Chapter 9

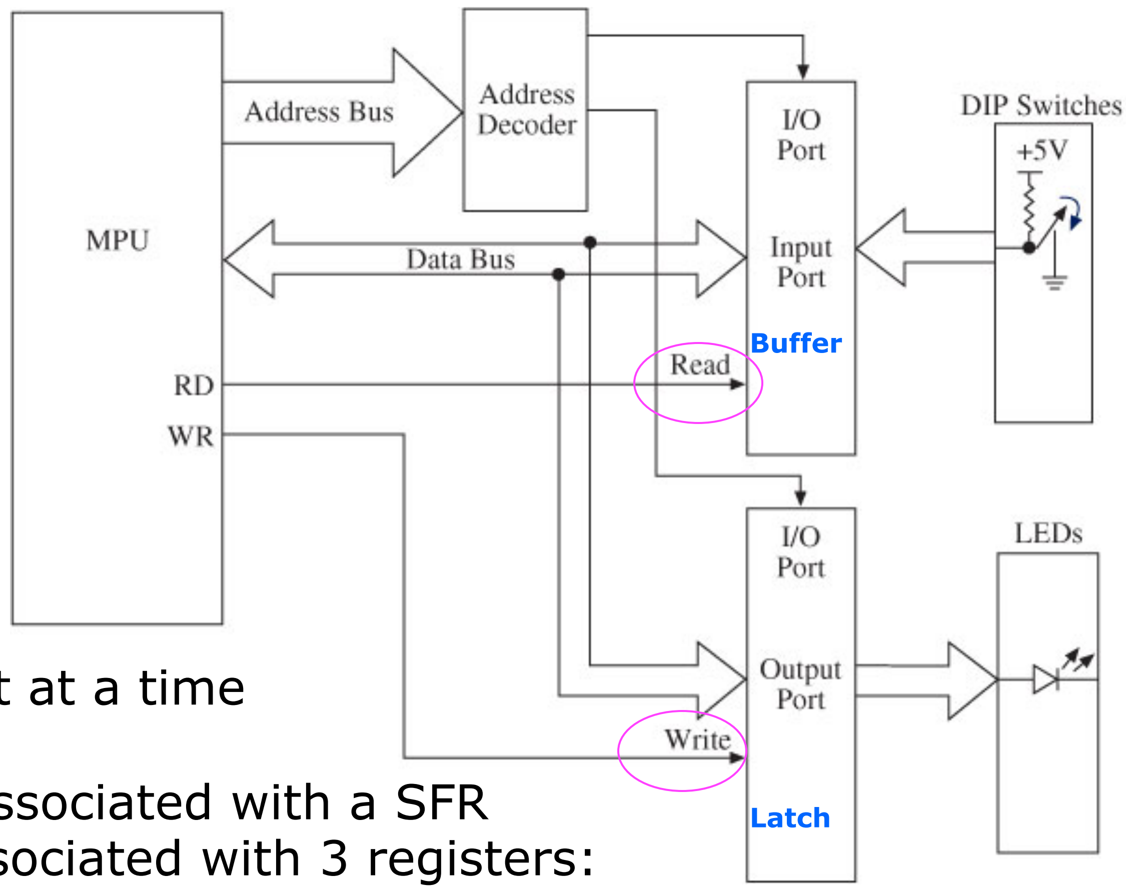
Input/Output (I/O) Ports and Interfacing

Updated: 3/13/12

Basic Concepts in I/O Interfacing and PIC18 I/O Ports (1 of 2)

- I/O devices (or peripherals) such as LEDs and keyboards are essential components of the microprocessor-based or microcontroller-based systems.
  - Classified into two groups
    - input devices
    - output devices

Block Diagram of I/O Interfacing



Access one port at a time
8-bit registers
I/O ports are associated with a SFR
Each port is associated with 3 registers:

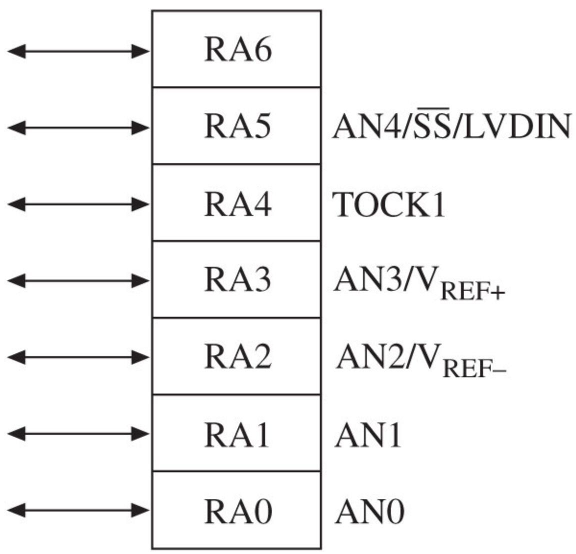
PORT / LAT / TRIS

I/O Ports: Interfacing and Addressing

- To read (receive) binary data from an input peripheral
  - MPU places the address of an input port on the address bus, enables the input port by asserting the RD signal, and reads data using the data bus.
- To write (send) binary data to an output peripheral
  - MPU places the address of an output port on the address bus, places data on data bus, and asserts the WR signal to enable the output port.
- Remember:
  - Writing to the port
    - When the MPU sends out or transfers data to an output port
  - Reading from the port
    - When the MPU receives data from an input port

PIC18F452/4520 I/O Ports (1 of 5)

- MCU includes five I/O ports
  - PORTA, PORTB, PORTC, PORTD, and PORTE
- Ports are multiplexed meaning they can be set up by writing instructions to perform various functions



PORTA: Example of Multiple Functions

- Digital I/O: RA6-RA0
- Analog Input: AN0-AN4
- V_{REF}+ : A/D Reference Plus Voltage
- V_{REF}- : A/D Reference
- Minus Voltage
- TOCK1: Timer0 Ext. Clock
- SS: SPI Slave Select Input
- LVDIN: Low voltage Detect Input

PIC18F452/4520 I/O Ports (2 of 5)

- Each I/O port is associated with the special functions registers (SFRs) to setup various functions.
  - Can be set up as entire ports or each pin can be set up.
    - PORT: This register functions as a latch or a buffer determined by the logic levels written into the associated TRIS register.
    - TRIS: This is a data direction register. Writing logic 0 to a pin sets up the pin as an output pin, and logic 1 sets up the pin as an input pin.
    - LAT: This determines if port is bidirectional .

PIC18F452/4520 I/O Ports (3 of 5)

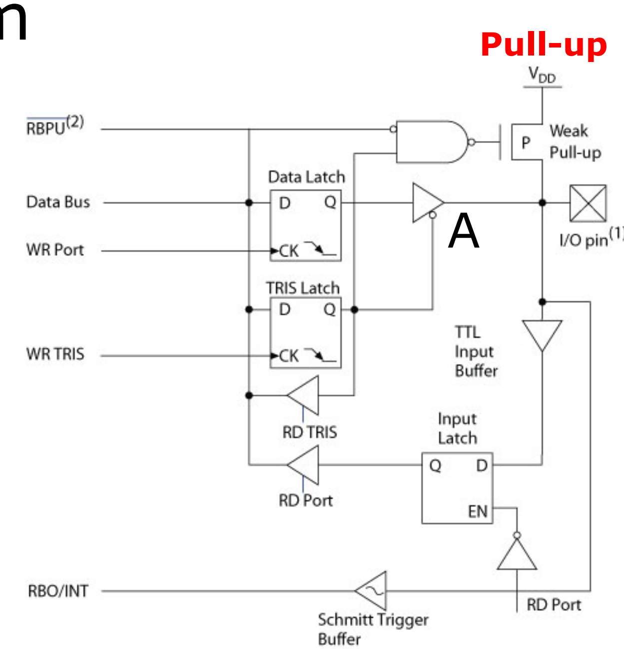
- Internal block diagram of PORTB includes:
  - Three internal D flip-flops (latches)
    - Data latch to output data
    - TRIS latch to setup data direction
    - Input latch for input data

PIC18F452/4520 I/O Ports (4 of 5)

- PORTB Internal Block Diagram

Three internal D flip-flops (latches):

Data latch to output data
TRIS latch to setup data direction
Input latch for input data



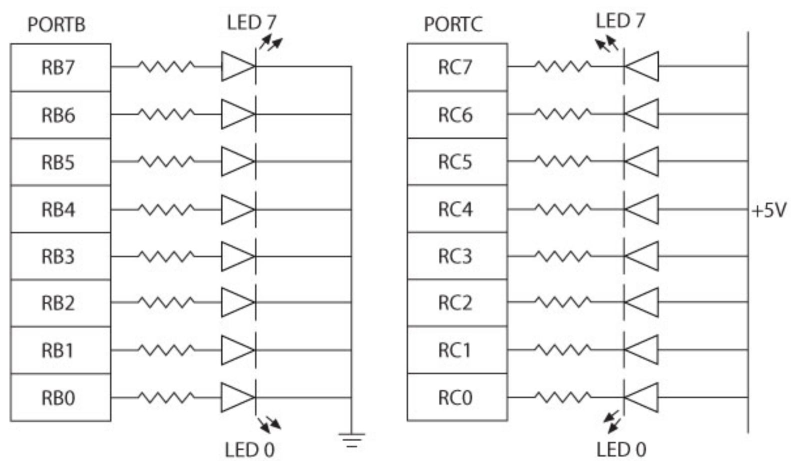
Not

Q- TRIS: 0 → A is enabled

Q- TRIS: 1 → A is disabled

Interfacing Output Peripherals (1 of 2)

- Commonly used output peripherals in embedded systems are
  - LEDs, seven-segment LEDs, and LCDs; the simplest is LED
- Two ways of connecting LEDs to I/O ports:
  - LED cathodes are grounded and logic 1 from the I/O port turns on the LEDs - The current is supplied by the I/O port called current sourcing.
  - LED anodes are connected to the power supply and logic 0 from the I/O port turns on the LEDs - The current is received by the chip called current sinking.



