay for 20 times.
      banksel     PORTB
      btfss       PORTB, IRX
      goto        BEGIN             ;if IRX is LOW, go to start again
                                    ;to wait until the current on-going
                                    ;data stream is over
      banksel     first             ;if IRX is high, then give enough
                                    ;delay to read fresh start IR stream
      movlw       0x14
      movwf       first
redo  call        Delay10ms

      btfss       PORTB, IRX        ;for continuous 200 ms
      goto        BEGIN
      decfsz      first
      goto        redo
                                    ;NOW ready to fresh read IR data

jam
      banksel     PORTB
      btfsc       PORTB, IRX              ;Wait for START bit
      goto        jam

      banksel     CMcount           ;now start bit is detected
      movlw       0x07
      movwf       CMcount           ;command has 7 bits
WAIT  btfss       PORTB, IRX        ;wait until the Start bit goes to High
      goto        WAIT
CMNEXT
      clrf        Pcount            ;now, we are in the first pulse
      bcf         STATUS, CARRY     ;Clear the Carry Bit
```

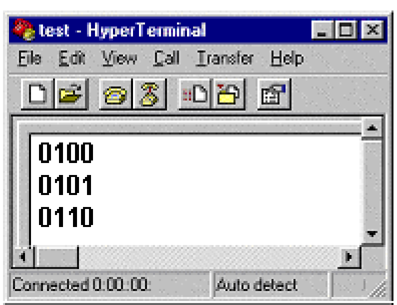
```
      rrf          COMreg             ;COMMAND<7>=0

wait2 btfsc        PORTB, IRX         ;Wait for the pulse to go to LOW
                                      ;(the space)
      goto         wait2
DST   call         Delay120us         ;We are in space (IRX is LOW NOW)
                                      ;Delay 120 uS to measure the space length

wait3 btfsc        PORTB, IRX
      goto         onezero            ;until the end of space
      incf         Pcount             ;if IRX still HIGH, increase the count
      goto         DST                ;repeat

;Here we counted the number of 120us time delays in the space.
;Let's determine the bit value of the space
;"1" or "0" determination
;
onezero
      btfsc        Pcount,0x03        ;B3=1 or 0  (bigger than 7?)
      bsf          COMreg, MSB        ;B3=1, then COMreg<7>=1
      decfsz       CMcount            ;B3=0, then COMreg<7>=0 the old value
      goto         CMNEXT             ;Have we done 7 times?  If not, do again

      bcf          STATUS, CARRY      ;Yes we read 7 spaces
      rrf          COMreg             ;Fill the 7th bit with 0 to make a byte.
;THE END OF 7-BIT COMMAND READING

;ADDRESS READING Begins here
      movlw        0x05
      movwf        Adcount            ;ADDRESS has 5 bits
ADNEXT
      clrf         Pcount
      bcf          STATUS, CARRY      ;Clear the Carry Bit
      rrf          ADDRreg            ;rotate to the right
cwait2
      btfsc        PORTB, IRX         ;Does the pulse go to LOW to space?
      goto         cwait2
cDST  call         Delay120us         ;In space. Delay 120 uS

cwait3
      btfsc        PORTB, IRX
      goto         conezero           ;End of space, then "1" or "0" check
      incf         Pcount             ;If IRX still LOW, increase Pcount
      goto         cDST               ;repeat
;"1" or "0" check
;
conezero
      btfsc        Pcount,0x03        ;B3=1 or 0?
      bsf          ADDRreg, MSB       ;If B3=1, ADDRref<7>=1
      decfsz       Adcount            ;If B3=0, keep the old value
                                      ;Have we read 5 times?
      goto         ADNEXT             ;No.  Then, do more.

