Chapter 10. Synchronous Serial Communication and Keyboard Connection

1. Synchronous Communication

As its name implies, synchronous communication takes place between a transmitter and a receiver operating on synchronized clocks. In a synchronous system, the communication partners have a short conversation before data exchange begins. In this conversation, they align their clocks and agree upon the parameters of the data transfer, including the time interval between bits of data. Any data that falls outside these parameters will be assumed to be either in error or a placeholder used to maintain synchronization. (Synchronous lines must remain constantly active in order to maintain synchronization, thus the need for placeholders between valid data.) Once each side knows what to expect of the other, and knows how to indicate to the other whether what was expected was received, then communication of any length can commence.

Even though 16F877's USART module provides hardware enabled synchronous master/slave mode of serial communication, we opt to a software enabled approach. It's because the built-in serial port will be connected to a host PC for hex code download. Of course, that same port can be used for other serial device, it would be cumbersome to connect and disconnect a code. Moreover, we will connect another serial device, like a keyboard or mouse, in the example of this chapter, therefore, software approach will give us more freedom of adding additional serial device.

An application of this chapter is to connect a keyboard (eventually two keyboards) and one LCD to the 16F877 in order to let two persons of hearing or speaking disability communicate by typing and reading.   The keyboard we are going to connect is the most common type, IBM AT or PS/2 keyboard.   The keyboard communicates with PC in synchronous serial communication.

AT type keyboard has 5 pins while PS/2 type keyboard has 6 pins. As illustrated below, for both types of keyboard, there are total 4 signals: +5V power signal, ground, and a CLOCK line, and a DATA line.

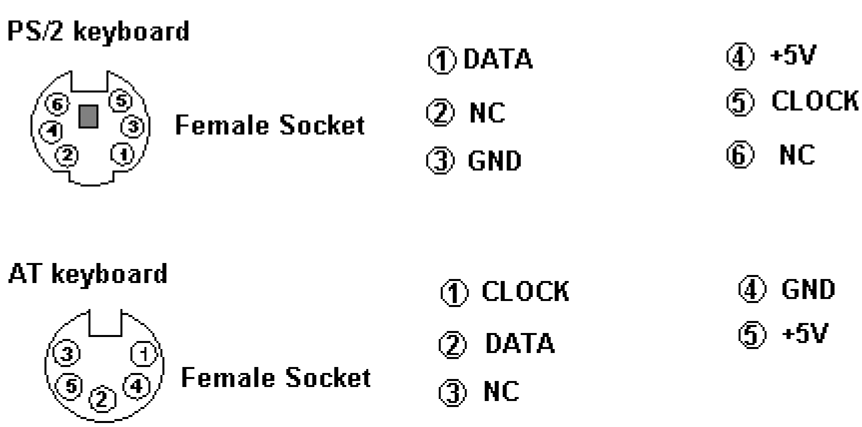


Fig. 75 PS/2 and AT type keyboard

2. IBM AT- or PS/2 – type Keyboard Protocol

The protocol between the keyboard and PC is the most important subject we have to understand in our example application.

The PS/2 mouse and keyboard implement a bidirectional synchronous serial protocol. The bus is "idle" when both lines are high (open-collector). This is the only state where the keyboard/mouse is allowed begin transmitting data. The host has ultimate control over the bus and may inhibit communication at any time by pulling the Clock line low.

The keyboard always generates the clock signal. This is done by a keyboard controlling microcontroller inside the keyboard.

The Data and Clock lines are both open-collector with pull-up resistors to +5V. An "open-collector" interface has two possible state: low, or high impedance. In the "low" state, a transistor pulls the line to ground level. In the "high impedance" state, the interface acts as an open circuit and doesn't drive the line low or high.

The first thing we have to know is how the keyboard controller chip send data to a host (PC or 17F877 in our case). As mentioned above there are two signals from the keyboard: CLOCK and DATA. DATA is sent only when synchronized with the CLOCK. When the keyboard is idle, without any key pressed, both CLOCK and DATA are remained pulled up High. When a key is pressed in the keyboard, both the CLOCK pulse and DATA pulse are transmitted from the keyboard. Through the DATA, strings of byte data are generated. The clock pulses are generated during the data transmission through the CLOCK line.

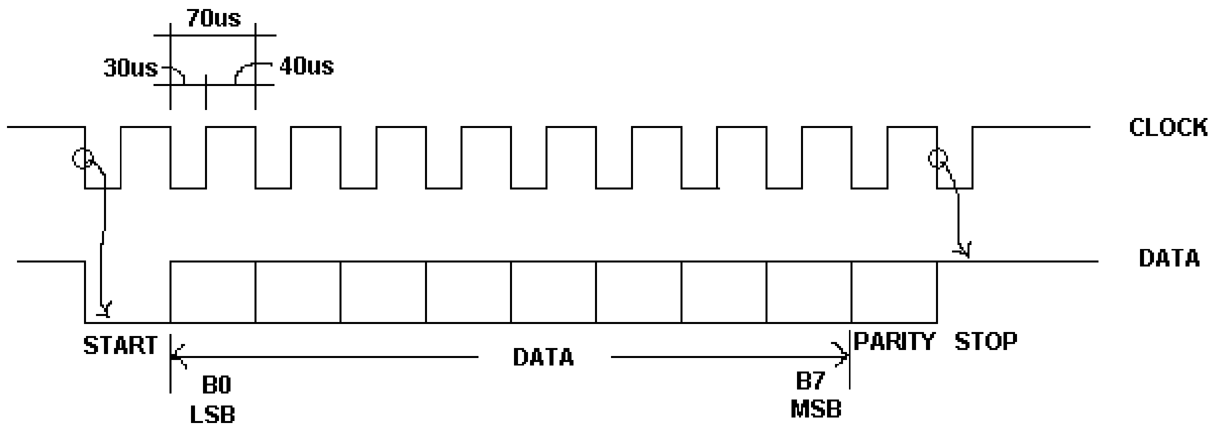
One thing we have to remember is that a single key stroke does not generate only a byte of data: it generates usually 3 bytes of data and, but other keys generate 5 bytes of data. The list of byte data generated by each individual key is called Keyboard Scan Codes. This discussion follows.

Let's continue our discussion on keyboard protocol. A byte data from the keyboard is sent in a frame consisting of 11 bits. The frame consists, in the following order, of:

1 Start bit (Low),
8-bit data (LSB first, as usual),
1 Odd Parity bit, and
1 Stop bit (High).

The width of the data bit is about 70μs. The frame is synchronized with 11 clock pulses of 70μs with about 40% duty cycle. Namely, the clock pulse's width is 70μs and it's High is about 307μs and its Low for 40μs. As indicated below, a host can sample (or monitor) each bit of the frame at the falling edge of the clock pulse. If we allow a short transition time of High-to-Low change, it would be safe to sample after around 5 μs of the High-to-Low transition of the clock.

To read the frame using 16F877, we need two I/O ports configured as inputs for CLOCK and DATA lines. First we monitor the CLOCK line for transition from High to Low. When it changes to Low, after 5 µs, we read the DATA line for each bit. Once a bit is read, now we back to the CLOCK monitoring. The CLOCK must go back to High and do the High-to-Low transition for the next bit reading. This process goes for all 11 bits of a frame. Since the 8-bit byte is sent LSB first, as soon as each bit of the byte is read, it must be rotate to the right by one to make a regular byte format: MSB to LSB.



Before we further proceed to the Keyboard Scan Codes, let's have a 16F877 connection with a standard AT or PS/2 keyboard. CLOCK line is connected to RB7 and DATA line to RB6 as illustrated below.

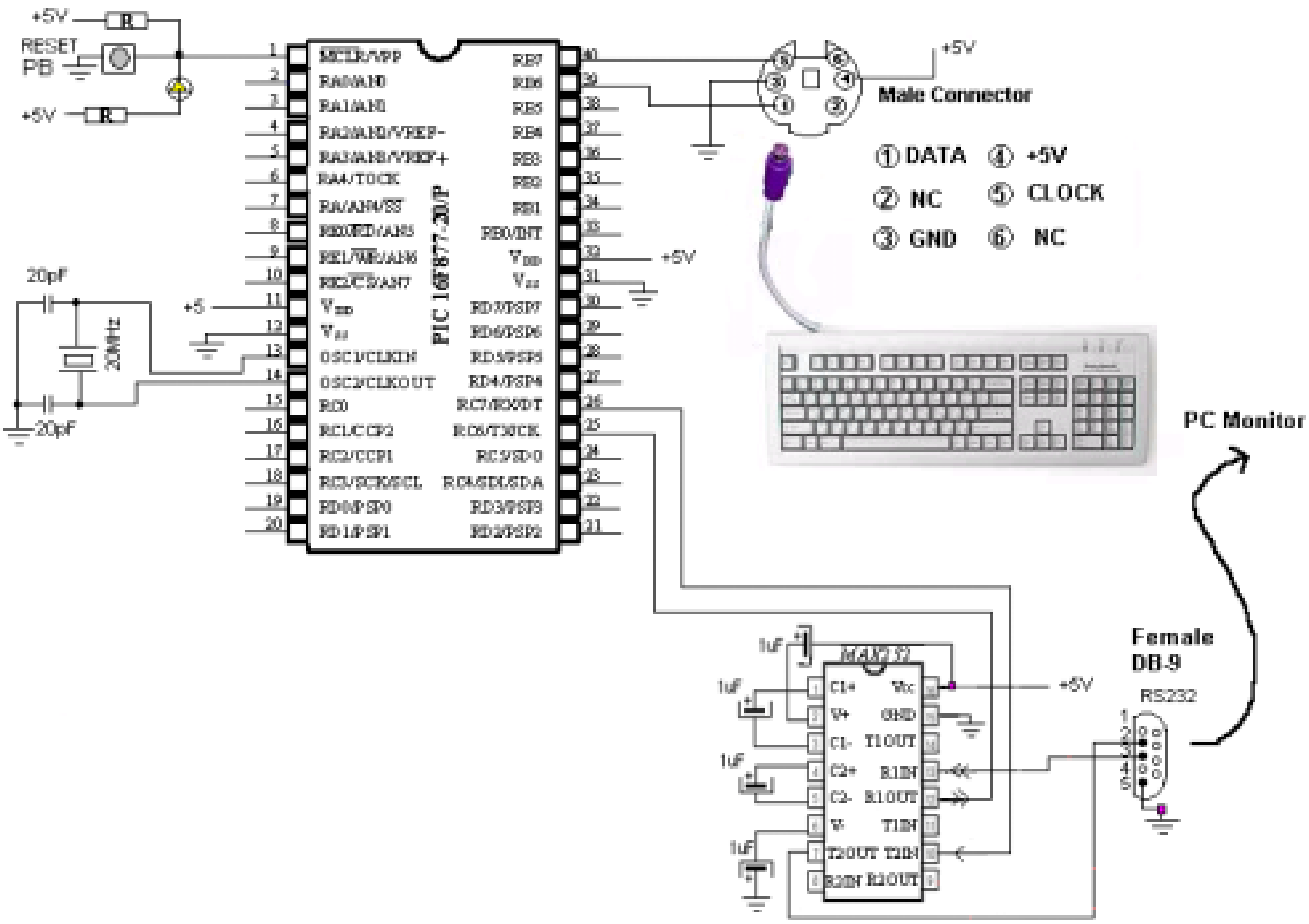


Fig. 76 PIC 16F877 connection to PS/2 Keyboard

Then, let's build a subroutine to read the 11-bit frame: the basic building block of keyboard reading. As explained above, reading each bit is based on the monitoring of the CLOCK line (RB7) of the High-to-Low transition. Since there are chances that the monitoring would be in the middle of the frame transmission, we may want to have a short High CLOCK before we allow to receive a frame. The subroutine, RX11bit, is listed below with ample amount of comments. In the subroutine, we keep the Parity bit for later use of transmission parity check. Also, an unknown transmission error is recorded when the last bit (STOP) is not High. The 8-bit data, after the subroutine, is stored in the file register of `DATAreg`.

```
;SUBROUTINE RX11bit ===========================================================
;RX Routine for 11-bit frame read
;1 Start
;8 Data (LSB first)
;1 Parity (Odd)
;1 Stop (HIGH)
;KSTAT Bit Info: KSTAT<0> : parity   KSTAT<2>:KBD Error
;
RX11bit
      clrf        DATAreg
      banksel     PORTB
;Let it have at least 200us CLOCK high period
      btfss       PORTB, CLOCK
      goto        RX11bit            ;if CLOCK is LOW, start again
      call        Delay100us         ;200uS delays
      call        DElay100us
;check again for CLOCK
      btfss       PORTB, CLOCK
      goto        RX11bit
;READY TO MONITOR CLOCK of H-to-L TRANSITION
Scheck
      btfsc       PORTB, CLOCK
      goto        Scheck
;CLOCK pulse is LOW
      call        delay5us           ;wait for 5us for data stabilization
      btfsc       PORTB, KDATA
      goto        KERROR             ;if START BIT is not Zero ERROR
;START Detected
;8-bit Data Check
      movlw       0x08
      movwf       Bitcount           ;Read 8 times for 8-bit Data
RXNEXT
      bcf         STATUS, CARRY      ;Clear the Carry Bit
      rrf         DATAreg            ;rotate to the right
CKHIGH
      btfss       PORTB, CLOCK       ;Wait for CLOCK to back to High
      goto        CKHIGH
CKLOW btfsc       PORTB, CLOCK       ;wait for CLOCK now to LOW
      goto        CKLOW
      call        delay5us           ;5us delay
      btfsc       PORTB, KDATA       ;DATA line reading. 0 or 1
      bsf         DATAreg, MSB       ;1? Then set the MSB
      decfsz      Bitcount
      goto        RXNEXT
;Check for Parity Bit
;Wait for CLOCK back to High
```