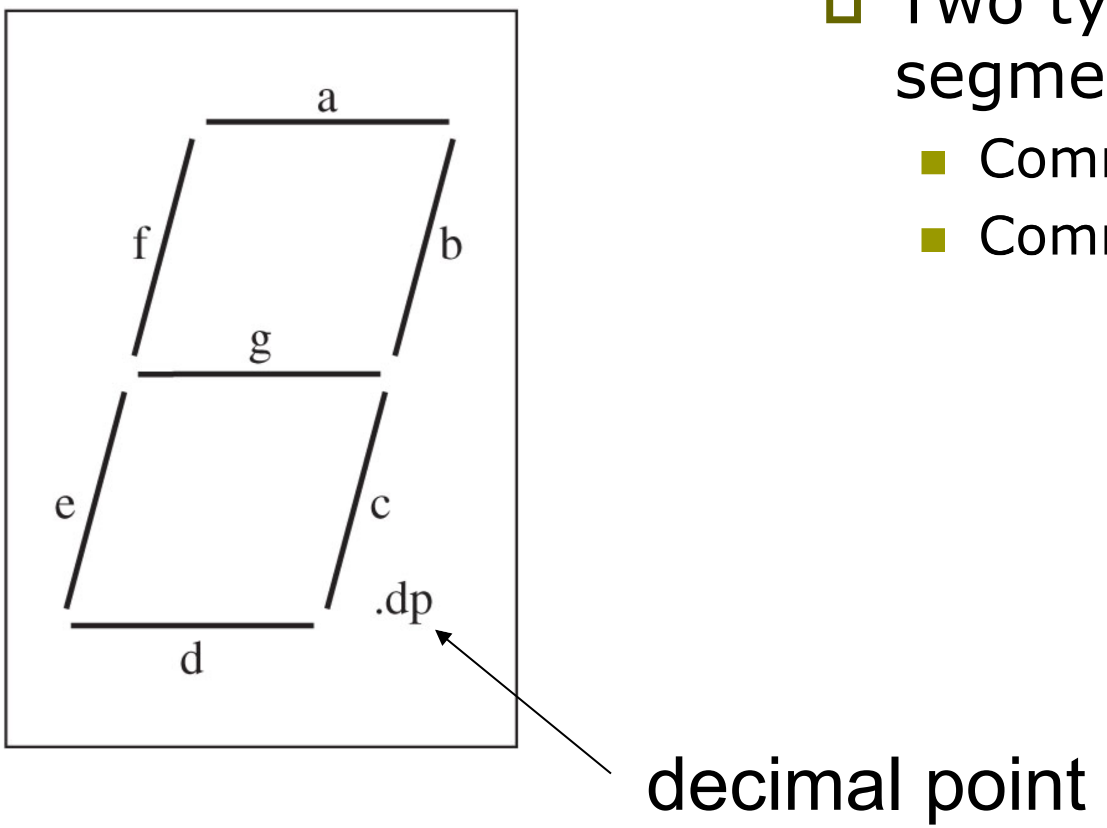
Common Cathode
Active high

Common Anode
Active low

Interfacing Seven-Segment LEDs as an Output (1 of 4)

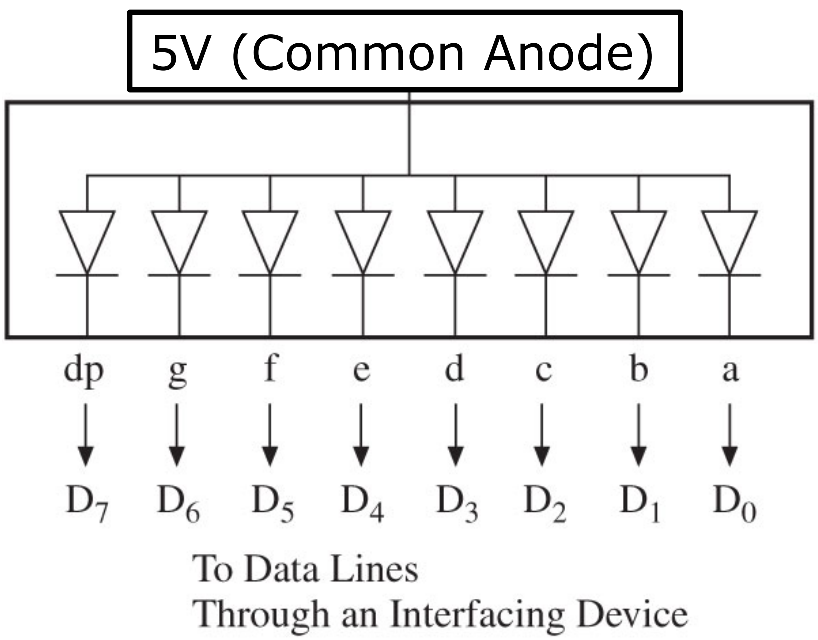
- Seven-segment LEDs
  - Often used to display BCD numbers (1 through 9) and a few alphabets
  - A group of eight LEDs physically mounted in the shape of the number eight plus a decimal point as shown in Figure 9-5 (a)
  - Each LED is called a segment and labeled as ‘a’ through ‘g’.

Interfacing Seven-Segment LEDs as an Output (2 of 4)



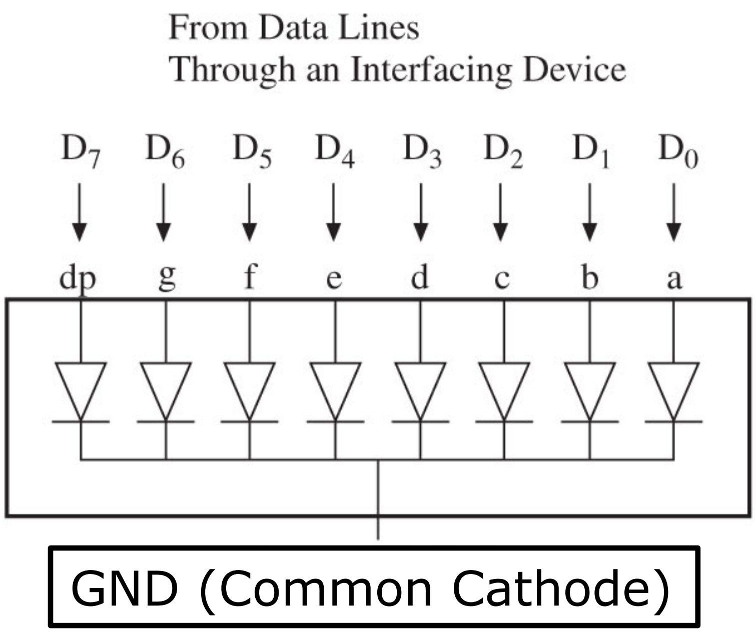
- Two types of seven-segment LEDs
  - Common anode
  - Common cathode

Interfacing Seven-Segment LEDs as an Output (3 of 4)



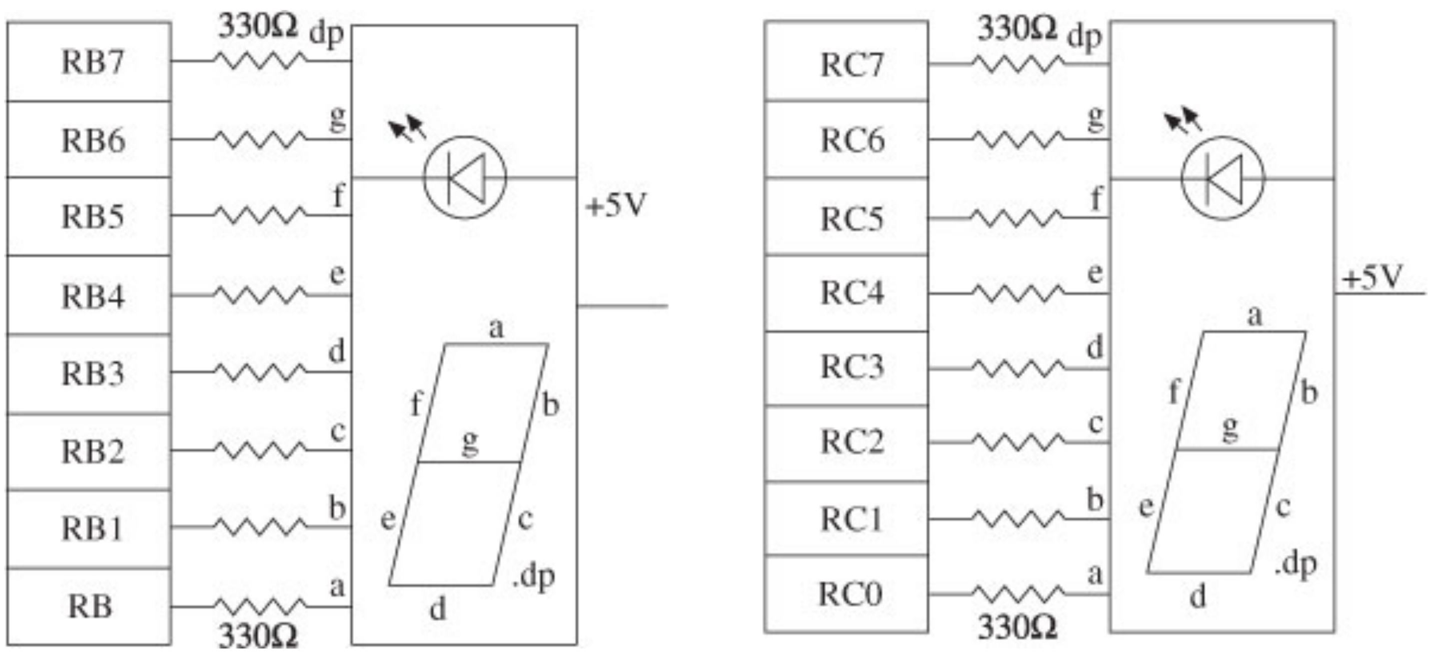
- In a common anode seven-segment LED
  - All anodes are connected together to a power supply and cathodes are connected to data lines
- Logic 0 turns on a segment.
- Example: To display digit 1, all segments except b and c should be off.
- Byte 11111001 = F9H will display digit 1.