;
ewait4
      btfss        PORTB, IRX         ;Is it now end of the data stream?
                                      ;with IRX High?
      goto         ewait4
;THE END OF ADDRESS READING
      bcf    STATUS, CARRY            ;We have to fill the 3 MSBs with 0
      rrf    ADDRreg                  ;to make a byte information
      rrf    ADDRreg
      rrf    ADDRreg
;THE END OF ADDRESS READING
```

```
;ADDRreg holds the ADDRESS Info
;COMreg holds the COMMAND Info
;
;ASCII Converion of ADDRreg and COMreg
;
      movf        ADDRreg,0
      movwf       ADDRtemp
      swapf       ADDRtemp,0    ;SWAP upper and lower nibbles --->W

      andlw   0x0F                  ;Mask off upper nibble


;; === hex to ascii conversion subroutine
;move the content to W before call this routine
;final result will be stored back to W

      call        HTOA
      movwf       ADDR1       ;First Hex Digit of ADDRESS

      movf        ADDRreg,0
      andlw       0x0F         ;mask of upper nibble
      call        HTOA
      movwf       ADDR2       ;Second Hex Digit of ADDRESS

      movf        COMreg,0
      movwf       COMtemp
      swapf       COMtemp,0    ;SWAP upper and lower nibbles --->W

      andlw       0x0F        ;Mask off upper nibble

      call        HTOA
      movwf       COM1        ;First Hex Digit of COMMAND

      movf        COMreg,0
      andlw       0x0F       ;mask of upper nibble
      call        HTOA
      movwf       COM2        ;Second Hex Digit of COMMAND

;
      call  ASYNC_mode        ;Enable the Serial Communication

;TX ROUTINE FOR ADDR INFO

      movf        ADDR1,0
      call        Txcall              ;First Hex Digit of ADDRESS display
      movf        ADDR2,0             ;Followed by 2nd digit
      call        TXcall
      movf        COM1,0
      call        TXcall              ;Followed by the first digit of COMMAND
      movf        com2,0
      call        Txcall              ;followed by the 2nd
;add one line as a delimiter
      call  CRLF                      ;ends with Carriage Return and Line Feed
                                      ;which moves the cursor to the first
                                      ;column of the next line.
      goto  BEGIN                     ;REPEAT

;===SUBROUTINES ====
;=====================================================
;RX TX Initialization with Async Mode
;Async_mode Subroutine
Async_mode
      banksel     SPBRG
      movlw       baud                ;B'00001111' (19200)
```

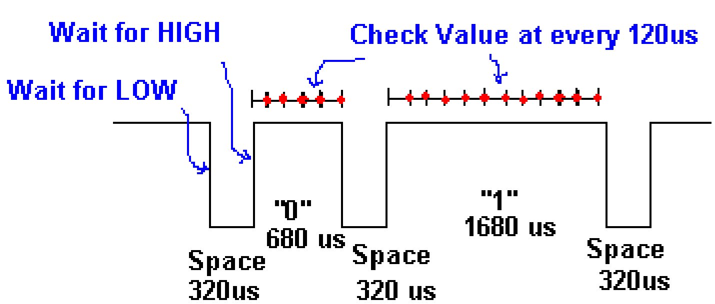
```
      movwf       SPBRG
      banksel     TXSTA
      movlw       TXMODE            ;B'00100000' Async Mode
      movwf       TXSTA
      banksel     RCSTA
      movlw       RXMODE            ;B'10010000' Enable Port
      movwf       RCSTA
      return
;========================================================================
TXCALL
      banksel     PIR1
      btfss       PIR1, TXIF  ; Check if TX buffer is empty
      goto        TXCALL
      banksel     TXREG
      movwf       TXREG       ; Place the character to TX buffer
      return
;===
CRLF
      banksel     PIR1
      btfss       PIR1, TXIF
      goto        CRLF
      banksel     TXREG
      movlw       0x0D        ;ASCII code for CR
      movwf       TXREG
LFkey
      banksel     PIR1
      btfss       PIR1, TXIF
      goto        LFkey
      banksel     TXREG
      movlw       0x0A        ;ASCII code for LF
      movwf       TXREG
      return
;============================================================
;DELAY SUBROUTINES
Delay120us
      banksel     Kount120us
      movlw       0xC5                    ;D197d
      movwf       Kount120us
R120us
      decfsz      Kount120us
      goto        R120us
      return
;
Delay100us
      banksel     Kount100us
      movlw       0xA4
      movwf       Kount100us
R100us
      decfsz      Kount100us
      goto        R100us
      return