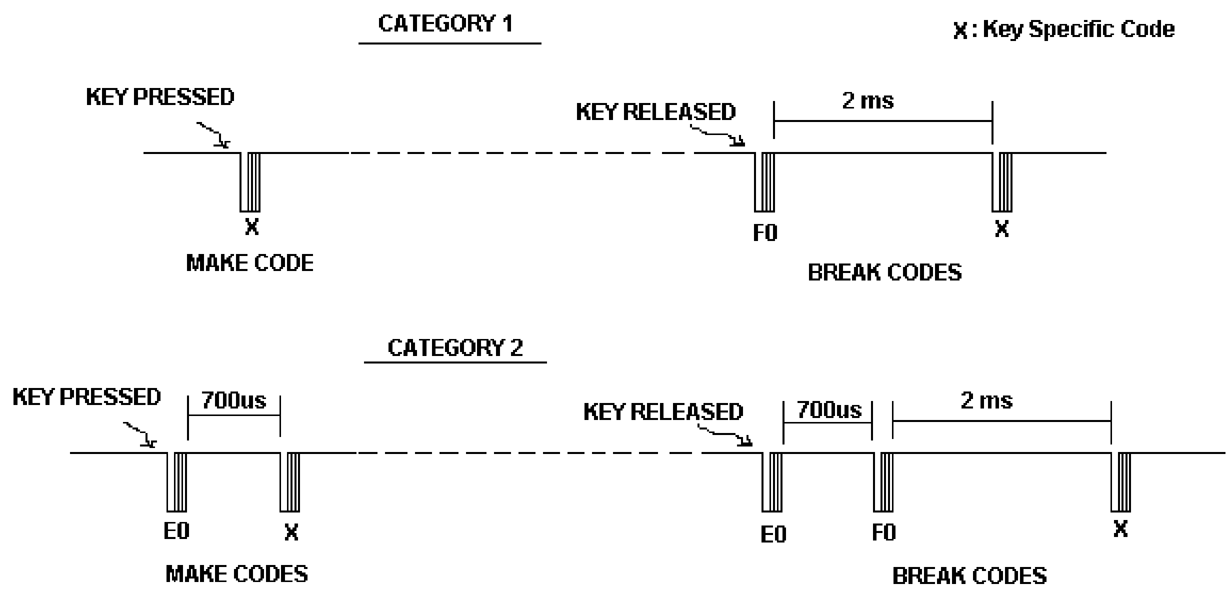
```
CKHIGH2
      btfss       PORTB, CLOCK      ;Wait for CLOCK to back to High
      goto        CKHIGH2
CKLOW2
      btfsc       PORTB, CLOCK      ;wait for CLOCK now to LOW
      goto        CKLOW2
      call        delay5us          ;5us delay
      btfsc       PORTB, KDATA      ;Parity Bit
      goto        OneP              ;Pbit=1
      bcf         Kstat,0x00        ;Pbit=0
      goto        Stopcheck
Onep  bsf         Kstat, 0x00       ;Parity bit=1 flag
Stopcheck
;wait for CLOCK back to High
CKHIGH3
      btfss       PORTB, CLOCK      ;Wait for CLOCK to back to High
      goto        CKHIGH3
CKLOW3
      btfsc       PORTB, CLOCK      ;wait for CLOCK now to LOW
      goto        CKLOW3
      call        delay5us          ;5us delay
      btfss       PORTB, KDATA      ;STOP bit
      goto        KERROR            ;if STOP=0 , ERROR
      return
KERROR
      bsf         KSTAT, 0x02       ;ERROR FLAG set
      return
```

As you see from the subroutine, the code is not far from the one we developed for asynchronous subroutine for data reception. Only difference is, here, we read the data bit by monitoring the clock transition and this is the essence of the synchronous serial communication. Since we built the basic building block of a frame read, now we have to look at the Scan Codes of keyboard to know what codes are transmitted when a key is pressed and released. When a key is pressed, the keyboard controller transmits one or two 1-byte "Make" code, and when the key is released it transmits two or three 1-byte "Break" codes.

Most of the keys in a keyboard (Category 1), thus, generate 1 Make code and 2 Break codes, when they are pressed and released. The first Break code is always F0h, and the second Break code is the same as the Make code. The Make code is separated from the Break codes only by how long the key is being pressed. If that key is kept on being pressed, only the 1 Make code would be continuously generated. However, the two Break codes are separated by about 2ms.

Break code is to know when a key is actually released especially for Shift keys. While a Shift key is pressed, an 'A' would generate 'A', however, when the Shift key is released, an 'A' would be interpreted as 'a', instead.

Some keys, like HOME, DEL, Page Up, Page Down, ←,↑,→, and ↓ (Category 2), generate two Make codes and three Break codes. For this group of keys, the first Make code is always E0. The first two Break codes are E0h and F0h, and the last break code is the same as the second Make code. Two Make codes are separated by about 700µs. The second Break code comes 700µs later after the first Break code. The last Break code arrives 2ms after the second Break code.



The following illustration shows the key specific codes for a keyboard. Category 2 keys are shaded. There are two keys which are not in either group: Print Screen/Sys Request and Pause keys. Print Screen key has the following codes, all in hexadecimal numbers (note that key released point is indicated by a vertical bar (|)): E0, 12, E0, 7C, |, E0, F0, 7C, E0, F0, 12. The Pause key has only Make codes: E1, 14, 77, E1, F0, 14. More details on Keyboard Scan Code can found in the Microsoft's Keyboard Scan Code Specification.

Fig. 77 Key Specific Codes for a Keyboard

Therefore, according to the Scan Code illustration, the 'A' without Shift key would produce: 1C | F0, 1C. On the other hand, if we press left Shift key and holding it until we type and release 'A', and then release the left Shift key, it would generate the following codes:
12 (Left Shift Make), 1C ('A' Make), | F0 (Break), 1C ('A' Break), | F0 (Break), 12(Left Shift Break).

If you keep press the Space bar for a few seconds followed by the release, it would generate the following codes:
29 (Space Make), 29 (Space Make), 29(Space Make),|, F0 (Break), 29 (Space Break).

On the other hand, if you press the Up Arrow (↑) key and release it, it would generate:
E0, 75(↑ Make), |, E0, F0 (Break), 75 (↑ Break).

3. First Code - Display of the Key Code Sequence

As we just examined the code generation by keys, there are two categories considered in displaying the codes of the keys on a PC monitor. If the first code from the keyboard (the Make code) is E0, it belongs to the Category2. In category 2, we have to read at least 5 Make and Break codes. When the category key is held pressed, the Make code should be read continuously until a Break is detected. If the first Make code is not E0, it belongs to the Category 1. In Category, we have to read 3 Make and Break codes in addition to the Make codes when the key is held pressed. In Category 1, there is one exception: Left and Right Shift Keys. Usually, the Shift Make code is followed by another Make code of the Category 1 key. Therefore, we must read 5 Make and Break codes in addition to the repeated Shift Make codes and/or the repeated Make codes of the Category 1 code.

Before discussing the main part of the code, for key reading and displaying the Make and Break codes of the key, we briefly study a few subroutines that we have not discussed so far. The 5 μs delay to read a bit after the High-to-Low transition of the CLOCK pulse, comes simply by having 10 `nop` (No Operation) instructions. Each instruction in 20 MHZ clock takes 0.2 μs.

```
;5us Delay Subroutine
delay5us
;need total 10 insturctions
      nop
      nop
      nop
      nop
      nop
      nop
      nop
      nop
      nop
      nop
      return
```

The code display subroutine, `Kdisplay`, is to change 1-byte hex number into 2-digit ASCII codes, and transmit the ASCII codes via the serial communication module of 16F877. This subroutine is almost identical to previous Monitor display routine. Only a slight variation and variable changes has been made caused by the keyboard reading accommodation.

```
;SUBROUTINE Kdisplay
Kdisplay
```

```
      banksel     Kreg
      movwf       Kreg
      movf        Kreg,0
      movwf       Ktemp
      swapf       Ktemp,0    ;SWAP upper and lower nibbles --->W
      andlw       0x0F        ;Mask off upper nibble

      call        HTOA        ;HEX---->ASCII
      movwf       DATA1       ;First Digit

      movf        Kreg,0
      andlw       0x0F       ;mask of upper nibble

      call        HTOA        ;HEX---->ASCII
      movwf       DATA2       ;Second Digit

;TX ROUTINE FOR Display

      movf        DATA1,0
      call        Txcall            ;Serial Transmission to PC monitor
      movf        DATA2,0
      call        Txcall            ;Serial Transmission to PC monitor

      return
```

One more thing we have to know about keyboard reading is that when the keyboard is powered up it automatically sends something called BAT (Basic Assurance Test) code which basically informs the host of the status of itself. Part of the BAT is an initial set-up for the selection of Brake only, Break/Make code, and Typematic. These pieces of information are generated about 1 second after the power on. So in the code we will examine, we may want to have at least 1 second delay after the power-on reset. The follow example code is to read a keyboard and then display the key using a PC monitor, by running the `hyperterminal` at the PC side. No subroutine is listed in the code.

```
;kbd1.asm
;
;MAIN FOCUS: DISPLAYING THE KEYS PRESSED on A PC MONITOR
;
;This program is to:
;1. Read At or PS/2 Type Keyboard
;2. Display the Make and Break codes, in HEX format, on PC monitor
;
;
;   Baud rate for this is set as 19200 for Monitor display
;
;Keyboard has bi-directional synchronous serial communication
;with clock of LOW period of 30us and High period of 40us
;Data comes with 11 bits:
;1 start bit of Low
;8 data bits (LSB first)
;1 Odd parity bit
;1 Stop bit of High
;
```

```
;Data reading is done when clock goes High-to-Low transition
;
;PIC can block the data from Keyboard by pulling down the clock Low >100ms
;
;
;To leave CLOCK line High-Z, set the port Input
;To make the CLOCK line LOw, set the port as ouput and write 0 to the port.

;
;Algorithm
;1. Check the CLOCK pin (RB7)
;2. When CLOCK goes to LOW, read the DATA line (RB6)
;3. START>8-bit Data>1 Parity>Stop
;4. Display the result
;

;Make /Break format
;(1) (X) | (F0) (X) =================> CAT1
;   L-SHIFT followed by a Key:  12 (X) |(F0) (X) |(F0) 12
;   R-SHIFT followed by a key:  59 (X) |(F0) (X) |(F0) 59
;(2) (E0)(X) | (E0)(F0) (X) ==========> CAT2
;
;
;Terminal set up: 8N1 19200
;
;Asynchronous mode
;
        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
RXMODE      EQU   0x90        ;RCSTA=10010000 : 8-bit, enable port, enable RX
BAUD        EQU   0x0F        ;0x0F (19200), 0x1F (9600)
CARRY       EQU   0x00
ZERO        EQU   0x02
MSB         EQU   0x07
CLOCK       EQU   0x07        ;from Keyboard
KDATA       EQU   0x06        ;from Keyboard

;
;RAM AREA

      CBLOCK      0x20
            KSTAT
            DATAreg
```

```
            MAKEreg

            BREAKreg1
            BREAKreg2
            BREAKreg

            Sbit
            Kount120us   ;Delay count (number of instr cycles for delay)
            Kount100us
            Kount1ms
            Kount10ms
            Kount100ms
            Kount1s
            Kount10s
            Kount1m
            Bitcount    ;data bit count
            Kount       ;Delay count (number of instr cycles for delay)
            DATAtemp    ;for ASCII conversion
            DATA1
            DATA2
            ASCIIreg
            Kreg
            Ktemp
      ENDC


;==========================================================
      org         0x0000
      GOTO        START
;=======================================================
      org         0x05
;start of the program from $0005
START

      banksel     TRISB
; 1100 0000

      movlw       B'11000000' ;RB7 for CLOCK and RB6 for DATA as inputs
      movwf       TRISB
      call        ASYNC_mode
      call        delay1s           ;Give Keyboard to send STATUS to the host
;KBD initial set-up by itself
;BAT(Basic Assurance Test) code
;typematic/make/break coding

BEGIN
      banksel     TXREG
      clrf        TXREG
      banksel     DATAreg           ;the RX11bit result here
      clrf        DATAreg
      clrf        Kreg
      clrf        ASCIIreg          ;ASCII equivalent here
      clrf        Ktemp

; CHECK IF THE CLOCK is HIGH at least for 100 mS
;to make sure it does not read in the middle of data/clock stream
```