Interfacing Seven-Segment LEDs as an Output (4 of 4)

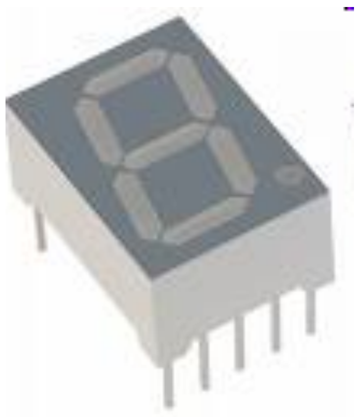


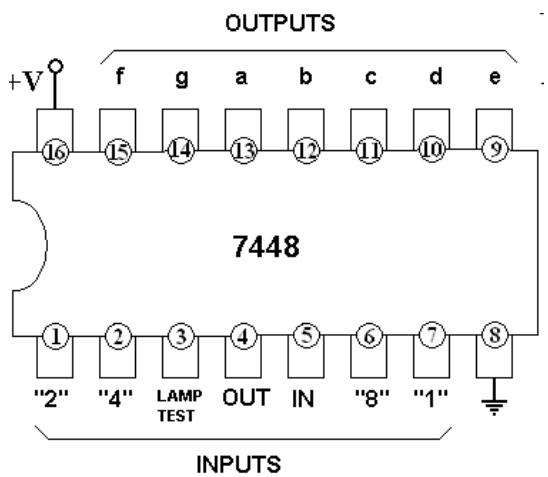
- In a common cathode seven-segment LED
  - All cathodes are connected together to ground and the anodes are connected to data lines
- Logic 1 turns on a segment.
- Example: To display digit 1, all segments except b and c should be off.
- Byte 00000110 = 06H will display digit 1.

Segment LEDS to PORTB and PORTC

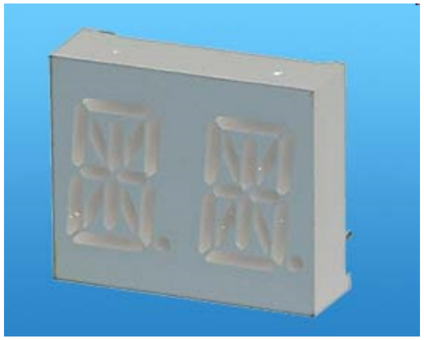


Seven-Segment Chips



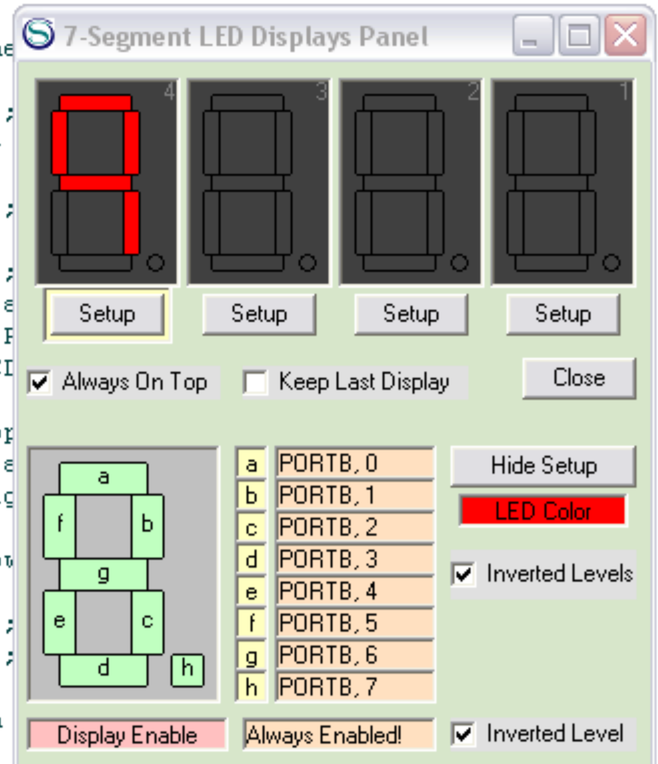


ALPHA/ NUMERIC C/A DISPLAY

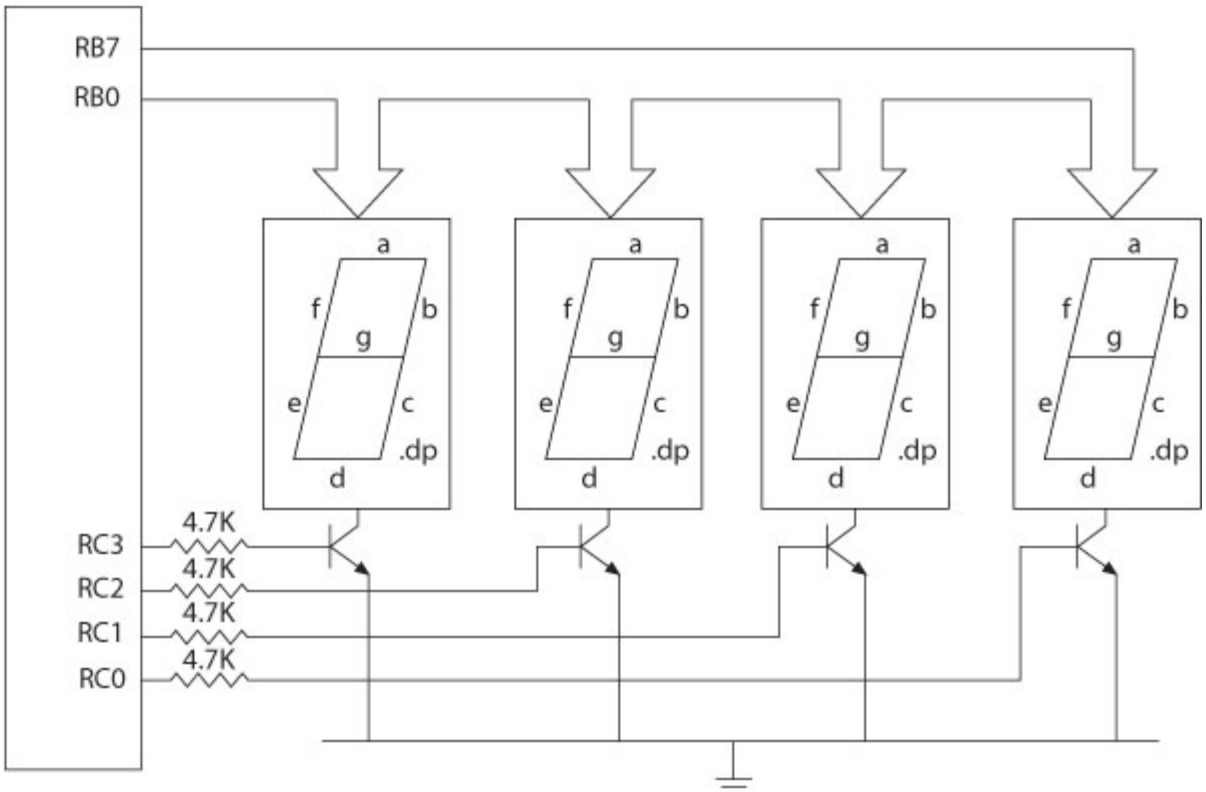


Sample Program

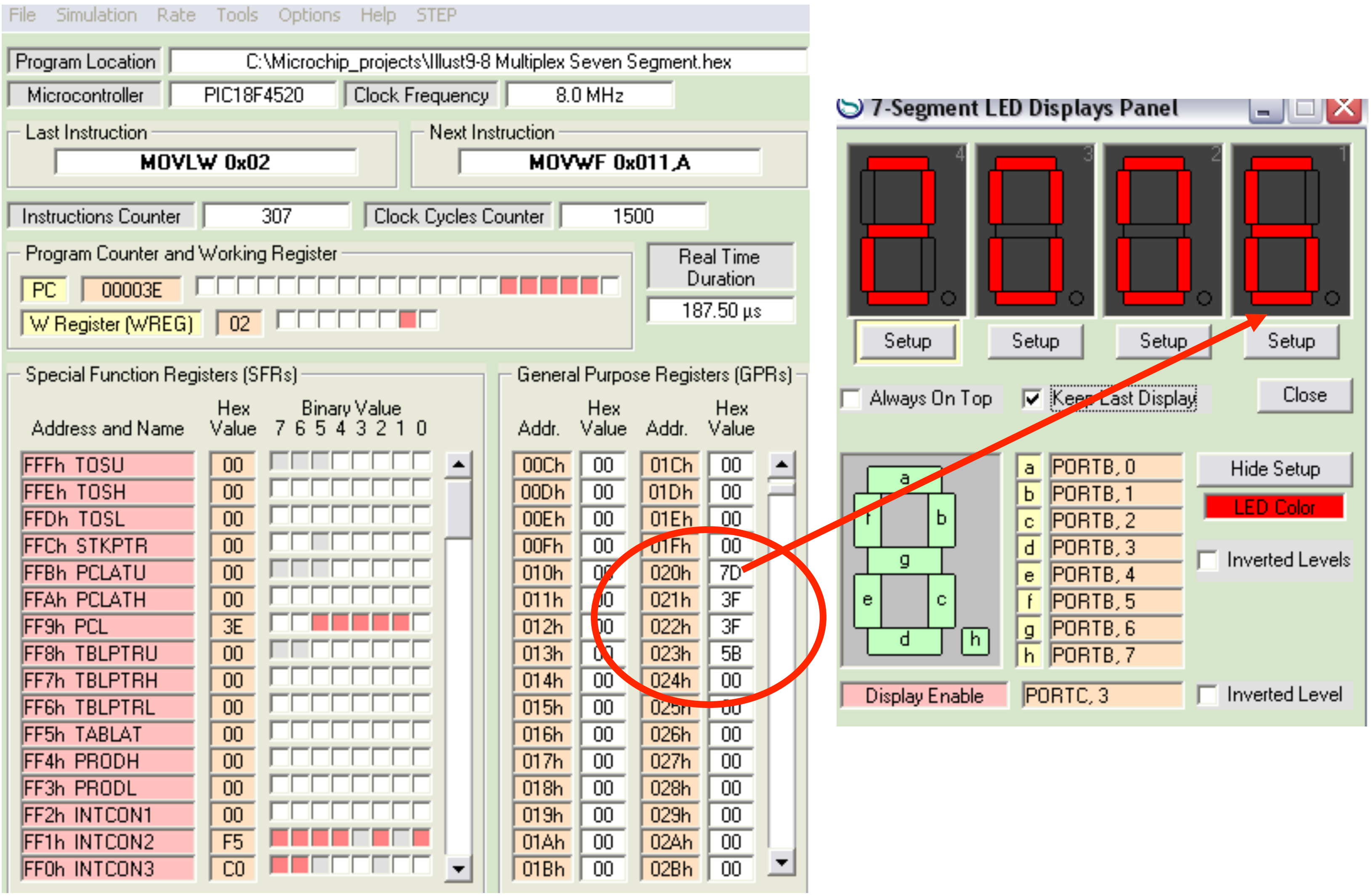
```
         Title "Ex9-4 BCD Digit Display"
         List p=18F452, f =inhx32
         #include <p18F452.inc>      ;This is a he

REG0     EQU       0x00                           ;
NUMBER   EQU       0x9                     ;<<<----
                   ORG                 00
                   GOTO      0x20                 ;

                   ORG                 0x20       ;
START:   CLRF      PORTB             ;Intial reading a
                   CLRF      TRISB             ;Set up P
                   MOVLW     NUMBER                 ;BCI
                   MOVWF     REG0, 0
                   MOVLW     UPPER CODEADDR    ;Copy upp
                   MOVWF     TBLPTRU           ;21-bit a
                   MOVLW     HIGH CODEADDR     ;Copy hig
                   MOVWF     TBLPTRH
                   MOVLW     LOW CODEADDR      ;Copy low
                   MOVWF     TBLPTRL
                   MOVF      REG0, 0, 0                   ;
                   ADDWF     TBLPTRL,1,0                  ;
                   TBLRD*
                   MOVFF     TABLAT, PORTB     ;Turn on

         ORG       0x40                        ;Store LED code starting at location 0040H

CODEADDR:          DB        0xc0, 0xF9, 0xA4, 0xB0, 0x99 ;LED code for digits 0 to 4
                   DB        0x92, 0x82, 0xF8, 0x80, 0x98 ;LED code for digits 5 to 9
```