;
Delay10ms
      banksel     Kount10ms
      movlw       0x64              ;100d
      movwf       Kount10ms
R10ms call        delay100us
      decfsz      Kount10ms
      goto        R10ms
      return
;
;; === hex to ascii conversion subroutine
;move the content to W before call this routine
```

```
;final result will be stored back to W
HTOA
      movwf       ASCIIreg
;check 0-9 or A-F
      btfsc       ASCIIreg, 0x03
      goto        RECHK
THIRTY
      movlw       0x30
      addwf       ASCIIreg
      movf        ASCIIreg,0
      return

RECHK andlw       0x06
      btfsc       STATUS,ZERO
      goto        THIRTY
      movlw       0x37
      addwf       ASCIIreg
      movf        ASCIIreg,0
      return
;
;END OF CODE
        END
```

If you run the above code, you would have the hex numbers displayed on your monitor as illustrated below, when you sequentially press keys of "1", "3", and "Channel +" of your Sony TV remote controller.



<u>Programming for Sharp Remote Controller</u>
The programming for a Sharp remote controller is not different from that for a Sony remote controller. In 13-bit Sharp protocol, however, the separator is a space of Low, and the "1" or "0" representation is determined by the length of a pulse of High. The separating space is 320 µs long and pulse length for "1" is 1680 µs, for "0" 680 µs. There is no lengthy Start bit in Sharp protocol and Address comes before Command. The Start bit is just a space of Low.



The 120us time delay we used for Sony protocol is used for Sharp protocol to measure the length of a pulse for "1" or "0" determination. Also, the Pcount with "1" or "0" determination utilizing the 3rd bit applies here without change.

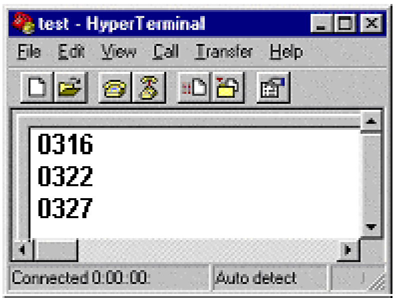
Now, let's have an example code for Sharp protocol. Read each line of instruction and comments to follow the logic of IR decoding in Sharp remote controller. As before, the decoded contents in two hex numbers (one for Address and the other for Command) in two digits are displayed on a monitor. Since the declaration part and the subroutine part are the same as that of Sony protocol code, here shows only the main part.

```
;for Sharp VCR Remote Controller
;Here's the way to read and decode the IR
;1. Detect the IRX for LOW
;2. Wait until IRX goes to HIGH
;3. Wait for 120uS
;4. Check IRX (Add IRCounter if IRX=HIGH)
;   IF IRX=LOW goto 2.

; if IRCOUNT <8 : "0"
; if IRCOUNT >10 : "1"
;
; Repeat 5 times for Address --->ADDRreg
; Repeat 8 times for Command --->COMreg
; Repeat 2 times for EXP and CHK ---> Do not store. Ignore them.
;
;=========================================================
      org         0x0000            ;line 1
      goto        START             ;line 2 ($0000)
;======================================================
      org         0x05
START

      banksel     TRISB
      movlw       0x80
      movwf       TRISB                   ;RB7 - IRX Pin (IN)


BEGIN
      banksel     TXREG
      clrf        TXREG
      banksel     ADDRreg
      clrf        ADDRreg
      clrf        COMreg
      clrf        Pcount

      banksel     PORTB
      btfss       PORTB, IRX
      goto        BEGIN       ;if IRX is LOW, start again
;call delay10ms 20 times
      banksel     first
      movlw       0x14        ;20
      movwf       first
redo  call        Delay10ms   ;200mS delays
;check again for IRX
      btfss       PORTB, IRX
      goto        BEGIN
      decfsz      first
      goto        redo
;NOW ready to fresh read IR data
;
      movlw       0x05
```

```
      movwf       ADcount
; Check for START bit
ADNEXT
      clrf        Pcount
      bcf         STATUS, CARRY     ;Clear the Carry Bit
      rrf         ADDRreg           ;rotate to the right
WAIT  btfsc       PORTB, IRX        ;IRX=LOW?
      goto        WAIT              ;NO
wait2 btfss       PORTB, IRX        ;YES. Then check it AGain
      goto        wait2
DST   call  Delay120us              ; Delay 120 uS
;count (or measure) the HIGH duration
wait3 btfss       PORTB, IRX
      goto        onezero
      incf        Pcount
      goto        DST