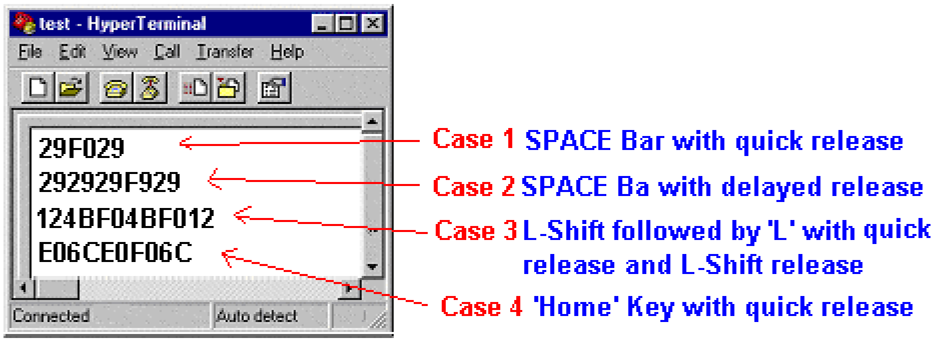
```
      banksel     PORTB
      btfss       PORTB, CLOCK
      goto        BEGIN       ;if CLOCK is LOW, start again
      call        Delay100ms  ;100mS delays

;check again for CLCOK
      btfss       PORTB, CLOCK
      goto        BEGIN
;READY FOR CLOCK PULSES
;KSTAT
;KSTAT<0>: Parity Bit Value
;KSTAT<2>: Kbd error
      clrf        KSTAT
KEYIN
;X reading
      call        RX11bit                 ;reading a frame

      clrf        STATUS
      movf        DATAreg,0               ;Break Code?
      xorlw       0xF0
      btfss       STATUS,ZERO
      goto        CAT
;BREAK, before MAKE code,  detected. Abort It. Resume It
      goto        BEGIN
;Category detection
CAT   clrf        STATUS
      movf        DATAreg,0
      xorlw       0xE0
      btfsc       STATUS,ZERO
      goto        CAT2
      clrf        STATUS
      movf        DATAreg,0         ;L-SHIFT Key Detection
      xorlw       0x12
      btfsc       STATUS,ZERO
      goto        LRSHIFT
      clrf        STATUS
      movf        DATAreg,0         ;R-SHIFT key detection
      xorlw       0x59
      btfsc       STATUS,ZERO
      goto        LRSHIFT

                              ;L Shift ===>12 | F0 12
                              ;R Shift ===>59 | F0 59
;CAT1 has the format of (X)|(F0)(X)
CAT1  movf        DATAreg,0
      movwf       MAKEreg
      call        Kdisplay          ;(X)

;(F0) detection
      call        RX11bit
      clrf        STATUS
      movf        DATAreg,0
      xorlw       0xF0
      btfss       STATUS,ZERO
;Key is not broken. Still pressed,
      goto        CAT1              ;IF No BREAK code, Key is still pressed.
;Key is broken
```

```
      movf        DATAreg,0
      movwf       BREAKreg1
      call        Kdisplay          ;F0
;Last (X) reading
      call        RX11bit
      movf        DATAreg,0
      movwf       BREAKreg2
      call        Kdisplay          ;(X)

      call        CRLF
      call        CRLF
      goto        BEGIN             ;Read next Key

;CAT2 format (E0)(X)|(E0)(F0)(X)
CAT2  movf        DATAreg,0
      call        Kdisplay          ;E0
      call        RX11bit
      movf        DATAreg,0
      movwf       MAKEreg
      call        Kdisplay          ;(X)
; KEY still PRESSED or BROKEN
      call        RX11bit
      movf        DATAreg,0
      clrf        STATUS
      xorlw       0xE0
      btfss       STATUS,ZERO
; NOT BROKEN
      goto        CAT2
;BROKEN
      movf        DATAreg,0
      call        Kdisplay          ;E0
      call        RX11bit
      movf        DATAreg,0
      call        Kdisplay          ;F0
      call        RX11bit
      movf        DATAreg,0
      movwf       BREAKreg
      call        Kdisplay          ;(X)

      call        CRLF
      call        CRLF
      goto        BEGIN             ;Read next Key

;L-SHIT and R-SHIFT has the form
;L-SHIFT and a character 12 X | F0 X |F0 12
;R-SHIFT and a character 59 X | F0 X |F0 59

LRSHIFT
      movf        DATAreg,0
      movwf       MAKEreg
      call        Kdisplay          ;12 or 59

;(F0) detection
      call        RX11bit
      clrf        STATUS
      movf        DATAreg,0
      xorlw       0xF0
```

```
      btfsc       STATUS,ZERO
      goto        BEGIN             ;IF no BREAK, key is not Broken yet


LRS   movf        DATAreg,0
      movwf       BREAKreg1
      call        Kdisplay          ;X

;(F0) detection
      call        RX11bit
      clrf        STATUS
      movf        DATAreg,0
      xorlw       0xF0
      btfss       STATUS,ZERO
      goto        LRS
      movf        DATAreg,0
      call        Kdisplay          ;F0
;Last (X) reading
      call        RX11bit
      movf        DATAreg,0
      movwf       BREAKreg2
      call        Kdisplay          ;(X)

      call        RX11bit
      movf        DATAreg,0
      call        Kdisplay          ;(F0)

      call        RX11bit
      movf        DATAreg,0
      call        Kdisplay          ;12 or 59
      call        CRLF
      call        CRLF
      goto        BEGIN             ;Read next key
;
; SUBROUTINES
;HERE
      END
;END OF THE CODE
```

Try several keys using a keyboard and see if you get the following or similar monitor display.

1. 'Space' Bar press and quick release.
2. 'Space' Bar press and delayed release.
3. 'Shift' and 'L' followed by L release and Shift Release.
4. 'Home' key.



4. Second Code - Display of Key Itself

The second version of the program is to display the key on the monitor, not the Make and Brake codes of the key. To do this, we have to closely examine the code of a key and a combination of keys.

Category 1 Keys (Keypad is ignored)

			Without **Shift** or **Caps Lock**		With **Shift** or **Caps Lock**	
	Key	Make/Break Code (hex)	Character to be Displayed	ASCII code	Character to be Displayed	ASCII code
	~ `	0E	` (Apostrophe)	60	~ (Tilde)	7E
	!1	16	1	31	!	21
	@2	1E	2	32	@	40
	#3	26	3	33	#	23
	$4	25	4	34	$	24
	%5	2E	5	35	%	25
	^6	36	6	36	^	5E
	&7	3D	7	37	&	26
	*8	3E	8	38	*	2A
	(9	46	9	39	(	28
	)0	45	0	30	)	29
	_-	4E	-	2D	_	5F
	+=	55	=	3D	+	2B
	BS	66	BS	08	BS	08
	A	1C	a	61	A	41
	B	32	b	62	B	42
	C	21	c	63	C	43
	D	23	d	64	D	44
	E	24	e	65	E	45
	F	2B	f	66	F	46
	G	34	g	67	G	47
	H	33	h	68	H	48
	I	43	i	69	I	49
	J	3B	j	6A	J	4A
	K	42	k	6B	K	4B
	L	4B	l	6C	L	4C
	M	3A	m	6D	M	4D
	N	31	n	6E	N	4E
	O	44	o	6F	O	4F
	P	4D	p	70	P	50

	Q	15	q	71	Q	51
	R	2D	r	72	R	52
	S	1B	s	73	S	53
	T	2C	t	74	T	54
	U	3C	u	75	U	55
	V	2A	v	76	V	56
	W	1D	w	77	W	57
	X	22	x	78	X	58
	Y	35	y	79	Y	59
	Z	1A	z	7A	Z	5A
	L-Shift	12				
	R-Shift	59				
	Enter	5A	CR	0D	CR	0D
	Space	29	Space	20	Space	20
	Cap	58				
	{ [	54	[	5B	{	7B
	}]	5B	]	5D	}	7D
	| \	5D	\	5C	|	7C
	: ;	4C	;	3B	:	3A
	" '	52	'	27	"	22
	< ,	41	,	2C	<	3C
	> .	49	.	2E	>	3E
	? /	4A	/	2F	?	3F

As we see from the Category 1 key table, relating its Make/Brake code and matching ASCII code of the corresponding character, it is apparent that the Make/Break codes of alphanumeric keys doe not match with ASCII codes of the characters of the keys. When we type a key 'A' with or without pressing a Shift key, the Make/Break code the host would get is 1Ch. This code must be changed to either 61h (ASCII cod for 'a') without Shift or 41h(ASCII code for 'A') with Shift pressed. Since there is no pattern to easily convert a Make/Break code to ASCII code, we have to reply on a look-up table approach.

The approach here is to use a Make/Break code as the address where its ASCII equivalent code is stored. In other words, the Make/Break code will direct where to get the ASCII equivalent code. This sounds very simple with one minor constraint. A key in the keyboard generate the same Make/Break code, however, depending upon the Shift key or Cap Lock key, it has two ASCII equivalent codes. Therefore, we have to have two look-up tables, one without Shift (or Cap) key, and the other with Shift (or Cap) key.

The following table summarizes the two look-up tables, rearranged in the rising order of the Make/Break codes, the above table of Category 1 key.

Make/Break Code (hex)	Without **Shift** or **Caps Lock**		With **Shift** or **Caps Lock**	
	Character to be Displayed	ASCII code	Character to be Displayed	ASCII code
0E	`	60	~	7E
15	q	71	Q	51
16	1	31	!	21
1A	z	7A	Z	5A

1B	s	73	S	53
1C	a	61	A	41
1D	w	77	W	57
1E	2	32	@	40
21	c	63	C	43
22	x	78	X	58
23	d	64	D	44
24	e	65	E	45
25	4	34	$	24
26	3	33	#	23
29	Space	20	Space	20
2A	v	76	V	56
2B	f	66	F	46
2C	t	74	T	54
2D	r	72	R	52
2E	5	35	%	25
31	n	6E	N	4E
32	b	62	B	42
33	h	68	H	48
34	g	67	G	47
35	y	79	Y	59
36	6	36	^	5E
3A	m	6D	M	4D
3B	j	6A	J	4A
3C	u	75	U	55
3D	7	37	&	26
3E	8	38	*	2A
41	,	2C	<	3C
42	k	6B	K	4B
43	i	69	I	49
44	o	6F	O	4F
45	0	30	)	29
46	9	39	(	28
49	.	2E	>	3E
4A	/	2F	?	3F
4B	l	6C	L	4C
4C	;	3B	:	3A
4D	p	70	P	50
4E	-	2D	_	5F
52	'	27	"	22
54	[	5B	{	7B
55	=	3D	+	2B
5A	CR	0D	CR	0D
5B	]	5D	}	7D
5D	\	5C	|	7C
66	BS	08	BS	0B

Then, how do we generate a table or two tables in 16F877 assembly language programming environment? The easiest way of table building is to change the program counter (PC). As we all know, PC indicates the next address to fetch a program code and execute. PC in 16F877 is a 13-bit register which has address access range of 2^{13}=8K words. The lower 8 bits of the PC is can be controlled (i.e., read from and written to) by PCL (PC Lower Byte) register. The upper 5 bits are not directly accessed by PCH (PC High Byte), instead it is controlled by PCLATH (PC Latch) register.