Interfacing to Multiple 7-Segments



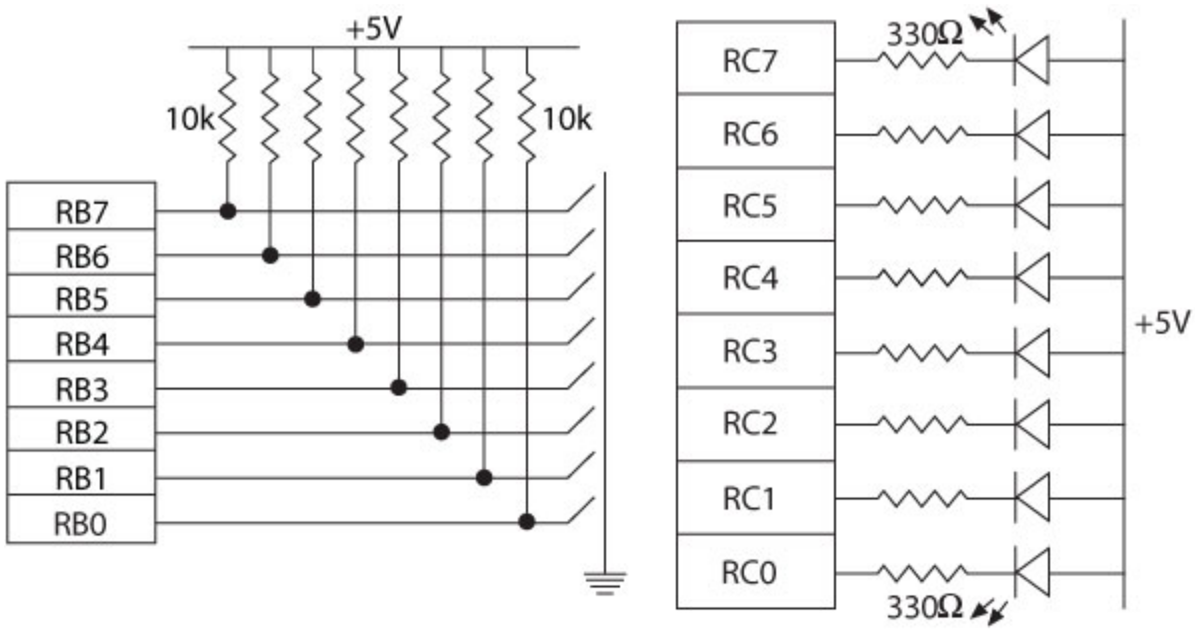
Using the Simulator



Interfacing Input Peripherals

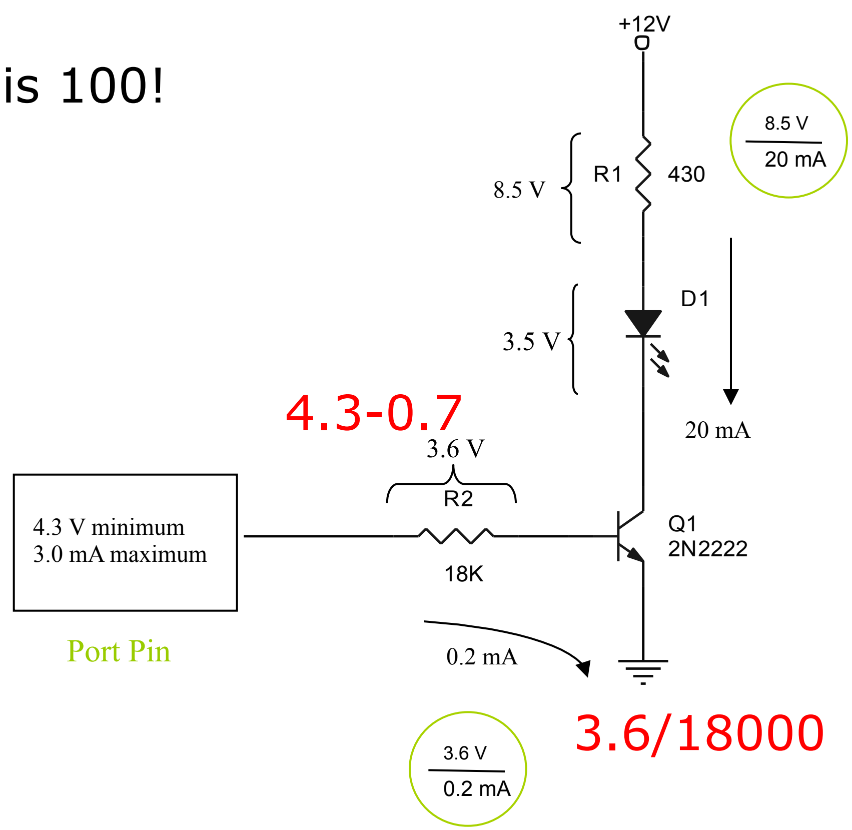
- Commonly used input peripherals in embedded systems are:
  - DIP switches, push-button keys, keyboards, and A/D converters.
- DIP switch: One side of the switch is tied high (to a power supply through a resistor called a pull-up resistor), and the other side is grounded. The logic level changes when the position is switched.
- Push-button key: The connection is the same as in the DIP switch except that contact is momentary.

Interfacing Dip Switches and Interfacing LEDs



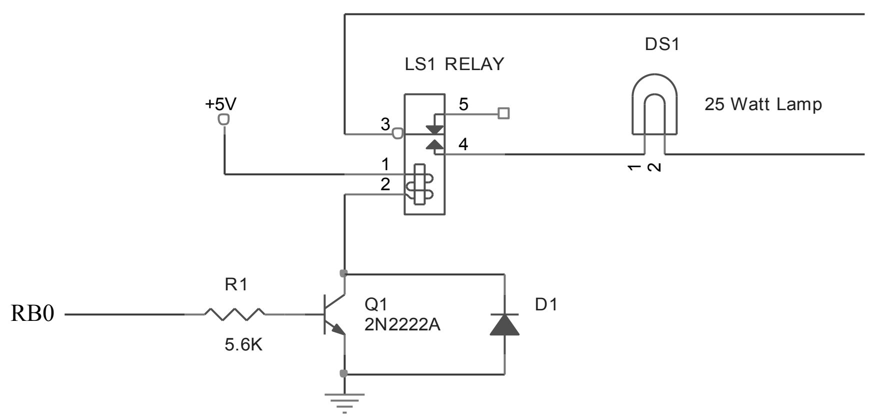
Driving an LED

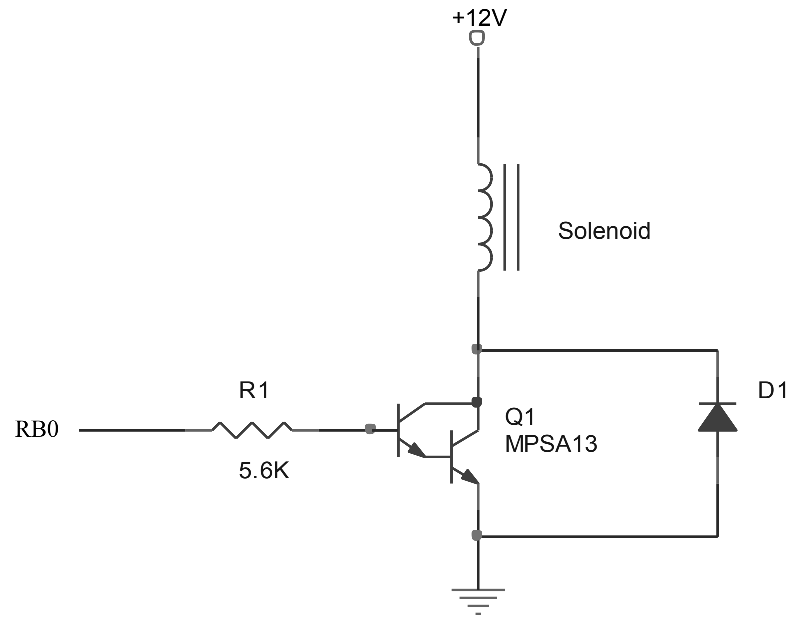
- Transistor inverter acting as the driver
- Assume LED requires 3.5 Volts and 20 mA
- 2N2222 is Current Amplifier → generates collector current (multiplies base-current)
- In this case the current gain is 100!
- Note Vbe=0.7