;
;"1" or "0" check
;
onezero
      btfsc       Pcount,0x03
      bsf         ADDRreg, MSB
      decfsz      ADcount
      goto        ADNEXT

      bcf         STATUS, CARRY
      rrf         ADDRreg
      rrf         ADDRreg
      rrf         ADDRreg

;Now COMMAND READING
      movlw       0x08
      movwf       CMcount
; Check for START bit
CMNEXT
      clrf        Pcount
      bcf         STATUS, CARRY     ;Clear the Carry Bit
      rrf         COMREG            ;rotate to the right

cwait2
      btfss       PORTB, IRX  ;YES. Then check it AGain
      goto        cwait2
cDST  call        Delay120us
;count (or measure) the HIGH duration
cwait3
      btfss       PORTB, IRX
      goto        conezero
      incf        Pcount
      goto        cDST
;"1" or "0" check
;
conezero
      btfsc       Pcount,0x03
      bsf         COMreg, MSB
      decfsz      CMcount
      goto        CMNEXT


;read next two more data for EXP and CHK

ewait2
      btfss       PORTB, IRX  ;YES. Then check it AGain
      goto        ewait2
```

```
ewait3
      btfsc       PORTB, IRX
      goto        ewait3
ewait4
      btfss       PORTB, IRX
      goto        ewait4
;
;
;now send the IR info

;ASCII Converion of ADDRreg and COMreg
;
      movf        ADDRreg,0
      movwf       ADDRtemp
      swapf       ADDRtemp,0    ;SWAP upper and lower nibbles --->W

      andlw       0x0F        ;Mask off upper nibble


;; === hex to ascii conversion subroutine
;move the content to W before call this routine
;final result will be stored back to W

      call        HTOA
      movwf       ADDR1

      movf        ADDRreg,0
      andlw       0x0F       ;mask of upper nibble
      call        HTOA
      movwf       ADDR2

      movf        COMreg,0
      movwf       COMtemp
      swapf       COMtemp,0    ;SWAP upper and lower nibbles --->W

      andlw       0x0F        ;Mask off upper nibble

      call        HTOA
      movwf       COM1

      movf        COMreg,0
      andlw       0x0F       ;mask of upper nibble
      call        HTOA
      movwf       COM2
;rx TX SET UP
      call        ASYNC_mode

;TX ROUTINE FOR ADDR INFO

      movf        ADDR1,0
      call        TXcall
      movf        ADDR2,0
      call        TXcall
      movf        COM1,0
      call        TXcall
      movf        com2,0
      call        TXcall
;add one line as a delimiter