Table formation is utilized by the PCL in a subroutine, along with an instruction (`retlw k`, return with k in W register) or a Microchip Assembler (MPASM) directive (`DT`, Define Table). In other words, a table formation is a subroutine building with PCL, and lines of `retlw` and/or `DT`.

Let's have a simple program to illustrate how to form a table. In the previous example of serial communication, we have had a key typed in the keyboard echoed on a PC monitor. Now we want to change the program slightly so that the program receives only numbers (0 to 9) from the keyboard and echoes corresponding characters determined by the following table:

Key	0	1	2	3	4	5	6	7	8	9
Echo	A	B	C	D	E	F	G	H	I	J

The pseudo-code of the revised program would go like this:
(a) Receive a number from keyboard by calling `RXPOLL` subroutine
(b) The received key is converted from ASCII (30h for '0', for example) to a hex number (00h for '0', for example).
(c) Check the table and find the corresponding character ('A' for 00h for example).
(d) Transmit the matched character to PC monitor by calling `TXPOLL` subroutine.

The following code lists the table subroutine, `Keytable`:

```
;
Keytable
      addwf       PCL
      retlw       'A'    ;PC+0
      retlw       'B'    ;PC+1
      retlw       'C'    ;PC+2
      retlw       'D'
      retlw       'E'
      retlw       'F'
      retlw       'G'
      retlw       'H'
      retlw       'I'
      retlw       'J'          ;PC+9
```

When this subroutine is called, the return address is stored in the Stack, and the content of PC is the starting address of this subroutine in other words, the PC is the address of the first instruction line is the subroutine:
`addwf PCL`.

That's why when a subroutine is called, it's executed from the first line of the subroutine. When the `addwf PCL` is executed (sum of the content of W and PCL), the PC changes by the amount of W content. If the W is 0, then there is no change in PC so the next line
`retlw 0x41`
would be executed. (Remember PC always directs the next code to fetch and execute.)

The instruction
`retlw k`

is combination of two instructions:

```
movlw  k
return
```

Therefore, `retlw  0x41` returns to the next line after the caller with 41h in the W register. If we place this content in the `TXREG` of USART module and call `TXPOLL`, the character 'A' would be displayed on the monitor.

What happens if number 5 is pressed in the keyboard? The hex-converted number 05h would be placed in W register after `RXPOLL` subroutine, then when the `Keytable` is called, PCL is added by 5, therefore, the sixth line after the PCL instruction would be executed, which results in the display of 'F' on the monitor. The following code lists the main part and the table of the whole program.

```
;=========================================================
      org         0x0000
      GOTO        START
;======================================================
      org         0x05
;start of the program from $0005
START
      call        ASYNC_mode        ;initialization of USART module

BEGIN
      banksel     TXREG
      clrf        TXREG
      clrf        RCREG
      clrf        Ktemp
AGAIN
      call        RXPOLL      ;read a key
      movwf       Ktemp
      movlw       0x30
      subwf       Ktemp,0     ;Make it into a hex number  (f-W -->d)
      call        Keytable    ;Call Keytable match
      call        TXPOLL      ;matched character
      call        CRLF        ;CR and LF
      goto        AGAIN       ;Repeat

Keytable
      addwf       PCL
      retlw       'A'   ;PC+0
      retlw       'B'   ;PC+1
      retlw       'C'   ;PC+2
      retlw       'D'
      retlw       'E'
      retlw       'F'
      retlw       'G'
```

```
      retlw       'H'
      retlw       'I'
      retlw       'J'
```

The keytable can also be formed by using an assembler directive, DT. DT actually is a multiple of `retlw` instructions:

```
Keytable
      addwf       PCL
      DT          "ABCDEFGHIJ"      ;PC+0, 1, 2, 3, 4, 5, 6, 7, 8, and 9
```

There is one important matter to remember when we use the PCL writing instruction, `addwf PCL`. When a PCL is written, the lower 5 bits of PCLATH register is also written to PCH. Since PC is PCH (PC[12:8])concatenated by PCL (PC[7:0]), if both the location of the caller line and that of the keytable subroutine are located inside the 00h – FFh boundary, there is no problem, since PCH for both parts is the same. Let's discuss this using the List file of the above program.

```
MPASM 02.61 Released         19TABLE1.ASM   5-20-2004  14:58:05
LOC  OBJECT CODE     LINE SOURCE TEXT

                      00036
;=========================================================
0000                  00037         ORG     0X0000
0000   2805           00038         GOTO    START
                      00039
;======================================================
0005                  00040         ORG     0X05
                      00041 ;start of the program from $0005
0005                  00042 START
0005   201E           00043         CALL    ASYNC_MODE
                      00044
                      00045
0006                  00046 BEGIN
0006   1283 1303      00047         BANKSEL TXREG
0008   0199           00048         CLRF    TXREG
0009   019A           00049         CLRF    RCREG
000A   01A0           00050         CLRF    KTEMP
000B                  00051 AGAIN
000B   2033           00052         CALL    RXPOLL
000C   00A0           00053         MOVWF   KTEMP
000D   3030           00054         MOVLW   0X30
000E   0220           00055         SUBWF   KTEMP,0
000F   2013           00056         CALL    KEYTABLE
0010   202B           00057         CALL    TXPOLL
0011   203B           00058         CALL    CRLF
0012   280B           00059         GOTO    AGAIN
                      00060
                      00061
0013                  00062 KEYTABLE
0013   0782           00063         ADDWF   PCL
0014   3441           00064         RETLW   'A'     ;PC+0
0015   3442           00065         RETLW   'B'     ;PC+1
0016   3443           00066         RETLW   'C'     ;PC+2
```

```
0017   3444          00067         RETLW   'D'
0018   3445          00068         RETLW   'E'
0019   3446          00069         RETLW   'F'
001A   3447          00070         RETLW   'G'
001B   3448          00071         RETLW   'H'
001C   3449          00072         RETLW   'I'
001D   344A          00073         RETLW   'J'
```

The first column is the program memory address and the second and the thirds are for the op-codes of the instruction. As we see from the List file, the caller (which calls the `Keytable` subroutine) is located at 000F. In other words, up until the `Keytable` is called, the PC is, in binary number, 0 0000 0000 1111 (or 000Fh). In other words, the upper byte of PC, PCH, is 00000 (or 00h). This would be the content of the lower 5 bits of PCLATH when the `Keytable` is called. When the PCL writing is done, 00h, the content of PCLATH will be filled to PCH, making the upper byte of PC zero. This, however, does not cause any trouble, because the `Keytable` subroutine, from 0013h to 001D, is with only lower byte portion of PC, PCL. In other words, the PC access to the subroutine instructions are inside the range with PCH=00h.

If the caller and the `keytable` subroutine is apart more than FFh each other, and there is uncertainty about this, we have to see the List file and decide what action should be made to avoid possible PC related problem.

For example, consider that the caller is located in 000Fh (with the current PC) and the `keytable` subroutine is located in 0113h instead. When the subroutine is called, the content of PC is 0000Fh. Therefore the PCH is 00000b. This binary value of 00000h would be written to the PCH portion of the PC when PCL writing is performed. Therefore the final value of PC would be 0013h, instead of the desired value of 0113h. Therefore, we have to have the following instruction just above the PCL writing instruction:

```
bsf  PCLATH, 0x00
```

Check with Microchip's 16F877 manual for detailed description of PC, PCH, PCL, PCLATH, and PCL writing with relation to the four different pages of 8K word program memory structure.

Now, we are ready to explore the second version of the keyboard connection to a PC monitor. The second example is to display the characters of the keys itself not the Make/Break codes of the keys. We use the Make/Break code table for keytable formulation. Since we have to consider the three following conditions; (a) when Shift key is not pressed, (b)when Shift key is pressed, and (c) when Caps Lock key is pressed. When Caps Lock is pressed we change only alphabets to upper cases and keep all other keys as if no Shift key is pressed.

The first table is for the keys without a Shift key, `NoShiftKeyTable`. According to the Make/Break code, the lowest hex number of the code is 0E which, without a Shift Key, is for '`' (Apostrophe), with its ASCII code 60. Then there are gaps until we have the next Make/Break code, which are 15h for 'q' and 16h for '1'. The following code shows the table for no Shift key. We will start the table 0100h. Since we assume that the caller is located before 0100h address, before the call is made, the PCH would be 00h. At the first line of the subroutine, we configure the PCH of PC by PCLATH instruction.

The second table is for the keys with a Shfit key, `ShiftKeyTable`, located just after the first table. This table can be similarly formed.

The third table accommodates Caps Lock key, `CapKeyTable`, which starts from 0200h. This address (also PC) requires additional PCL configuration. Details are commented in the code section.

The pseudo-code of the main section of the program goes like this. It's rather complicated. So we have to fully understand the Make/Break code pattern when a key is pressed and released.

(1)BEGIN: read the first frame and get the 8-bit key data
(2)Check if the key is 12h (Left Shift) or 59h (Right Shift). If it is, jump to LRSHIFT
(3) Check if the key is 58h(Caps Lock). If it is, jump to CAPS
(4)NOSHIFT:
    (a) Call NoShiftKeyTable and display what W register holds
    (b) read the next frame. If the key is F0h, then read one more frame then go to BEGIN
    (c) If the key is not F0h, go to NOSHIFT
(5) LRSHIFT:
    (a)Read the next frame. If the key is F0h, then read one more frame then go to BEGIN
    (b)If the key is 12 or 59h (Shift key is not broken yet), go to LRSHIFT
    (c) If the key is neither 12h nor 59h, call ShiftKeyTable and display what W register holds.
    (d)read the next frame. If the key is not F0h, call ShiftKeyTable and display
    (e) if the key is F0h, go to LRS
    (f)LRS: read next frame, if the key is 12h, go to BEGIN. If the key is not12h, go to LRSHIFT.
(6)CAPS:
    (a)Read the next two frames.
    (b) CAPNEXT: Read the next frame. If the key is 58h, then read two more frames, then go to BEGIN
    (b)If the key is not 58h, call CapKeyTable and display what W register holds.
    (c)Read the next frame. If the key is F0h, read the next frame and go to CAPNEXT.
    (d) If the key is 58h, then read two more frames, then go to BEGIN.