Driving a RELAY & SOLENOID

- Controlling appliances
- Driving solenoid



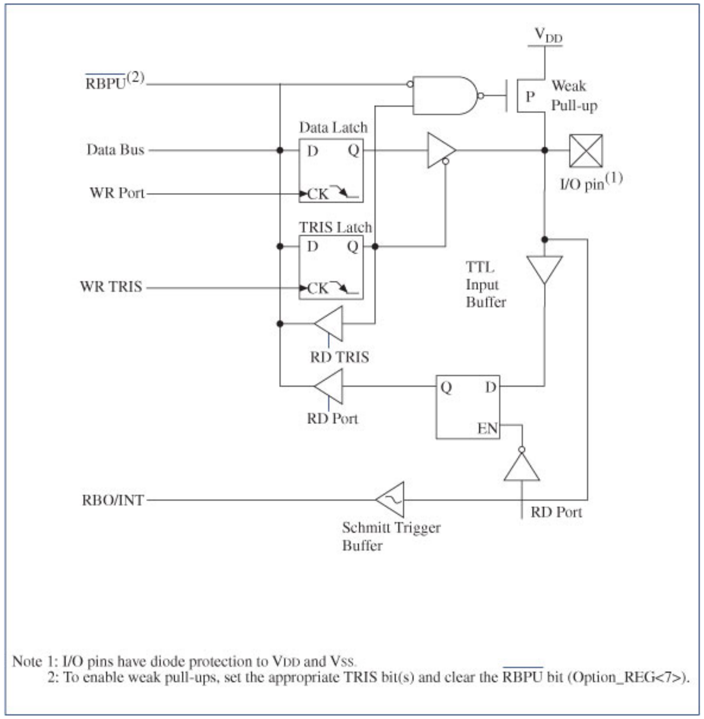


Example: Reading from an I/O Port

- The instruction: MOVF PORTB, W reads from PORTB.
- To execute the instruction, the MPU does the following:
  - Reads the instruction from memory
  - Places the address of PORTB (F81H) on the address bus of data memory
  - Selects PORTB
  - Asserts the RD signal and enables PORTB
  - Reads logic levels (1/0) of the switches and places on the data bus
  - Saves the reading in the WREG

Internal Pull-Up Resistor (1 of 2)

- The pull-up resistors are connected externally. However, PORTB can provide equivalent resistors internally through initialization.
- Turning off the internal FET is equivalent to providing a pull-up resistor.



Internal Pull-Up Resistor (2 of 2)

- Bit7 (RBPU) in the INTCON2 register enables or disables the pull-up resistor
  - Instruction to Enable Pull Up Resistors:
    BCF INTCON2 7, 0

C Code: INTCON2bit.RBPU = 0 // pull-ups on

B7	B6	B5	B4	B4	B3	B2	B1	BØ
$\overline{\text{RBPU}}$								

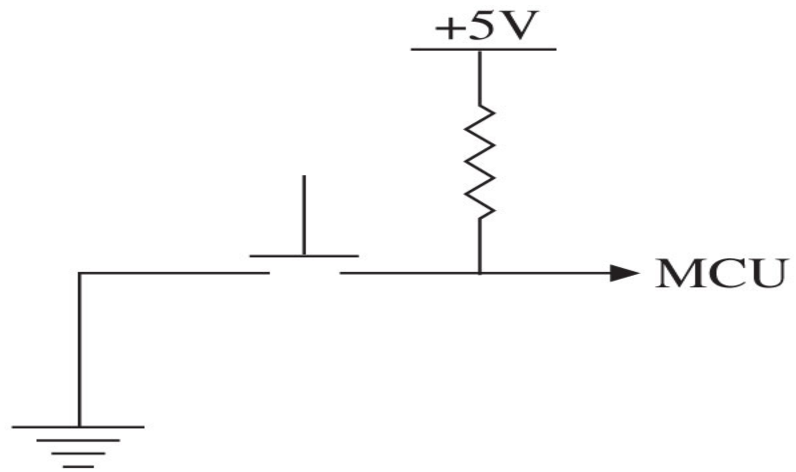
$\overline{\text{RBPU}}$ = PORTB pull-up resistor enable bit
0 = Pull-up resistors are enabled
1 = Pull-up resistors are disabled

Interfacing Push-Button Keys (1 of 2)

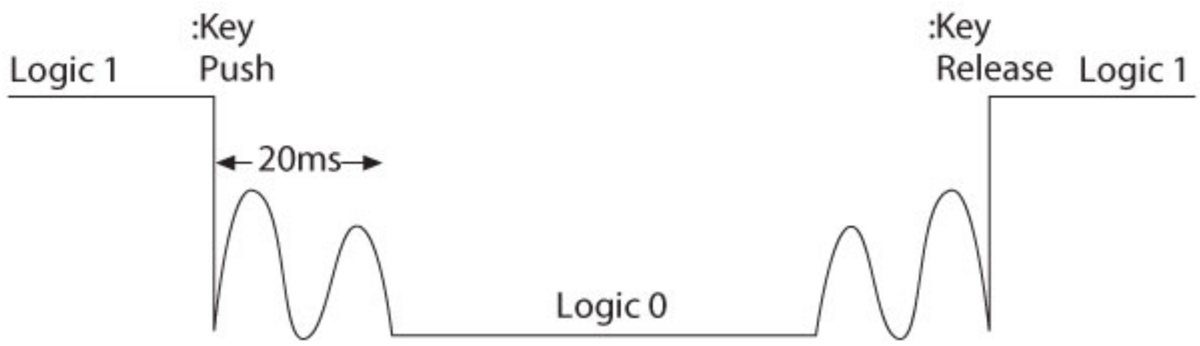
- Electrical connection of a push-button key is same as that of a DIP switch except that the connection is temporary when the key is pressed.
  - When a key is pressed (or released), mechanical metal contact bounces and can be read as multiple inputs.
  - The reading of one contact as multiple inputs can be eliminated by a key-debounce technique, using either hardware or software.

Interfacing Push-Button Keys (2 of 2)

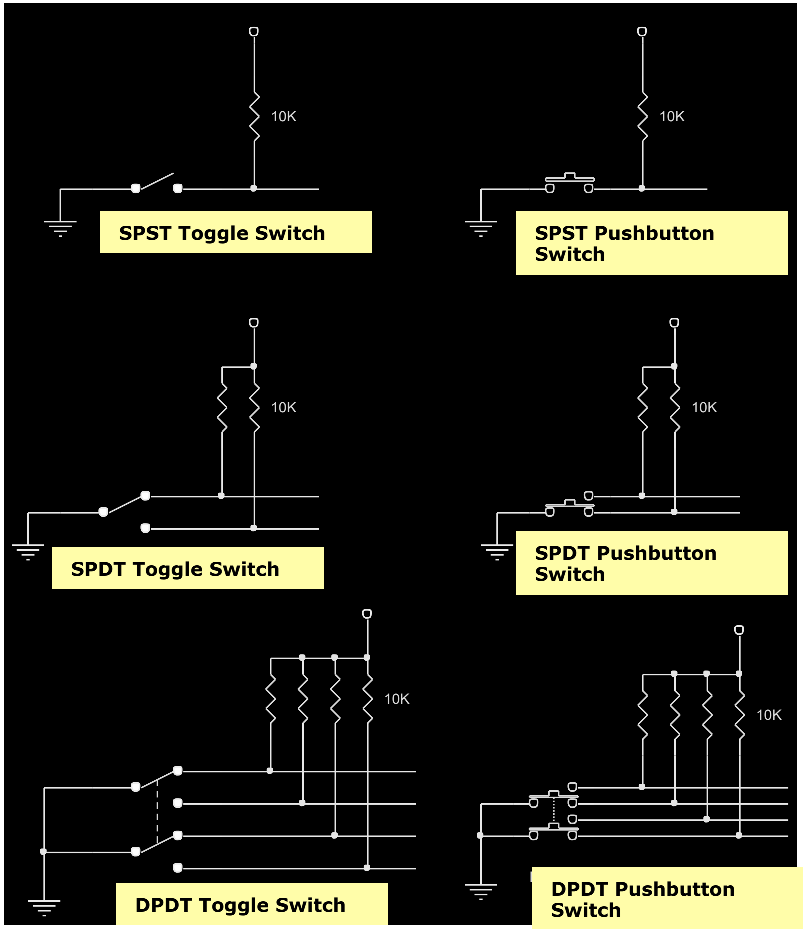
(a)



(b)



Various Switches



Key Debounce Techniques

- Hardware technique
  - Two circuits, based on the principles of generating a delay and switching the logic level at a certain threshold level.
  - Two NAND gates connected back to back, equivalent of a S-R latch. The output of the S-R latch is a pulse without a bounce.
  - An integrated circuit (MAX 6816) that bounces the key internally and provides a steady output.

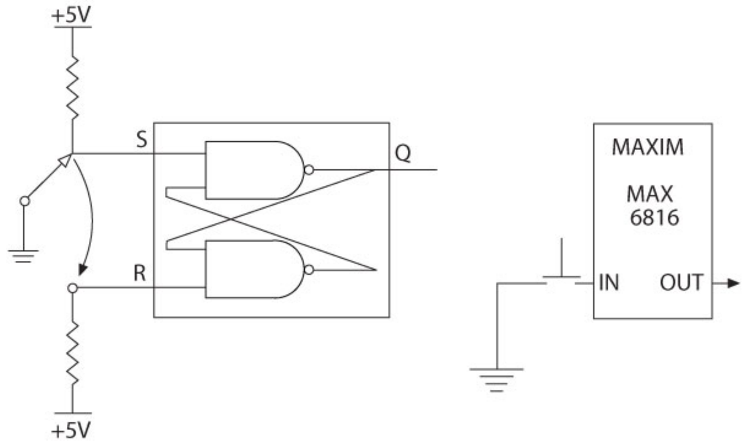
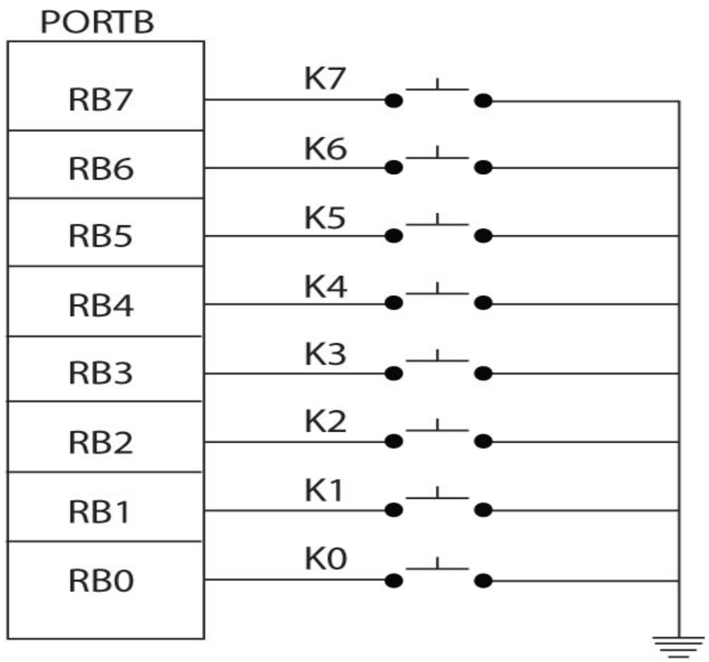