      call        CRLF
      goto        BEGIN
```

If you run the above code, you would have the hex numbers displayed on your monitor as illustrated below, when you sequentially press keys of "Power", "Play", and "Stop" of your Sharp VCR remote controller.



6. Remote Control of LED On/Off by Sony Remote Controller

Since we learned how to read an IR remote controller, we can now apply it to remotely control a device. A simple way to do is to turn on and off an LED by the IR remote controller. For this remote LED control, we connect an LED through a register. The value of resistor can be any value like 1KΩ or 2KΩ. If you prefer brighter light, reduce the resistance to 470Ω or 330Ω or even 100Ω. As shown below the LED is connected to the RD1 port. High output from RD1 pin turns on the LED, and Low turns off the LED. The remote control action we install is to change the length of LED-on period depending upon the numeric key of a Sony TV remote controller. In other words, if you press key "1", it would turn the LED on for 1 second. Key "9" would turn the LED for 9 seconds. All other keys are ignored and the LED would be kept off.

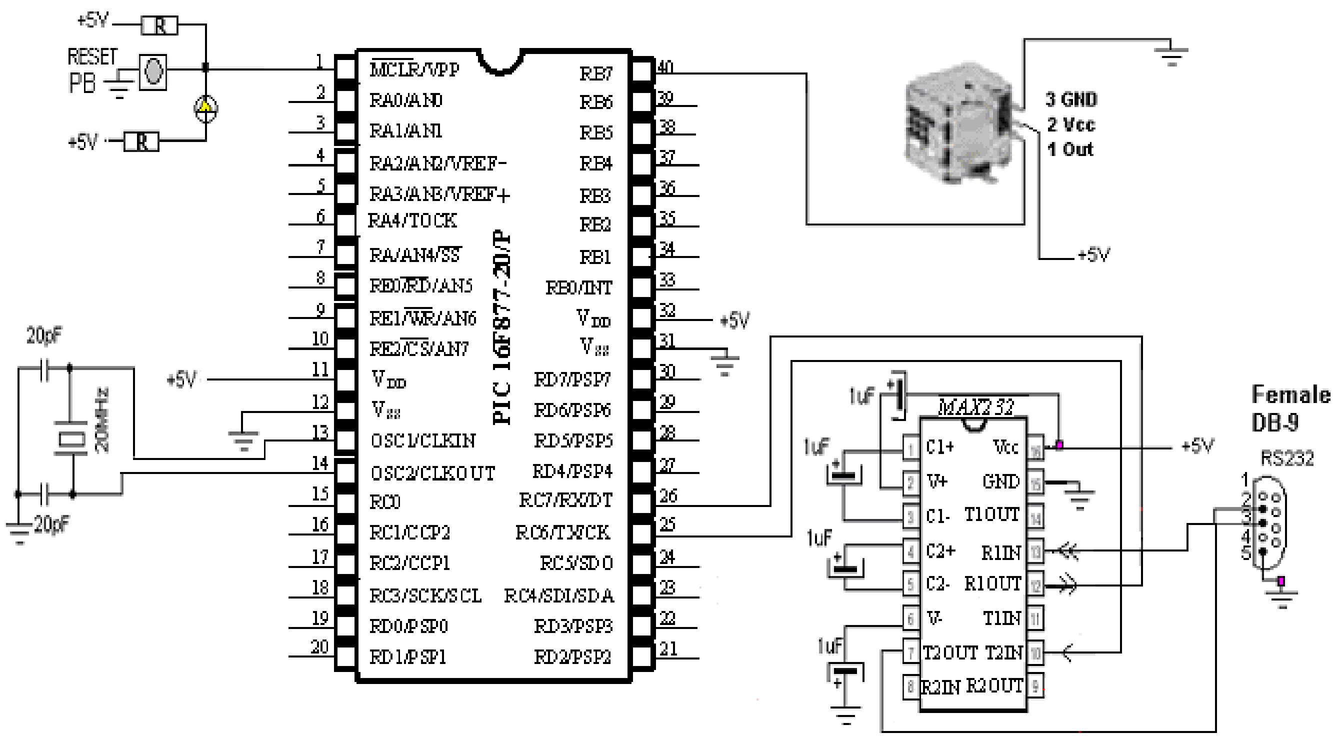


Fig. 31 Remote Control of LED

Since we already have the Sony IR program, the only thing we have to add is to decode the IR remote controller. Since the Address part is not important in this case, all our attention is to decode the content of the file register `COMreg`. Let's assume that `COMreg` now holds byte-long information of the current IR signal received. From the table for the list of messages sent by Sony remote controls in 12-bit protocol, we know that a key "4" would give 03h or 00000011b to `COMreg`.

Key Pressed	Content of `COMreg`	
	Hex	Binary
"0"	09	0 0 0 0 1 0 0 1
"1"	00	0 0 0 0 0 0 0 0
"2"	01	0 0 0 0 0 0 0 1
"3"	02	0 0 0 0 0 0 1 0
"4"	03	0 0 0 0 0 0 1 1
"5"	04	0 0 0 0 0 1 0 0
"6"	05	0 0 0 0 0 1 0 1
"7"	06	0 0 0 0 0 1 1 0
"8"	07	0 0 0 0 0 1 1 1
"9"	08	0 0 0 0 1 0 0 0

Now the question is how do we find the content of `COMreg`. An easy way is to use, as explained in Chapter 3, `andlw` instruction. For example, what would be the result of an AND operation?