The next listing shows a complete code for displaying characters, numbers, and other symbols while accommodating Shift (Left and Right) and Caps Lock keys. Follow each line and comment closely to better understand the program.

```
;kbd3.asm
;
;SHIFT and CAPS LOCK are featured
;
;This program is to:
;1. Read At or PS/2 Type Keyboard
;2. Display them, as characters, on PC monitor
;
;3. Note that all Category 2 keys (E0 keys) are ignored
```

```
;
;   Baud rate for this is set as 19200 for Monitor display
;
;
;Terminal set up: 8N1 19200
;
;Asynchronous mode
;


        list P = 16F877
PCL         EQU   0x02        ;For Key Table Calling (Lower PC)
PCLATH      EQU   0x0A        ;For upper part of PC
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
RXMODE      EQU   0x90        ;RCSTA=10010000 : 8-bit, enable port, enable RX
BAUD        EQU   0x0F        ;0x0F (19200), 0x1F (9600)
CARRY       EQU   0x00
ZERO        EQU   0x02
MSB         EQU   0x07
CLOCK       EQU   0x07        ;from Keyboard
KDATA       EQU   0x06        ;from Keyboard

;
;RAM AREA

      CBLOCK      0x20
            TXtemp
            KSTAT
            DATAreg
            Kount100us
            Kount10ms
            Kount1s
            Bitcount    ;data bit count
      ENDC


;==========================================================
      org         0x0000
      GOTO        START
;=======================================================
      org         0x05
;start of the program from $0005
START
```

```
      banksel     TRISB
; 1100 0000

      movlw       B'11000000' ;RB7 for CLOCK and RB6 for DATA as inputs
      movwf       TRISB
      call        ASYNC_mode
      call        delay1s           ;Give Keyboard to send STATUS to the host
;KBD initial set-up by itself
;BAT(Basic Assurance Test) code
;typematic/make/break coding

BEGIN
      banksel     TXREG
      clrf        TXREG
      banksel     DATAreg
      clrf        DATAreg

; CHECK IF THE CLOCK is HIGH at least for 10mS

      banksel     PORTB
      btfss       PORTB, CLOCK
      goto        BEGIN       ;if CLOCK is LOW, start again
      call        Delay10ms   ;10mS delays
;This short delay speeds up the response
;check again for CLCOK
      btfss       PORTB, CLOCK
      goto        BEGIN
;READY FOR CLOCK PULSES
;KSTAT
;KSTAT<0>: Parity Bit Value
;KSTAT<2>: Kbd error
      clrf        KSTAT
KEYIN
;X reading
      call        RX11bit                 ;
      clrf        STATUS
      movf        DATAreg,0   ;Break Code?
      xorlw       0xF0
      btfss       STATUS,ZERO
      goto        CAT
;BREAK is detected. Abort It. Resume It
      goto        BEGIN
;Category detection (SHIFT or CAPS LOCK)
CAT   clrf        STATUS
      movf        DATAreg,0
      xorlw       0xE0
      btfsc       STATUS,ZERO
      goto        Begin       ;E0 keys (CAT2) are ignored
      clrf        STATUS
      movf        DATAreg,0
      xorlw       0x12
      btfsc       STATUS,ZERO
      goto        LRSHIFT
      clrf        STATUS
      movf        DATAreg,0
      xorlw       0x59
```

```
      btfsc       STATUS,ZERO
      goto        LRSHIFT
      clrf        STATUS
      movf        DATAreg,0
      xorlw       0x58        ;CAPS LOCK
      btfsc       STATUS,ZERO
      goto        CAPS

                              ;L Shift ===>12 | F0 12
                              ;R Shift ===>59 | F0 59
;CAT1 without Shift or Caps Lock Key
CAT1  movf        DATAreg,0
      call        NoShiftKeyTable         ;(X) display
      call        TXCALL
;(F0) detection
      call        RX11bit
      clrf        STATUS
      movf        DATAreg,0
      xorlw       0xF0
      btfss       STATUS,ZERO
;Key is not broken. Still pressed,
      goto        CAT1
;Key is broken
;Last (X) reading
      call        RX11bit                 ;(X) after F0

      goto        BEGIN

;L-SHIT and R-SHIFT has the form
;L-SHIFT and a character 12 X | F0 X |F0 12
;R-SHIFT and a character 59 X | F0 X |F0 59

LRSHIFT                             ;12 or 59 entered

;(F0) detection
      call        RX11bit
      clrf        STATUS
      movf        DATAreg,0
      xorlw       0xF0
      btfsc       STATUS,ZERO
      goto        BEGIN

                              ;X

      clrf        STATUS            ;if (12) do not display
      movf        DATAreg,0
      xorlw       0x12
      btfsc       STATUS, ZERO
      goto        LRSHIFT

      clrf        STATUS            ;if (59) do not display
      movf        DATAreg,0
      xorlw       0x59
      btfsc       STATUS, ZERO
      goto        LRSHIFT

                        ;a Key (X) is entered
```

```
      movf        DATAreg,0
      call        ShiftKeyTable
      call        TXCALL

;(F0) detection
      call        RX11bit
      clrf        STATUS
      movf        DATAreg,0
      xorlw       0xF0
      btfss       STATUS,ZERO
      goto        LRSHIFT
;Last (X) reading
      call        RX11bit
      movf        DATAreg,0
      clrf        STATUS            ;check if (X) or (12) entered after F0
      xorlw       0x12
      btfsc       STATUS,ZERO
      goto        BEGIN
      goto        LRSHIFT
;
CAPS                          ;caps lock (58) entered

;(F0) detection
      call  RX11bit                 ;this must be F0
      call  RX11bit                 ;this must be (58) again

CAPnext
      call        RX11bit                 ;Check if (58) or other
      clrf        STATUS
      movf        DATAreg,0
      xorlw       0x58
      btfss       STATUS,ZERO
      goto        CAPtwo                  ;End of CAP session
      call        RX11bit                 ;F0
      call        RX11bit                 ;(58)
      goto        BEGIN
                              ;a Key (X) is entered
CAPtwo
      movf        DATAreg,0
      call        CAPKeyTable
      call        TXCALL

;(F0) detection
      call        RX11bit                 ;
      clrf        STATUS
      movf        DATAreg,0
      xorlw       0xF0
      btfss       STATUS,ZERO
      goto        CAPtwo


;Last (X) reading             ;F0 is read
      call        RX11bit                 ;(X) again and ignore
      goto        CAPnext
```

```
;SUBROUTINE RX11bit ========================================================
;RX Routine for 11-bit read
;1 Start
;8 Data (LSB first)
;1 Parity (Odd)
;1 Stop (HIGH)
;KSTAT Bit Info
; KSTAT<0> : parity
; KSTAT<2>:kBD Error
RX11bit
      clrf        DATAreg
      banksel     PORTB
;Let it have at least 500us CLOCK high period
      btfss       PORTB, CLOCK
      goto        RX11bit           ;if CLOCK is LOW, start again
      call        Delay100us  ;200uS delays
      call        DElay100us
;check again for CLCOK
      btfss       PORTB, CLOCK
      goto        RX11bit
;Clock Check
Scheck
      btfsc       PORTB,CLOCK
      goto        Scheck
      call        delay5us    ;wait for 5us     for data stabilization
      btfsc       PORTB, KDATA
      goto        KERROR                  ;if START BIT is not Zero ERROR
;START Detected
;8-bit Data Check
      movlw       0x08
      movwf       Bitcount    ;8 data bits
RXNEXT
      bcf         STATUS, CARRY     ;Clear the Carry Bit
      rrf         DATAreg           ;rotate to the right
CKHIGH
      btfss       PORTB, CLOCK      ;Wait for CLOCK to back to High
      goto        CKHIGH
CKLOW btfsc       PORTB, CLOCK      ;wait for CLOCK now to LOW
      goto        CKLOW
      call        delay5us    ;5us delay
      btfsc       PORTB, KDATA      ;0 or 1
      bsf         DATAreg, MSB      ;1? Then set the MSB
      decfsz      Bitcount
      goto        RXNEXT
;Check for Parity Bit
;Wait for CLOCK back to High
CKHIGH2
      btfss       PORTB, CLOCK      ;Wait for CLOCK to back to High
      goto        CKHIGH2
CKLOW2
      btfsc       PORTB, CLOCK      ;wait for CLOCK now to LOW
      goto        CKLOW2
      call        delay5us    ;5us delay
      btfsc       PORTB, KDATA      ;Parity Bit
      goto        OneP        ;Pbit=1
      bcf         Kstat,0x00  ;Pbit=0
      goto        Stopcheck
```

```
Onep  bsf         Kstat, 0x00 ;Pbit=1
Stopcheck
;wait for CLOCK back to High
CKHIGH3
      btfss       PORTB, CLOCK      ;Wait for CLOCK to back to High
      goto        CKHIGH3
CKLOW3
      btfsc       PORTB, CLOCK      ;wait for CLOCK now to LOW
      goto        CKLOW3
      call        delay5us    ;5us delay
      btfss       PORTB, KDATA      ;STOP bit
      goto        KERROR            ;if STOP=0 , ERROR
      return
KERROR
      bsf         KSTAT, 0x02
      return
;=========================================================
;RX TX Initialization with Asyc 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
;slight change so that CR make CR and LF together
      banksel     TXtemp
      movwf       TXtemp
      clrf        STATUS
      movf        TXtemp,0
      xorlw       0x0D
      btfsc       STATUS,ZERO
      goto        CRNLF
Keymain
      banksel     PIR1
      btfss       PIR1, TXIF  ; Check if TX buffer is empty
      goto        Keymain
      banksel     TXREG
      movf        Txtemp,0
      movwf       TXREG       ; Place the character to TX buffer
      return
CRNLF call        CRLF
      return
;===
CRLF
      banksel     PIR1
      btfss       PIR1, TXIF
      goto        CRLF
      banksel     TXREG
      movlw       H'0d'       ;CR
```

```
      movwf       TXREG
LFkey
      banksel     PIR1
      btfss       PIR1, TXIF
      goto        LFkey
      banksel     TXREG
      movlw       H'0A'       ;LF
      movwf       TXREG
      return
;=================================================
delay5us
;need total 10 insturctions
      nop
      nop
      nop
      nop
      nop
      nop
      nop
      nop
      nop
      nop
      return
;==========================================

Delay100us
      banksel     Kount100us
      movlw       H'A4'
      movwf       Kount100us
R100us
      decfsz      Kount100us
      goto        R100us
      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       0x6A
      movwf       Kount1s
R1s   call        Delay10ms
      decfsz      Kount1s
      goto        R1s
      return
```

```
;
;KEYTABLE STARTS HERE
;======================================================
      org   0x0100                  ;So that all the table
                                    ;range has the same bit=1
                                    ;for the bit8 of PC
;Without Shift (or CAPs Lock) Key Table
NoshiftKeyTable

      bsf         PCLATH, 0x00
      bcf         PCLATH,0x01
      addwf       PCL
;Note that writing to PCL also brings the content of lower 5 bits of PCLATH
;to PC.
;In this code, this Table starts from 0100h
;While the main part of program which calls this Table is somewhere in
;0045h.
;for 0045, the PCLATH part is 00
;Therefore we have to manually set the PCLATH part so that it
;can point inside this table
      retlw       0           ;PC+0  (return 0 means display nothing)
      retlw       0           ;PC+1
      retlw       0           ;+2
      retlw       0
      retlw       0
      retlw       0
      retlw       0
      retlw       0
      retlw       0
      retlw       0
      retlw       0
      retlw       0
      retlw       0
      retlw       0           ;+0D
      retlw       0x60        ;+0E MAKE/BREAK= 0E ---->ASCII = 0x60
Apostrophe
      retlw       0           ;+0F
      retlw       0
      retlw       0
      retlw       0
      retlw       0           ;+13
      retlw       0           ;+14
      DT          "q1"        ;+15, 16
      retlw       0           ;+17
      retlw       0
      retlw       0
      DT          "zsaw2"     ;+1A, 1B, 1C, 1D, 1E
      retlw       0           ;+1F
      retlw       0           ;+20
      DT          "cxde43"    ;+21, 22, 23, 24, 25, 26
      retlw       0           ;+27
      retlw       0           ;+28
      retlw       ' '         ;+29 Space
      DT          "vftr5"     ;+2A, 2B, 2C, 2D, 2E
      retlw       0           ;+2F
      retlw       0           ;+30
      DT          "nbhgy6"    ;+31, 32, 33, 34,35,36
```

```
      retlw       0           ;+37
      retlw       0           ;+38
      retlw       0           ;+39
      DT          +"mju78"    ;+3A, 3B, 3C, 3D, 3E
      retlw       0           ;+3F
      retlw       0           ;+40
      DT          ",kio09"    ;+41, 42,43,44,45,46
      retlw       0           ;+47
      retlw       0           ;+48
      DT          "./l;p-"    ;+49, 4A, 4B, 4C, 4D, 4E
      retlw       0           ;+4F
      retlw       0           ;+50
      retlw       0           ;+51
      retlw       0x27        ;+52  single quote
      retlw       0           ;+53
      DT          "[="        ;+54, 55
      retlw       0           ;+56
      retlw       0           ;+57
      retlw       0           ;+58
      retlw       0           ;+59
      retlw       0x0D        ;+5A  Return
      retlw       ']'         ;+5B
      retlw       0           ;+5C
      retlw       0x5C        ;+5D  \
      retlw       0           ;+5E
      retlw       0           ;+5F
      retlw       0           ;+60
      retlw       0           ;+61
      retlw       0           ;+62
      retlw       0           ;+63
      retlw       0           ;+64
      retlw       0           ;+65
      retlw       0x08        ;+66  Backspace
;end if NoShiftKetTable

;With Shift  Key Table
shiftKeyTable
      bsf         PCLATH, 0x00
      bcf         PCLATH,0x01
      addwf       PCL
;Note that writing to PCL also brings the content of lower 5 bits of PCLATH
;to PC.
;In this code, this Table starts from 0134h
;While the main part of program which calls this Table is somewhere in
;0045h.
;for 0045, the PCLATH part is 00
;Therefore we have to manually set the PCLATH part so that it
;can point inside this table
      retlw       0           ;PC+0
      retlw       0           ;PC+1
      retlw       0           ;+2
      retlw       0
      retlw       0
      retlw       0
      retlw       0
      retlw       0
      retlw       0
```

```
retlw       0
retlw       0
retlw       0
retlw       0
retlw       0           ;+0D
retlw       0x7E        ;+0E MAKE/BREAK= 0E ---->ASCII 7E (~)
retlw       0           ;+0F
retlw       0
retlw       0
retlw       0
retlw       0           ;+13
retlw       0           ;+14
DT          "Q!"        ;+15, 16
retlw       0           ;+17
retlw       0
retlw       0
DT          "ZSAW@"     ;+1A, 1B, 1C, 1D, 1E
retlw       0           ;+1F
retlw       0           ;+20
DT          "CXDE$#"    ;+21, 22, 23, 24, 25, 26
retlw       0           ;+27
retlw       0           ;+28
retlw       ' '         ;+29 Space
DT          "VFTR%"     ;+2A, 2B, 2C, 2D, 2E
retlw       0           ;+2F
retlw       0           ;+30
DT          "NBHGY^"    ;+31, 32, 33, 34,35,36
retlw       0           ;+37
retlw       0           ;+38
retlw       0           ;+39
DT          "MJU&*"     ;+3A, 3B, 3C, 3D, 3E
retlw       0           ;+3F
retlw       0           ;+40
DT          "<KIO)("    ;+41, 42,43,44,45,46
retlw       0           ;+47
retlw       0           ;+48
DT          ">?L:P_"    ;+49, 4A, 4B, 4C, 4D, 4E
retlw       0           ;+4F
retlw       0           ;+50
retlw       0           ;+51
retlw       0x22        ;+52 double quote
retlw       0           ;+53
DT          "{+"        ;+54, 55
retlw       0           ;+56
retlw       0           ;+57
retlw       0           ;+58
retlw       0           ;+59
retlw       0x0D        ;+5A  Return
retlw       '}'         ;+5B
retlw       0           ;+5C
retlw       0x7C        ;+5D  |
retlw       0           ;+5E
retlw       0           ;+5F
retlw       0           ;+60
retlw       0           ;+61
retlw       0           ;+62
retlw       0           ;+63
```

```
      retlw       0           ;+64
      retlw       0           ;+65
      retlw       0x08        ;+66  Backspace

;CAPs Lock Key Table
      org         0x0200            ;This starts from 0200h
                                    ;to well arrange the PCH of PC
CAPKeyTable
      bsf         PCLATH, 0x01
      bcf         PCLATH, 0x00      ;see here that PCH is 02h?
      addwf       PCL

      retlw       0           ;PC+0
      retlw       0           ;PC+1
      retlw       0           ;+2
      retlw       0
      retlw       0
      retlw       0
      retlw       0
      retlw       0
      retlw       0
      retlw       0
      retlw       0
      retlw       0
      retlw       0
      retlw       0           ;+0D
      retlw       0x60        ;+0E MAKE/BREAK= 0E ---->ASCII = 0x60
Apostrophe
      retlw       0           ;+0F
      retlw       0
      retlw       0
      retlw       0
      retlw       0           ;+13
      retlw       0           ;+14
      DT          "Q1"        ;+15, 16
      retlw       0           ;+17
      retlw       0
      retlw       0
      DT          "ZSAW2"     ;+1A, 1B, 1C, 1D, 1E
      retlw       0           ;+1F
      retlw       0           ;+20
      DT          "CXDE43"    ;+21, 22, 23, 24, 25, 26
      retlw       0           ;+27
      retlw       0           ;+28
      retlw       ' '         ;+29 Space
      DT          "VFTR5"     ;+2A, 2B, 2C, 2D, 2E
      retlw       0           ;+2F
      retlw       0           ;+30
      DT          "NBHGY6"    ;+31, 32, 33, 34,35,36
      retlw       0           ;+37
      retlw       0           ;+38
      retlw       0           ;+39
      DT          "MJU78"     ;+3A, 3B, 3C, 3D, 3E
      retlw       0           ;+3F
      retlw       0           ;+40
      DT          ",KIO09"    ;+41, 42,43,44,45,46
      retlw       0           ;+47
```

```
      retlw       0           ;+48
      DT          "./L;P-"    ;+49, 4A, 4B, 4C, 4D, 4E
      retlw       0           ;+4F
      retlw       0           ;+50
      retlw       0           ;+51
      retlw       0x27        ;+52  single quote
      retlw       0           ;+53
      DT          "[="        ;+54, 55
      retlw       0           ;+56
      retlw       0           ;+57
      retlw       0           ;+58
      retlw       0           ;+59
      retlw       0x0D        ;+5A  Return
      retlw       ']'         ;+5B
      retlw       0           ;+5C
      retlw       0x5C        ;+5D  \
      retlw       0           ;+5E
      retlw       0           ;+5F
      retlw       0           ;+60
      retlw       0           ;+61
      retlw       0           ;+62
      retlw       0           ;+63
      retlw       0           ;+64
      retlw       0           ;+65
      retlw       0x08        ;+66  Backspace

;END OF CODE
        END
```