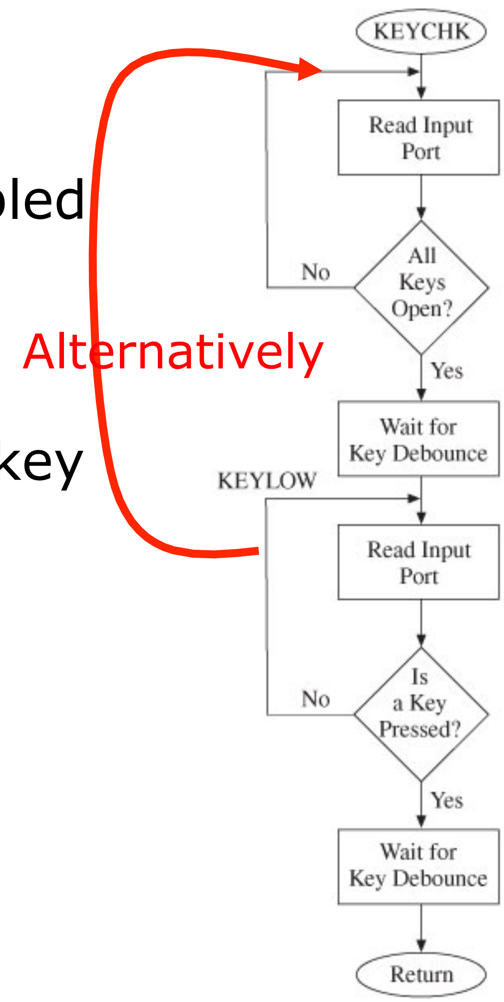
Illustration:
Interfacing Push-Button Keys (1 of 6)

- Problem statement
  - A bank of push-button keys are connected as inputs to PORTB.
  - The pull-up resistors are internal to PORTB.
  - Write a program to recognize a key pressed, debounce the key, and identify its location in the key bank with numbers from 0 to 7.

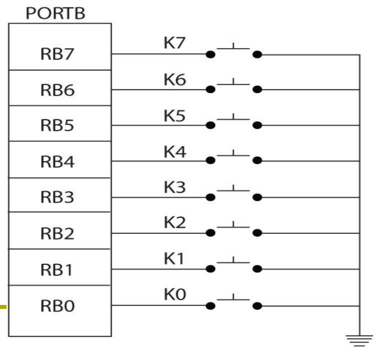


Interfacing Push-Button Keys (3 of 6)

- Hardware
  - PORTB should be set up as input port
  - Internal pull-up resistors should be enabled
- Software
  - Checking a key closure Debouncing the key
  - Encoding the key



Interfacing Push-Button Keys - Software Debounding



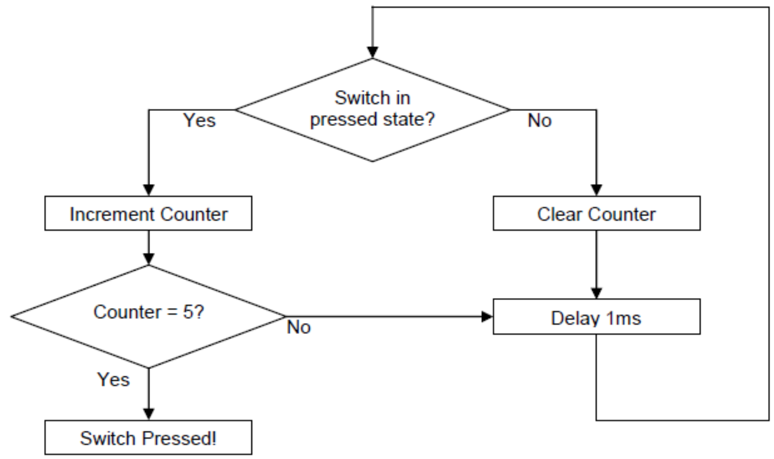
- Checking a key closure
  - When a key is open, the logic level is one (assuming pull-ups are enabled) and when it is closed, the logic level is zero.
  - When all keys are open, the reading will be 0xFF, and when a key is closed, the reading will be less than 0xFF.
    - Therefore, any reading less than FFH indicates a key closure.
    - This will be the first read!
- Debouncing the key
  - Software technique
    - Wait for 20 ms.
    - Read the port again.
    - If the reading is still less than FFH, it indicates that a key is pressed.
- Encoding the key
  - Key closure can be identified by rotating the reading right and looking for ‘No Carry’ and counting the rotations

Software Debouncing – Used for Active LOW!

```
// >>> Don't forget the #include <delays.h> statement <<<
// ********************** Switch *****************************
// to use this function, make sure that it is invoked as follows
//
//     Switch( 0x04 )   ← switch on bit 2
//
//          or
//
//        Switch( 0x40 )   ← switch on bit 6
//
//          or
//
//        Switch( 0x03 )   ← switches     bits 0 and 1
//
// ******************** CONSTANTS ****************************
#define KEYPORT PORTA                // change to match the actual port
#define DELAY 15                     // change as needed for time delay – 15 msec.
void Switch( char bit )
{
        do                                        // wait for release
        {
                while ( ( KEYPORT & bit ) != bit );
                Delay1KTCYx(DELAY);
        }
        while( ( KEYPORT & bit ) != bit );
        do                                        // wait for press
        {
                while ( ( KEYPORT & bit ) == bit );
                Delay1KTCYx(DELAY);
        }
        while( ( KEYPORT & bit ) == bit );
}
```

Switch(0x22)
0010 0010
Bits 1 or 5 is activated

Software Debouncing – Used for Active LOW! Another Example



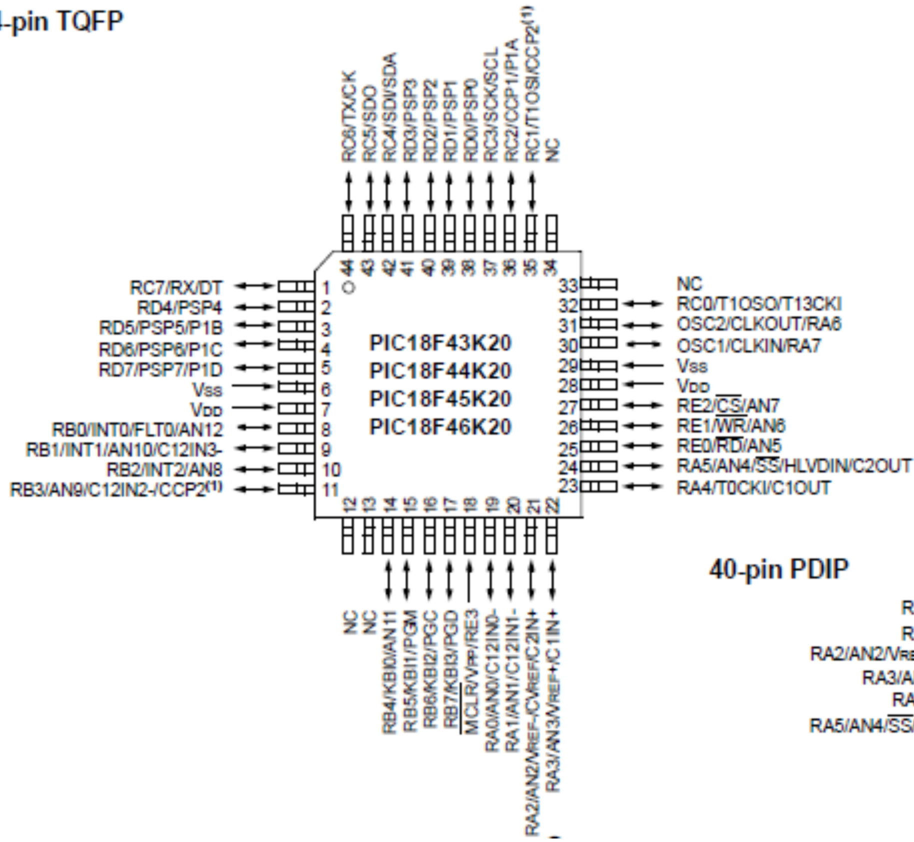

```
void main (void)
{
    unsigned char Switch_Count = 0;

    LED_Display = 1;              // initialize

    TRISD = 0b00000000;            // PORTD bits 7:0 are all outputs (0)

    INTCON2bits.RBPU = 0;          // enable PORTB internal pullups
    WPUBbits.WPUB0 = 1;            // enable pull up on RB0
    ANSELH = 0x00;                // AN8-12 are digital inputs (AN12 on RB0)
    TRISBbits.TRISB0 = 1;         // PORTB bit 0 (connected to switch) is input (1)

    while (1)
    {
        LATD = LED_Display;       // output LED_Display value to PORTD LEDs

        LED_Display <<= 1;        // rotate display by 1

        if (LED_Display == 0)
            LED_Display = 1;      // rotated bit out, so set bit 0

        while (Switch_Pin != 1);// wait for switch to be released

        Switch_Count = 5;
        do
        { // monitor switch input for 5 lows in a row to debounce
            if (Switch_Pin == 0)
            { // pressed state detected
                Switch_Count++;
            }
            else
            {
                Switch_Count = 0;
            }
            Delay10TCYx(25);      // delay 250 cycles or 1ms.
        } while (Switch_Count < DetectsInARow);
```