```
      movf        COMReg,0
      andlw       B'11111111'
```

The result would be zero only for COMreg=00000000b. If any bit of `COMreg` is not zero, the result would not be zero. In other words, if the above operation results in zero result, the content of COMreg must be 00000000b, i.e., the key "1" from the remote controller. Then, if the above operation is not zeroed, then we can easily see that, if the below operation results in zero, the key "5" must be pressed.

```
      movf        COMReg,0
      andlw       B'11111011'
```

So the content check goes on until the last key is checked.

Next consideration is to make a 1-second time delay subroutine. Since we already have 10ms time delay from the previous example code, we make 1 s time delay by calling 10ms delay for 100 times.

```
;1 sec delay subroutine
;call 100 times of 10ms delay
Delay1s
            banksel     Kount1s
            movlw       0x64              ;100d
            movwf       Kount1s
R1s         call        Delay10ms
            decfsz      Kount1s
```

```
            goto        R1s
            return
```

Also, to simplify the code, it would be better convert the whole IR reading part into a subroutine. So we converted the previous Sony protocol reading part to `SonyIr` subroutine.

```
;==SONY IR Subroutine =====
SONYIR
      banksel     ADDRreg
      clrf        ADDRreg
      clrf        COMreg
      clrf        Pcount
      banksel     PORTB
      btfss       PORTB, IRX
      goto        SONYIR            ;if IRX is LOW, start again
;call delay10ms 20 times
      banksel     first
      movlw       0x14        ;20
      movwf       first
redo  call        Delay10ms   ;200mS delays
;check again for IRX
      btfss       PORTB, IRX
      goto        SONYIR
      decfsz      first
      goto        redo
;NOW ready to fresh read IR data
;
;Wait for START bit

jam
      banksel     PORTB
      btfsc       PORTB, IRX
      goto        jam
;now start bit is entered
      banksel     CMcount
      movlw       0x07
      movwf       CMcount           ;command has 7 bits
;wait for a separator
WAIT  btfss       PORTB, IRX  ;
      goto        WAIT        ;
CMNEXT
      clrf        Pcount
      bcf         STATUS, CARRY     ;Clear the Carry Bit
      rrf         COMreg            ;rotate to the right
;HIGH seperator IN
;then wait for LOW to decode 1 or 0
wait2 btfsc       PORTC, IRX  ;YES. Then check it AGain
      goto        wait2
DST   call        Delay120us  ;IRX is LOW NOW.  Delay 120 uS
;count (or measure) the LOW duration
;wait for separator
wait3 btfsc       PORTC, IRX
      goto        onezero
      incf        Pcount            ;if IRX still HIGH
      goto        DST

;
;"1" or "0" check
;
onezero
      btfsc       Pcount,0x03
      bsf         COMreg, MSB
      decfsz      CMcount
```

```
      goto        CMNEXT
      bcf         STATUS, CARRY
      rrf         COMreg

;Now ADDRESS READING
;NOTE that if you are not interested in the ADDRESS part
;Simple eliminate the line below, except the return instruction
;at the bottom
      movlw       0x05
      movwf       ADcount
; Check for START bit
ADNEXT
      clrf        Pcount
      bcf         STATUS, CARRY     ;Clear the Carry Bit
      rrf         ADDRreg           ;rotate to the right
cwait2
      btfsc       PORTB, IRX  ;YES. Then check it AGain
      goto        cwait2
cDST  call        Delay120us  ;IRX is HIGH NOW.  Delay 120 uS
;count (or measure) the HIGH duration
cwait3
      btfsc       PORTB, IRX
      goto        conezero
      incf        Pcount            ;if IRX still HIGH
      goto        cDST
;"1" or "0" check
;
conezero
      btfsc       Pcount,0x03
      bsf         ADDRreg, MSB
      decfsz      ADcount
      goto        ADNEXT

; end of stream
ewait4
      btfss PORTD, IRX
      goto  ewait4

      bcf   STATUS, CARRY
      rrf   ADDRreg
      rrf   ADDRreg
      rrf   ADDRreg
      return                  ;COMreg holds the Command Information
```

Since we already discussed about subroutines of 1 second time delay and Sony IR reading, the following code lists only main program part. For a complete code, insert all the subroutines just above `END` instruction line at the bottom.