Compile and run the above example code, see how fast or slow the 16F988 responds and displays the keys.

5. Third Code - Display Key in LCD

The next, final version is to change the display medium from the PC monitor to the 20x4 LCD module, we studied in the Serial Communication. As in the digital clock, we will apply the 4-bit interface configuration to display characters.

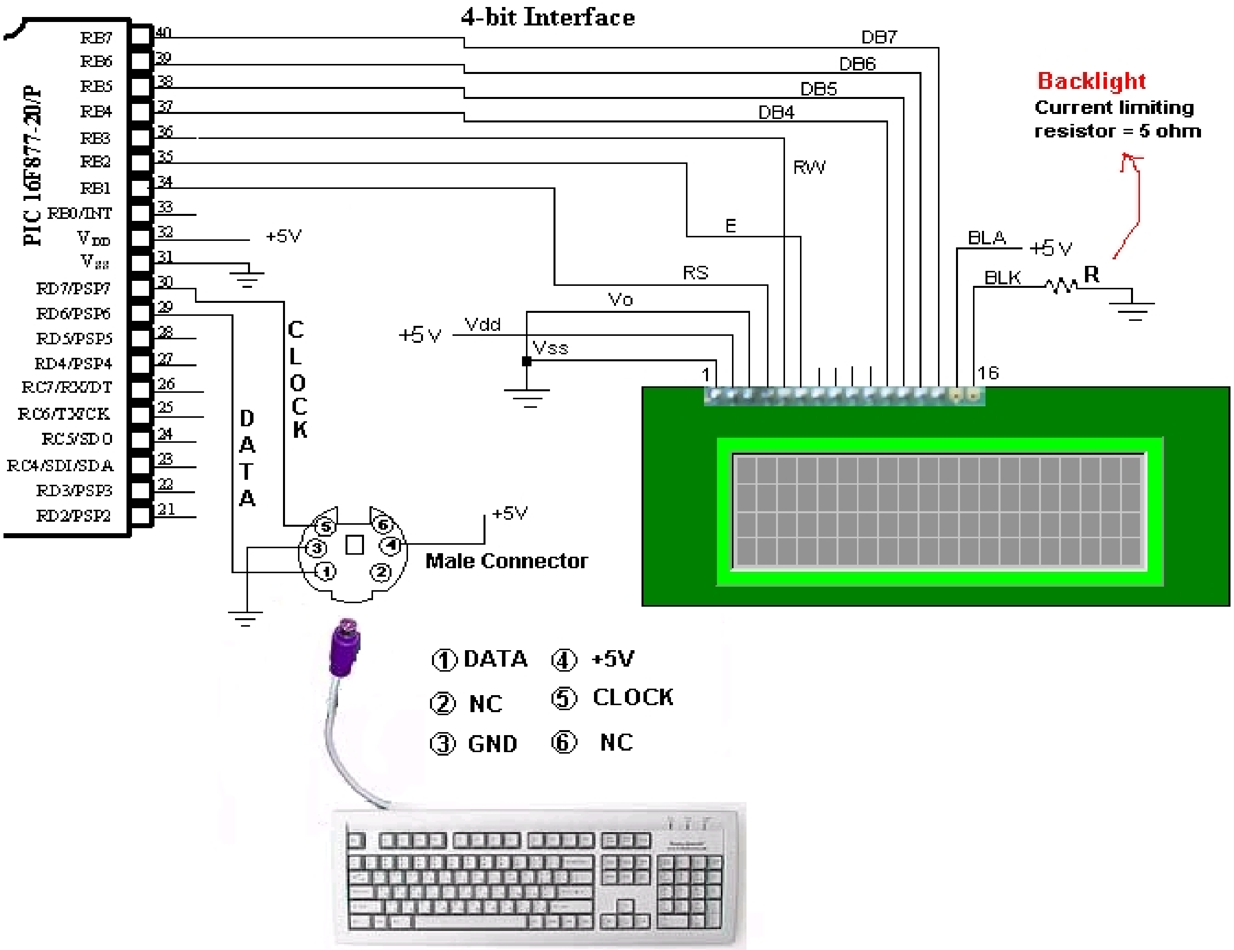


Fig. 78 Connection to display key in LCD

As shown in the schematic, PORTB is assigned to LCD control as we did before, and RD7 and RD6 are assigned to the DATA and CLOCK signal lines of the keyboard. In this final version, we will have two example codes. The first one is to display the keyboard on the LCD from the first column of the line 1 to the last column of the line 4. The first one, for convenience and simplicity, ignores Back Space (BS) and Carriage Return (CR) keys. The LCD controller inside the module does not have the stored character for BS and CR, therefore, no output will be displayed. The accommodation of these two keys is made in the second example code of this application.

Since most of the subjects here are related to the LCD control and keyboard reading, the only important thing is to remember the cursor location and its address and control them for display. In most LCD module, all the characters in ASCII code table and some other special characters are stored at the addresses which are the ASCII codes themselves. For example, the character 'A' in dot matrix form is stored at 41h, and the ASCII code of 'A' is 41h. This means that we can use the two key tables we used for PC monitor display without any change.

As we discussed in the LCD module in Chapter 6, there is somewhat weird address allocation of 20x4 positions of the LCD module. Here we show again the address of each display cell of 20x4 LCD module:

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

As you see the addresses are continuous from line 1 to line 3, and from line 2 and 4. Therefore, displaying characters continuously from the first line to the last involves tracking the current cursor position and its address. For example, if the current cursor position is the 20^{th} position of line 2 (address = 53h), the next cursor position must be the 1^{st} position of line 3 (address = 14h).

Indeed, there is a way to read the current cursor address by reading the address from the LCD module. However, actually, reading the address after every writing a data into LCD is not necessary. In the reset, the LCD is usually configured to start from the first position of the first line, and as a character is displayed the address incremented automatically. Therefore, if we assign the cursor position, not the cursor address which must be read from LCD module, and increase the cursor position every time we write a data to the LCD module, we can easily track the current cursor position. So in the first example code for LCD display, we assign a file register `CURSOR` to track the current cursor position. The `CURSOR`'s value of the cursor position, in the continuous order, is assigned as follows.

Position	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
First line	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14
Second line	15	16	17	18	19	1A	1B	1C	1D	1E	1F	20	21	22	23	24	25	26	27	28
Third line	29	2A	2B	2C	2D	2E	2F	30	31	32	33	34	35	36	37	38	39	3A	3B	3C
Fourth Line	3D	3E	3F	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F	50

So when it starts, the `CURSOR`=1 while the cursor address is 00h. Similarly, `CURSOR`=3B while the cursor address in the LCD module is 26h. Also, if `CURSOR`=15, we change the cursor address to 40h so that the LCD module actually moves the cursor to the first position of the second line. The cursor is not changed accordingly unless the cursor address is changed according to the address table: `CURSOR` itself cannot change the cursor position; it is only for our convenience in cursor position tracking.