#define Switch_Pin PORTBbits.RB0

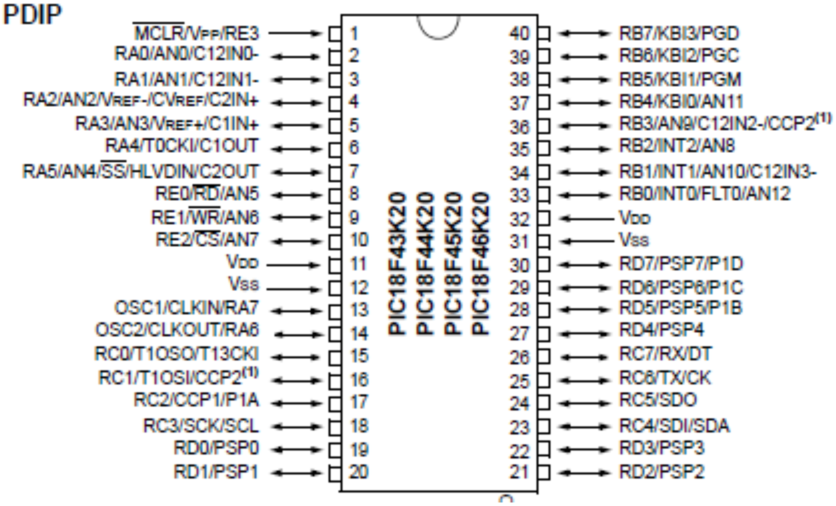
The demo board switch is connected to I/O pin RB0, which is normally pulled up to VDD internally. **When the switch is pressed, it pulls RB0 to ground** (low state).

PIC18F46K20 Pin Diagram

44-pin TQFP



40-pin PDIP



Digital Input Port

```
INTCON2bits.RBPU = 0;        // enable PORTB internal pullups   Interrupt Control
WPUBbits.WPUB0 = 1;          // enable pull up on RB0   Weak Pull Up
ANSELH = 0x00;              // AN8-12 are digital inputs (AN12 on RB0)   Analog Sel
TRISBbits.TRISB0 = 1;       // PORTB bit 0 (connected to switch) is input (1)
```

REGISTER 10-3: ANSELH: ANALOG SELECT REGISTER 2

U-0	U-0	U-0	R/W-1(1)	R/W-1(1)	R/W-1(1)	R/W-1(1)	R/W-1(1)
—	—	—	ANS12	ANS11	ANS10	ANS9	ANS8
bit 7							bit 0

Legend:

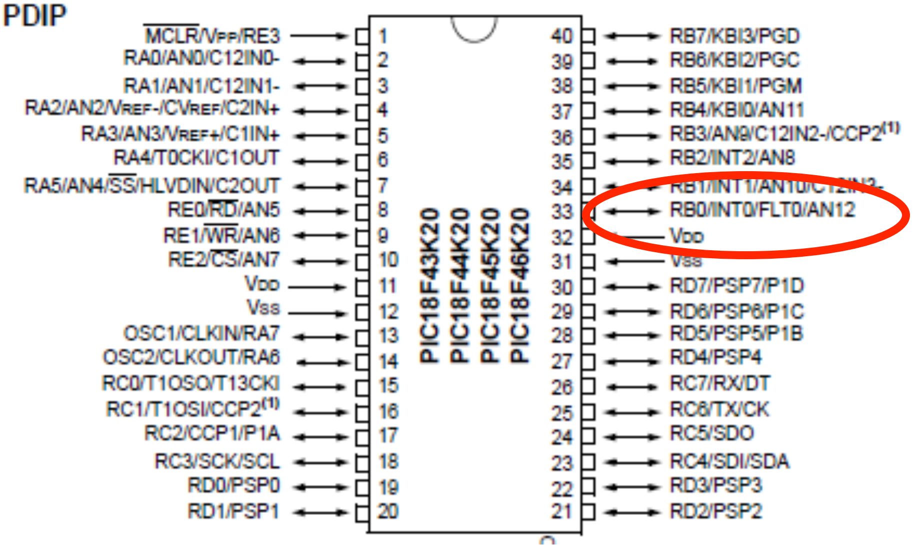
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7-5 **Unimplemented:** Read as '0'

bit 4 **ANS12:** RB0 Analog Select Control bit
1 = Digital input buffer of RB0 is disabled
0 = Digital input buffer of RB0 is enabled

PIC18F46K20

40-pin PDIP



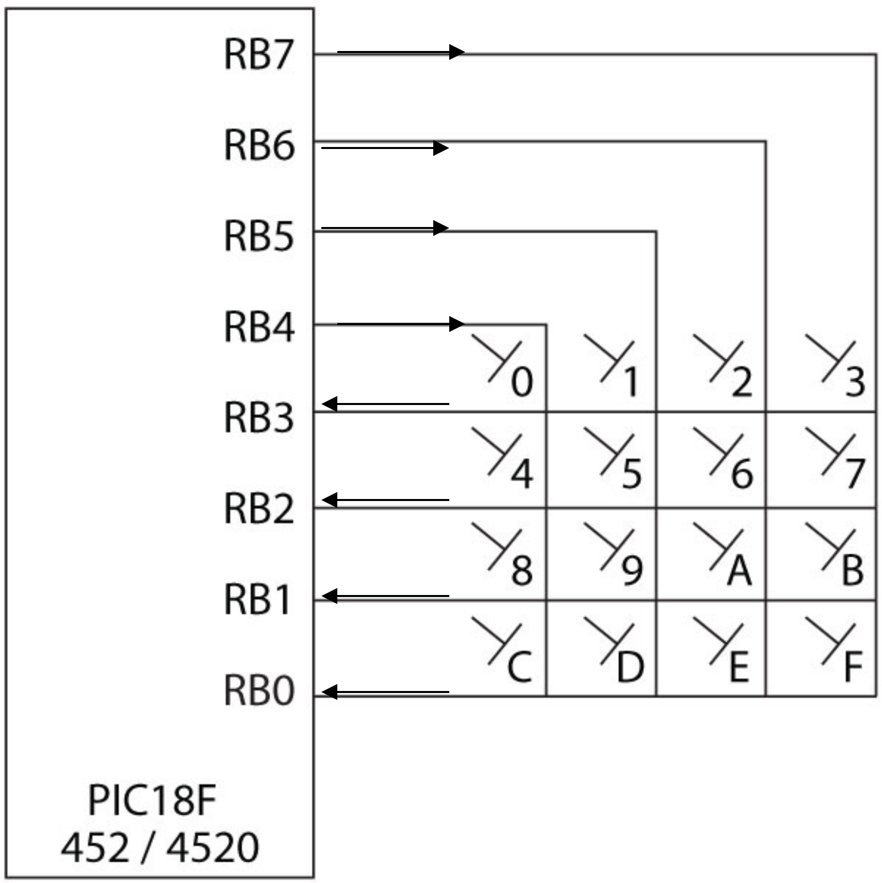
Pins are configured as
analog or digital in the SFRs
ANSEL and ANSELH

Class Exercise

- Find the following bits in the Data Sheet:
  - http://ww1.microchip.com/downloads/en/DeviceDoc/41303G.pdf

```
INTCON2bits.RBPU = 0;          // enable PORTB internal pullups
WPUBbits.WPUB0 = 1;            // enable pull up on RB0
ANSELH = 0x00;               // AN8-12 are digital inputs (AN12 on RB0)
TRISBbits.TRISB0 = 1;        // PORTB bit 0 (connected to switch) is input (1)
```