```
;5IR-LED.asm
;
;This program is to:
;1. Read IR command from SONY Remote Controller
;2. Turn ON the LED for a given amount of seconds by
;   the number pressed by the button:
;    '1': 1 sec
;    '2': 2 sec  etc
;
;
; LED is connected to RD1
;
```

```
;IR-RX pin(IRX) is dedicated to RB7 port
;
;
;

        list P = 16F877

STATUS      EQU   0x03
CARRY       EQU   0x00        ;STATUS<0>
ZERO        EQU   0x02        ;Z flag STATUS<2>
TRISB       EQU   0x86
PORTB       EQU   0x06
TRISD       EQU   0x88
PORTD       EQU   0x08
IRX         EQU   0x07        ;RB7 for IR receiver
LED         EQU   0x01        ;RD1 for LED
MSB         EQU   0x07

;RAM

      CBLOCK      0x20
            TIMEBLOCK
            Kount120us
            Kount100us
            Kount1ms
            Kount10ms
            Kount200ms
            Kount1s
            Kount10s
            Kount1m
            first
            second
            third
            Bitcount    ;data bit count
            Kount       ;Delay count (number of instr cycles for delay)
            ADDRreg           ;IR ADDRESS
            COMreg            ;IR Command
            Pcount            ;HIGH duration count
            ADcount
            CMcount
      ENDC


;=========================================================
      org         0x0000            ;line 1
      goto        START             ;line 2 ($0000)
;======================================================
      org         0x05
START
      banksel     TRISB
; Port setting (1 for input and 0 for output)
      clrf        STATUS
      movlw       0x80
      movwf       TRISB

      banksel     TRISD
      movlw       0x00
      movwf       TRISD       ;All ports are outputs
AGAIN
      banksel     PORTD
      bcf         PORTD, LED  ;turn off LED
      call        SONYIR      ;read SONY IR REmote
      movf        COMreg,0    ;W has now the content of the command
      andlw       B'11111111'
```

```
      btfss       STATUS, ZERO       ; W = 00?  then 1
      goto        next
      goto        oneLED      ;turn on from 1 second
next  movf        COMreg,0
      andlw       B'11111101'        ;W= 2?  then 3 sec
      btfss       STATUS, ZERO
      goto        next2
      goto        threeLED
next2 movf        COMreg, 0
      andlw       B'11111100'        ;W=3?  then 4 sec
      btfss       STATUS, ZERO
      goto        next3
      goto        fourLED
next3 movf        COMreg,0
      andlw       B'11111011' ;W=4? then 5 sec
      btfss       STATUS,ZERO
      goto        next4
      goto        fiveLED
next4 movf        COMreg,0
      andlw       B'11111010' ;W=5? then 6 sec
      btfss       STATUS, ZERO
      goto        next5
      goto        sixLED
next5 movf        COMreg,0
      andlw       B'11111001' ;W=6? then 7 sec
      btfss       STATUS,ZERO
      goto        next6
      goto        sevenLED
next6 movf        COMreg,0
      andlw       B'11111000' ;W=7? then 8 sec
      btfss       STATUS, ZERO
      goto        next7
      goto        eightLED
next7 movf        COMreg, W
      andlw       B'11110111' ;W=8? then 9 sec
      btfss       STATUS, ZERO
      goto        next8
      goto        nineLED
next8 movf        COMreg,0
      andlw       B'11111110' ;W=1? then 2 sec
      btfss       STATUS, ZERO
      goto        next9
      goto        twoLED
next9 goto        AGAIN

; LED routine
nineLED
      bsf         PORTD, LED
      call        delay1s
eightLED
      bsf         PORTD, LED
      call        delay1s
sevenLED
      bsf         PORTD, LED
      call        delay1s
sixLED
      bsf         PORTD, LED
      call        delay1s
fiveLED
      bsf         PORTD,LED
      call        delay1s
fourLED
      bsf         PORTD,LED
      call        delay1s
```

```
threeLED
      bsf         PORTD, LED
      call        delay1s
twoLED
      bsf         PORTD,LED
      call        delay1s
oneLED
      bsf         PORTD,LED
      call        delay1s
      goto        AGAIN

;subroutines BELOW

;subroutines ABOVE

;END OF CODE
        END
```