Since our LCD configuration automatically increases the cursor address by one whenever a character is displayed, we increase the `CURSOR` by 1 every time we write a character to the LCD module, and we check in which line the cursor is currently pointed. If the current cursor position is, for example, at the last position of line 1, then, the next `CURSOR` value must be changed to the first position of line 2.   This seems quite trivial since `CURSOR` value is well ordered; however, the cursor address is not so well ordered. Since we do not have the exact positional information on the cursor, we rely only on `CURSOR` to properly change the cursor address for the correct next cursor address depending upon the current position.

The following code is the usual LCD initialization routine for 4-bit interfacing. Here the starting cursor position is position 1 at line 1. The four-bit write routines, `hnibble4`, `instw4`, and `dataw4`, are those we already built and used in Chapter 6. Use them without any change here.

```
;SUBROUTINE LCD4INIT
;Function for 4-bit (only one write must be done)
;In other words, send only the high nibble
LCD4INIT
;IMPORTANT PART
      movlw       0x28
      call        hnibble4
;Fundtion for 4-bit, 2-line display, and 5x8 dot matrix
      movlw       0x28
      call        instw4
;Display On, CUrsor On, No blinking
      movlw       0x0E        ;0F would blink
      call        instw4
;DDRAM address increment by one & cursor shift to right
      movlw       0x06
      call        instw4
;DISPLAY CLEAR
CLEAR
      movlw       0x01
      call        instw4
;
      call        posline11         ;pos1 and line 1
;now CURSOR=1
      return
```

The following subroutine, `LCDisplay`, is the main displaying routine monitoring and handling the cursor positions and addresses.

```
;LCD DISPLAYING SUBROUTINE
LCDisplay

      call        dataw4            ;write a character
      incf        CURSOR            ;every time of display, increase cursor
;CURSOR is automatically incremented by 1 from LCDisplay
;if CURSOR is 20 (0x14), change to posline12
;if CURSOR is 40 (0x28), change to posline13
;if CURSOR is 60 (0x3C), change to posline14
;if CURSOR is 80 (0x50), change to posline11
      clrf        STATUS
      movf        CURSOR,0          ;if the CURSOR is supposed to be
                                    ;pos 1 and line 2, the Cursor Address
must
                                    ;be changed also
      xorlw       0x15
      btfsc       STATUS, ZERO
      goto        Toline2

      clrf        STATUS
      movf        CURSOR,0
```

```
      xorlw       0x29
      btfsc       STATUS,ZERO
      goto        Toline3

      clrf        STATUS
      movf        CURSOR,0
      xorlw       0x3D
      btfsc       STATUS,ZERO
      goto        Toline4

      clrf        STATUS              ;if the cursor is at the last pos
                                      ;at the 4th line, the next cursor
                                      ;position is the pos 1 at line 1
                                      ;after clearing the LCD
      movf        CURSOR,0
      xorlw       0x51
      btfsc       STATUS,ZERO
      call        LCDClearhome              ;delete all an move to (1,1)
      return

Toline2
      call  posline12
      return
Toline3
      call  posline13
      return
Toline4
      call  posline14
      return
```

Clearing the LCD and returning to position 1 at line is done by the following subroutine, `LCDclrearhome`. The first two instruction write clears and move the cursor to the "home" position. The next, third, writing is not necessary but used anyway to show the actual cursor address and the variable `CURSOR` we use throughout our example code.

```
;SUBROUTINE
;DISPLAY CLEAR and Cursor to Home position (line 1, position 1)
LCDclearhome
      movlw       0x01
      call        instw4
;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        instw4
;home position
      movlw       0x80
      call        instw4
      movlw       0x01
      movwf       CURSOR
      return
```

The following four subroutines are for moving the cursor address to the first positions of the four lines, respectively. Note and see the matching hex values for `CURSOR` and the actual cursor address values that are written by `instw4` subroutine.

```
posline11
;Position to pos 1 and line 1
      movlw       0x80
      call        instw4            ;Cursor address for (1,1)
      movlw       0x01
      movwf       CURSOR
      return

posline12                     ;pos 1 and line 2
      movlw       0xC0
      call        instw4
      movlw       0x15              ;21
      movwf       CURSOR
      return

posline13                     ;pos1 and line3
      movlw       0x94
      call        instw4
      movlw       0x29              ;41
      movwf       CURSOR
      return
posline14                     pos 1 and line 4
      movlw       0xD4
      call        instw4
      movlw       0x3D              ;61
      movwf       CURSOR
      return
```

The full example code, without subroutine listings, follows below.

```
;KBD4.asm
;
;NOTE: In this program
; BACK SPACE key is not honored
; CR key is not recognized
;
;
;This program is
;1. To read keys from AT or PS/2 keyboard
;2. to display the key on the 20x4 LCD module by Truly (HD44780 compatible)
;3. Displays from the first dot matrix to the last one
;4. 4. IF the last dot is reached, it is cleared and restart from the first
dot
;
; LCD is with 4-bit interfacing
;
```

```
;CR key would change the line
;
; 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)
;E (6)  ------RB2(35)
;RW (5)  -----RB3(36)
;RS (4)  -----RB1(24)
;Vo (3)  -----GND
;Vdd (2) ----+5V
;Vss (1) -----GND
;
;KEYBOARD Interfacing
;CLOCK -----RD7 (input)
;DATA ------RD6 (input)
;
;

      list P = 16F877

STATUS      EQU   0x03
PCL         EQU   0x02        ;For Key Table Calling
PCLATH      EQU   0x0A        ;upper part of PC
CARRY       EQU   0x00
ZERO        EQU   0x02
PORTB       EQU   0x06
TRISB       EQU   0x86
RS          EQU   0x01        ;RB1
E           EQU   0x02        ;RB2
RW          EQU   0x03        ;RB3
TRISD       EQU   0x88
PORTD       EQU   0x08
CARRY       EQU   0x00
MSB         EQU   0x07
CLOCK       EQU   0x07        ;from Keyboard (RD7)
KDATA       EQU   0x06        ;from Keyboard (RD6)


;RAM

      CBLOCK      0x20
            CURSOR            ;tracking the current display position
;CURSOR
;1  2  3   4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20  line 1
;21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40  line 2
;41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60  line 3
;61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80  line 4
;
            Daddr       ;Display address (cursor pos)
            Dkey        ;Key character to be displayed
            DATAreg
            Bitcount
            Kstat
            Kount120us   ;Delay count (number of instr cycles for delay)
```

```
            Kount100us
            Kount1ms
            Kount10ms
            Kount1s
            Kount10s
            Kount1m
            Temp        ;temp storage
      ENDC

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

      org         0x0000
      goto        START


      org         0x05
Start

      banksel     TRISD
; 1100 0000

      movlw       B'11000000' ;RB7 for CLOCK and RB6 for DATA as inputs
      movwf       TRISD

      call        delay1s           ;Give Keyboard to send STATUS to the host

      BANKSEL     TRISB
      movlw       0x00
      movwf       TRISB       ;All output

      banksel     PORTB
      clrf        PORTB       ;RW set LOW here

      clrf        CURSOR            ;Current Display Location
      incf        CURSOR            ;Home cursor position (1, 1)
;LCD routine starts
      call        delay10ms
      call        delay10ms

      banksel     PORTB
      clrf        PORTB       ;RW set LOW here
                  ;give LCD module to reset automatically
      call        LCD4init

;==============================================================
;KBD Monitoring
BEGIN

      banksel     DATAreg
      clrf        DATAreg


; CHECK IF THE CLCOK is HIGH at least for 10mS

      banksel     PORTD
      btfss       PORTD, CLOCK
```

```
      goto        BEGIN       ;if CLOCK is LOW, start again
      call  Delay10ms   ;10mS delays

;check again for CLCOK
      btfss       PORTD, CLOCK
      goto        BEGIN
;READY FOR CLOCK PULSES

      clrf        KSTAT
KEYIN
;X reading
      call        RX11bit                 ;
      clrf        STATUS
      movf        DATAreg,0   ;Break Code?
      xorlw       0xF0
      btfss       STATUS,ZERO
      goto        CAT
;BREAK is detected. Abort It. Resume It
      goto        BEGIN
;Category detection
CAT   clrf        STATUS
      movf        DATAreg,0
      xorlw       0xE0
      btfsc       STATUS,ZERO
      goto        Begin       ;E0 keys (CAT2) are ignored
      clrf        STATUS
      movf        DATAreg,0
      xorlw       0x12
      btfsc       STATUS,ZERO
      goto        LRSHIFT
      clrf        STATUS
      movf        DATAreg,0
      xorlw       0x59
      btfsc       STATUS,ZERO
      goto        LRSHIFT
      clrf        STATUS
      movf        DATAreg,0
      xorlw       0x58        ;CAPS LOCK
      btfsc       STATUS,ZERO
      goto        CAPS
      movf        DATAreg,0
      clrf        STATUS            ;CR check
      xorlw       0x5A
      btfsc       STATUS,ZERO
      goto        CRhandle
                              ;L Shift ===>12 | F0 12
                              ;R Shift ===>59 | F0 59
;CAT1 has the format of (X)|(F0)(X)
CAT1  movf  DATAreg,0
;check if the key in is CR
;Then we have to move the next line

      call        NoShiftKeyTable         ;(X) display
      call        LCDisplay
;(F0) detection
      call        RX11bit
      clrf        STATUS
```

```
      movf        DATAreg,0
      xorlw       0xF0
      btfss       STATUS,ZERO
;Key is not broken. Still pressed,
      goto        CAT1
;Key is broken
;Last (X) reading
      call        RX11bit                 ;(X) after F0

      goto        EGIN

;L-SHIT and R-SHIFT has the form
;L-SHIFT and a character 12 X | F0 X |F0 12
;R-SHIFT and a character 59 X | F0 X |F0 59

LRSHIFT                             ;12 or 59 entered

;(F0) detection
      call        RX11bit
      clrf        STATUS
      movf        DATAreg,0
      xorlw       0xF0
      btfsc       STATUS,ZERO
      goto        BEGIN

                              ;X

      clrf        STATUS            ;if (12) do not display
      movf        DATAreg,0
      xorlw       0x12
      btfsc       STATUS, ZERO
      goto        LRSHIFT

      clrf        STATUS            ;if (59) do not display
      movf        DATAreg,0
      xorlw       0x59
      btfsc       STATUS, ZERO
      goto        LRSHIFT

                        ;a Key (X) is entered
      movf        DATAreg,0
      call        ShiftKeyTable
      call        LCDisplay


;(F0) detection
      call        RX11bit
      clrf        STATUS
      movf        DATAreg,0
      xorlw       0xF0
      btfss       STATUS,ZERO
      goto        LRSHIFT
;Last (X) reading
      call        RX11bit
      movf        DATAreg,0
      clrf        STATUS            ;check if (X) or (12) entered after F0
      xorlw       0x12
```

```
      btfsc       STATUS,ZERO
      goto        BEGIN
      goto        LRSHIFT
;
CAPS                          ;caps lock (58) entered

;(F0) detection
      call        RX11bit                     ;this must be F0
      call        RX11bit                     ;this must be (58) again

CAPnext
      call        RX11bit                     ;Check if (58) or other
      clrf        STATUS
      movf        DATAreg,0
      xorlw       0x58
      btfss       STATUS,ZERO
      goto        CAPtwo                      ;End of CAP session
      call        RX11bit                     ;F0
      call        RX11bit                     ;(58)
      goto        BEGIN
                              ;a Key (X) is entered
CAPtwo
      movf        DATAreg,0
      call        CAPKeyTable
      call        LCDisplay

;(F0) detection
      call        RX11bit                     ;this
      clrf        STATUS
      movf        DATAreg,0
      xorlw       0xF0
      btfss       STATUS,ZERO
      goto        CAPtwo


;Last (X) reading             ;F0 is read
      call        RX11bit                     ;(X) again and ignore
      goto        CAPnext


;CR handling
CRhandle
      call        RX11bit           ;F0 read
      call        RX11bit           ;CR reading again
;have to move the cursor to the next line at the first position
;
;Routine Here

      goto        BEGIN

;SUBROUTINES and TABLES HERE
;
;HERE
      END
;END of CODE
```

6. A complete Keyboard-LCD Operation with Back Space and Carriage Return

The next version of Keyboard and LCD connection with a 16F877 is to accommodate the two keys we did not: Back Space (BS) and Carriage Return (CR) keys. When BS key is detected we have to move the cursor position to the left by 1 position. This, then, needs the current cursor position address. When CR is detected, again, the cursor position address must bee provided so that we move the cursor to the first position of the next line.

These new features require reading information from the LCD controller/interface inside the LCD module, which we have not discussed at all. We only discussed about writing instructions and data to the LCD controller. The reason we need reading information, especially the current cursor address information, is that, as we mentioned before, we do not have direct knowledge on the cursor position address unless we monitor the position every time we write a character on the LCD. Of course there is an indirect way to fulfill this task: monitoring `CURSOR` value every time we write a character to LCD and interpret the cursor position address from the `CURSOR` value.

As discussed above, CURSOR is the variable assigned for convenience; it does not indicate the cursor position inside the DD(Display Data) RAM of the LCD module. But reading the cursor position address would be better because we need not this reading all the time as we have to do in the `CURSOR` tracing approach; we do this cursor position address reading only when CR or BS key is detected. In conclusion, whether we use an arbitrary variable for cursor tracking or not, reading the location address from the LCD interface/controller brings more convenience and gives a good practice of more utilizing a LCD module.

As we studied in Chapter 6, there is a command in HD44780 or equivalent LCD controller/interface of "Reading Busy Flag and DD RAM address" as shown below. On additional command we include is the cursor address setting command. These two are most relevant in the discussion of this version of keyboard - LCD connection.

	Code										
Instruction	RS	R/$\overline{W}$	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Description
Set DDRAM address	0	0	1	ADD	ADD	ADD	ADD	ADD	ADD	ADD	Sets DDRAM address. DDRAM data is sent and received after this setting.
Ready busy flag & address	0	1	BF	AC	AC	AC	AC	AC	AC	AC	Reads busy flag (BF) indicating internal operation is being performed and reads address counter contents.

ADD: DDRAM address (corresponds to cursor address)
AC: Address counter used for DDRAM address

As the table shows, we read 7-bit DDRAM address of the current cursor position. We do not care much on the flag bit, BF, if the LCD is ready to receive data or not, since our interest is the 7 bits returned to 16F877.

By the way, cursor address read from the LCD module for 20x4 display format based on the position and the line is as follows:

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

To read the cursor address, we have to clear the RS pin and set the RW bit. Since we use 4-bit interfacing, we need two consecutive commanding to the LCD to read the total 8-bit data composed of the BF bit and the 7-bit cursor address. Care should be taken when we read cursor address from LCD module. Usually we set all the pins of PORTB as outputs since the upper 4 data lines RB<7:4> and RW, E, RS are all outputs. However, when we read the data bits RB<7:4> must be changed to input pins.

Another caution we have to use is that in both readings, even though there are only 4 data lines involved, when we read data from PORTB as a byte oriented instruction we read the whole 8 bits anyway. Since the first reading gets upper nibble and the second, the lower nibble, we have to extract the nibbles properly to form a byte hex number. The following code illustrates the DDRAM address reading subroutine, `readad4`, in 4-bit interfacing environment.

As you see at the bottom of the subroutine, once the reading is done, we move back to the usual writing mode by setting the PORTB<7:4> as outputs and clearing the RW line.

```
;subroutine reading the cursor position
;RW Must be High
;RS Must be Low
;the 7th bit is BF flag (so ignore this one, or make MSB 0)
;PORTB <7:4> as inputs
;High then Low nibbles of ADDRESS
;The content of DDADDR read from LCD module (HEX Numbers)
;Line 1: 00 01 02 ................ 13
;Line 2: 40 41 42 ................ 53
;Line 3: 14 15 16 ................ 27
;Line 4: 54 55 56 ................ 67


readad4
      banksel     TRISB             ;set Rb7 - DR4 as inputs
      movlw       0xF0              ;upper 4 bits as inputs
      movwf       TRISB
      banksel     PORTB
      bsf         PORTB, RW         ;READING MODE
      call        delay1ms
      bcf         PORTB,RS
      call        delay1ms
      bsf         PORTB, E
      call        delay1ms
      bcf         PORTB, E          ;Reading starts here now
                                    ;upper byte first
```

```
                                        ;now PORTB<7:4> has the DDRAM ADDRESS
                                        ;upper nibble
      movlw        0xF0
      andwf        PORTB,0
      movwf        DDtemp1              ;Ddtemp1 = DDADDR<7:4>|0000
                                        ;Reading for the second nibble
      bcf          PORTB,RS
      call         delay1ms
      bsf          PORTB, E
      call         delay1ms
      bcf          PORTB, E
                                        ;reading starts now
                                        ;for lower byte
                                        ;PORTB<7:4> has the DDADDR<3:0>
      movlw        0xF0
      andwf        PORTB,0
      movwf        DDtemp2              ;Ddtemp2 = DDADDR<3:0>|0000
      swapf        DDtemp2              ;Ddtemp2 = 0000|DDADDR<3:0>

;add DDtemp1 and DDtemp2 for DDADDR
;
      movf         DDtemp1,0
      addwf        DDtemp2,0
      movwf        DDADDR               ;DDADDR=DDADDR<7:4>|DDADDR<3:0>

                                        ;END of Reading
      banksel      TRISB                ;Change to Write Mode
      movlw        0x00
      movwf        TRISB                ;all outputs again
      banksel      PORTB
      bcf          PORTB,RW             ;back to writing mode
      return
```

The next subject is to include in the Category classification section of the code BS and CR key detection part. This is done by simply adding a few lines of the entered key check. The addition for these two keys is at the bottom of this category classification part of the code.

```
;Category detection
CAT   clrf         STATUS
      movf         DATAreg,0
      xorlw        0xE0
      btfsc        STATUS,ZERO
      goto         Begin        ;E0 keys (CAT2) are ignored
                                ;Shift Key Detection
      clrf         STATUS
      movf         DATAreg,0
      xorlw        0x12
      btfsc        STATUS,ZERO
      goto         LRSHIFT
      clrf         STATUS
      movf         DATAreg,0
      xorlw        0x59
      btfsc        STATUS,ZERO
      goto         LRSHIFT
```

```
      clrf        STATUS
      movf        DATAreg,0
      xorlw       0x58              ;CAPS LOCK
      btfsc       STATUS,ZERO
      goto        CAPS

      movf        DATAreg,0
      clrf        STATUS            ;CR check
      xorlw       0x5A
      btfsc       STATUS,ZERO
      goto        CRhandle

      movf        DATAreg,0
      clrf        STATUS
      xorlw       0x66
      btfsc       STATUS,ZERO
      goto        BShandle          ;Back Space Handling
```

So when the BS is entered we handle the case by executing the `Bshandle` part. CR would jump to `CRhandle`. Let's discuss about handling when CR key is entered. When CR key is entered, we read the next two frames (i,e, F0k and 0Dh break codes) from the keyboard and ignore them, then we change the new cursor position to the first position at the next line.

Therefore we have to know the current cursor position stored in the DDRAM address. Once the cursor position is read, we have to figure out at which line the cursor is located. The cursor position is read in 7-bit format, however, when we set the cursor position by writing an instruction (See above code table), the DB7 pin must be High, so as soon as we read the cursor position address, we set the 7th bit (MSB) of the address, so that we directly write the address as the new cursor position.

The `CRhandle` routine listed below shows how to find at which line the current cursor is located by the DDRAM address read and to change the new cursor address to the first position of the next line. In the routine, we notice that we use Borrow flag (which is same as the Carry flag used in `add` instruction) in `sub` instruction to find the current line position of the cursor, by employing `sublw k` instruction. The `sublw k` instruction is to have the operation of (k – W $\rightarrow$ W; subtract W from k and store the result to W) and check if k is bigger than W or not: if k is bigger than W there is no Borrow so the Borrow flag in STATUS register is set. The Borrow flag is kind of active low flag which clears when the condition is met. Conclusively, if the Borrow bit is set, the k is bigger than W.

```
;CR handling
CRhandle
      call        RX11bit           ;F0 read
      call        RX11bit           ;CR reading again
;read the current cursor position
      call        readad4
;DDADDR has the content
;NOTE: MSB must be 1 in the cursor command
      bsf         DDADDR, MSB       ;set the 7th bit
;if DDADDR<94, then new cursor position is C0
;if DDADDR<E8, then 80
```

```
;if DDADDR<C0, then D4
;if DDADDR<D4, then 94
      clrf        STATUS
      movf        DDADDR,0
      sublw       0x94              ;k-W -->W
      btfsc       STATUS,Borrow     ;No borrow means that k>W
      goto        CR94              ;is less than 94 i.e.,cursor is at line 1
      clrf        STATUS
      movf        DDADDR,0
      sublw       0xC0
      btfsc       STATUS,Borrow
      goto        CRC0              ;cursor in at line 3

      clrf        STATUS
      movf        DDADDR,0
      sublw       0xD4
      btfsc       STATUS,Borrow
      goto        CRD4              ;cursor is at line 2

      clrf        STATUS
      movf        DDADDR,0
      sublw       0xE8
      btfsc       STATUS,Borrow
      goto        CRE8              ;cursor is at line 4
      goto        BEGIN             ;cursor position out of range

CR94  call        posline12         ;move the cursor to pos 1 line 2
      goto        begin
CRC0  call        posline14         ;move the cursor to pos 1 line 4
      goto        BEGIN
CRD4  call        posline13         ;move the cursor to pos 1 line 3
      goto        BEGIN
CRE8  call        posline11         ;move the cursor to pos 1 line 1
      goto        BEGIN
```

The `BShandle` routine is not very different from `CRhandle` part. Once BS is entered, the next two frames must read but ignored. What it does is just to reduce the cursor address by 1 and write a cursor position instruction to LCD so that the cursor is 1 place left shifted.

One glitch in this simple procedure is that, when the current cursor is at the first position of a line, the new cursor should be moved to the last position of the line one above. Except that if the cursor is located at the first position of line 1, there is no change and keep the current position evne though BS action. Therefore, here again is where the current position is also an important part of the routine.

The majority of the routine is, therefore, dedicated to find if the current cursor is at the first position in any of 4 lines. If the cursor is not at the first position, we simply decrease the cursor address by 1 and write it back for the new DDRAM address. The following routine is for the BS key handler.

```
;BS Handling
BShandle
      movf        DATAreg,0         ;W holds $66
```

```
      call        RX11bit           ;F0 read
      call        RX11bit           ;BS break code
;read the current cursor position
      call        readad4
;DDADDR has the content
; SO move the current to the left
;NOTE: MSB must be 1 for commanding of the cursor position
      bsf         DDADDR, MSB
;if DDADDR = 94, then new cursor position is D3
;if DDADDR = C0, then new position is 93
;if DDADDR = D4, then new position is A7
;if DDADDR = 80, then new position is 80 (NO CHANGE)
; all other cases, new position is (DDADDR - 1)
      clrf        STATUS
      movf        DDADDR,0
      xorlw       0x94
      btfsc       STATUS, ZERO
      goto        DD94                    ;cursor in pos 1 line 3
      clrf        STATUS
      movf        DDADDR,0
      xorlw       0xC0
      btfsc       STATUS,ZERO
      goto        DDC0                    ;cursor in pos 1 line 2
      clrf        STATUS
      movf        DDADDR,0
      xorlw       0xD4                    ;cursor in pos 1 line 4
      btfsc       STATUS,ZERO
      goto        DDD4
      clrf        STATUS
      movf        DDADDR,0
      xorlw       0x80
      btfsc       STATUS,ZERO
      goto        DD80                    ;cursor in pos 1 line 1
;all others
      decf        DDADDR
      decf        CURSOR
      movf        DDADDR,0
      call        instw4
      goto        BEGIN

DD94  movlw       0xD3        ;move cursor to pos 20 line 2
      decf        CURSOR
      call        instw4
      goto        BEGIN

DDC0  movlw       0x93        ;move cursor to pos 20 line 1
      decf        CURSOR
      call        instw4
      goto        BEGIN

DDD4  movlw       0xA7        ;move cursor to pos 20 line 3
      decf        CURSOR
      call        instw4
      goto        BEGIN

DD80  movlw       0x80        ;move cursor to pos 1 line 1
      call        instw4
```

```
	goto		BEGIN
```

Except these new routines and subroutines, the code for this updated for CR and BS handling is not different from the previous version. Make a full code and run the program to see if the CR and BS keys are working as we intended them to work.