Interfacing a Matrix Keyboard

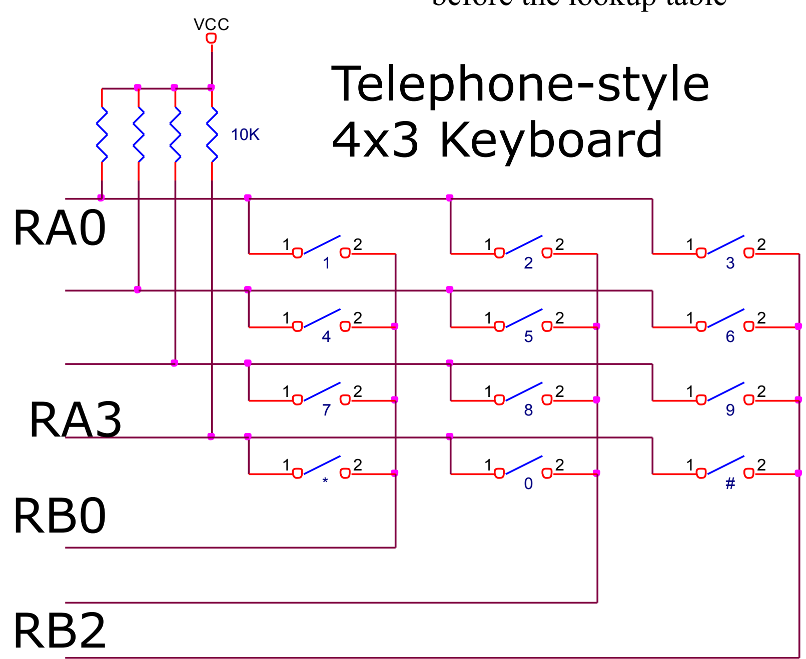


1	2	3
4	5	6
7	8	9
10	0	11

Actual Keyboard

0	4	8
1	5	9
2	6	10
3	7	11

Keyboard as seen by software before the lookup table



Interfacing a Matrix Keyboard

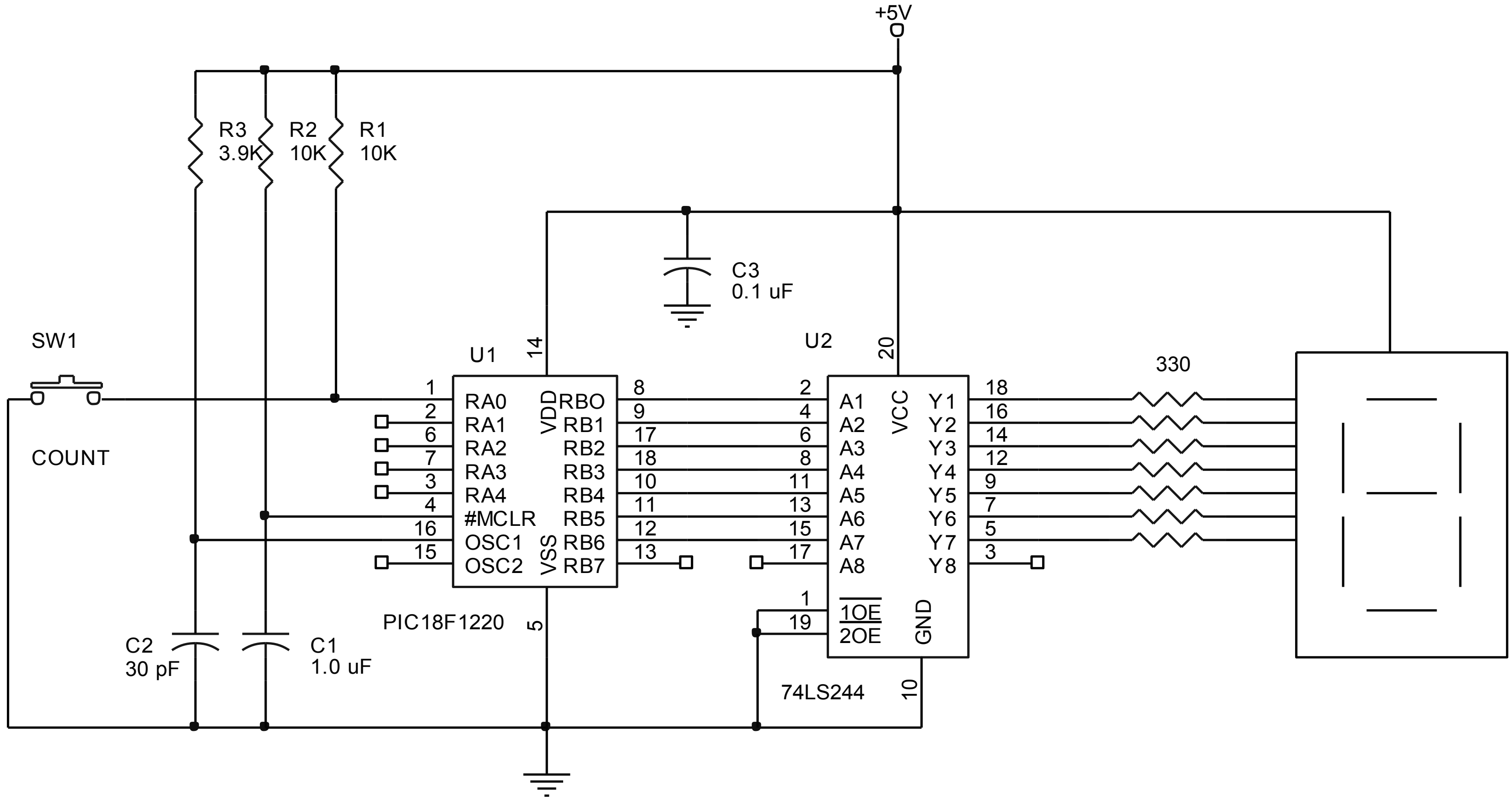
- Software
  - To recognize and encode the key pressed, the program should:
    - Ground all the columns by sending zeros.
    - Check each key in a row for logic zero.
    - Ground one column at a time and check all the rows in that column.
    - Once a key is identified, it is encoded based on its position in the column.

Matrix Keyboard Software

```
//
// key codes for a telephone style keypad
//          stored as static constants in the program memory
//
rom near char lookupKey[] =
{
          1, 4, 7, 10,                    // left column
          2, 5, 8, 0,                     // middle column
          3, 6, 9, 11                     // right column
};
//
// uses function Switch
//
unsigned char Key(void)
{
          #define MASK 0x0f                        // set mask
          #define ROWS 4                                    // set number of rows
          char a;
          char keyCode;
          PORTB = keyCode = 0;                     //clear Port B & keyCode
          Switch( MASK );                                   // de-bounce and wait for any key
          PORTB = 0xFE;                            // select a leftmost column

          while ( ( PORTA & MASK ) == MASK )  // while no key is found
          {
                    PORTB = (PORTB << 1) | 1;      // get next column
                    keyCode += ROWS;                        // add rows to keycode
          }
          for ( a = 1; a != 0; a <<= 1)
          {                                                           // find row
                    if ( ( PORTA & a ) == 0 )
                              break;
                    keyCode++;
          }
          return lookupKey[keyCode];               // lookup correct key code
}
```

1	2	3
4	5	6
7	8	9
10	0	11

7-Segment Interface

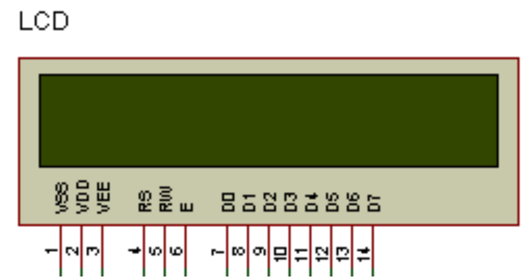


```
// **************** program memory data ************************
rom near char look7[] = // 7-segment lookup table
{
	0x40,				// 0		active low signals
	0x79,				// 1	x g f e  d c b a
	0x24,				// 2
	0x30,				// 3
	0x19,				// 4
	0x12,				// 5
	0x02,				// 6
	0x78,				// 7
	0x00,				// 8
	0x10				// 9
};
// **************** data memory data ***************************
int count;
#pragma code
// ****************de-bounce  functions **********************************
void Switch( char bit )
{
	do				// wait for release
	{
		while ( ( PORTA & bit ) != bit );
		Delay1KTCYx(30);		// 15 ms delay

	}while( ( PORTA & bit ) != bit );
	do				// wait for press
	{
		while ( ( PORTA & bit ) == bit );
		Delay1KTCYx(30);

	}while( ( PORTA & bit ) == bit );
}
// **************** main program *********************************
void main (void)
{
	ADCON1 = 0x7F;				// Ports A and B are digital
	TRISA = 1;			// Port A, bit 0 is input
	TRISB = 0;			// Port B is output
	count = 0;			// start count at zero
	while		( 1 )			// main loop
	{
		PORTB = look7[count];		// display number
		Switch( 1 );			// wait for pushbutton
		count++;
		if ( count >= 10 )
			count = 0;
	}
}
```

7-Segment
Control Software with
De-bounce →
Each time the input is
Pressed the number
Shown by the 7-segemnt
Increments!

Interfacing LCD (Liquid Crystal Display)

- Problem statement
  - Interface a 2-line x 20 character LCD module with the built-in HD44780 controller to I/O ports of the PIC18 microcontroller
- Multi-LCDs refer to LCDs with different interfaces



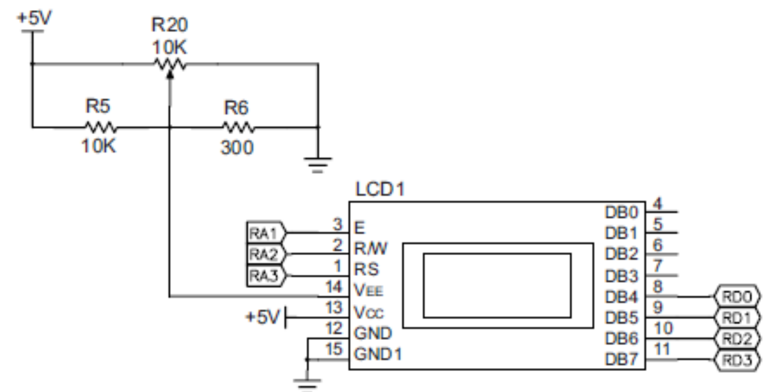
Converting to ASCII

- The LCD can represent characters in ASCII
- For example number 0x08 → must be converted to 0x38
- To perform this:
  - If W=0x08 then ASCII=XORLW 0x30→W=38

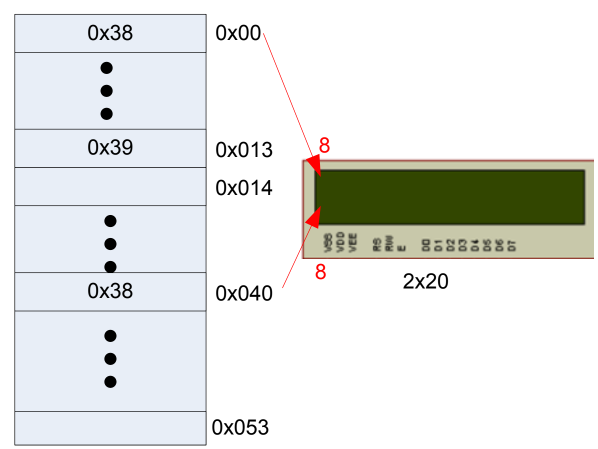
Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Interfacing LCD

- Hardware
  - 20 x 2-line LCD displays (two lines with 20 characters per line)
  - LCD has a display Data RAM (registers) that stores data in 8-bit character code.
  - Each register in Data RAM has its own address that corresponds to its position on the line.
    - The address range for Line 1 is 00 to 13H and Line 2 is 40H to 53H.

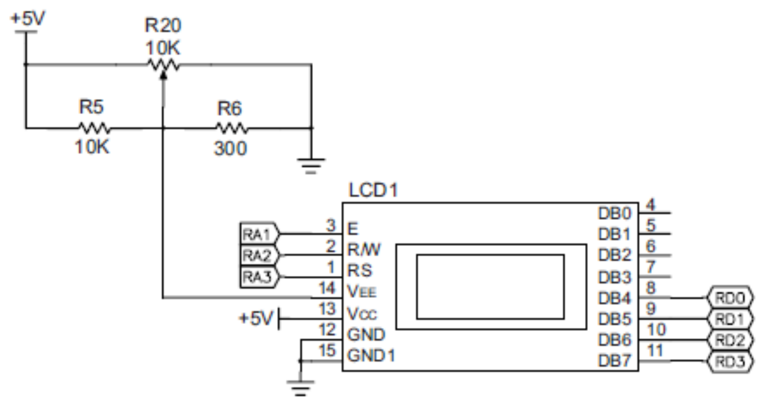


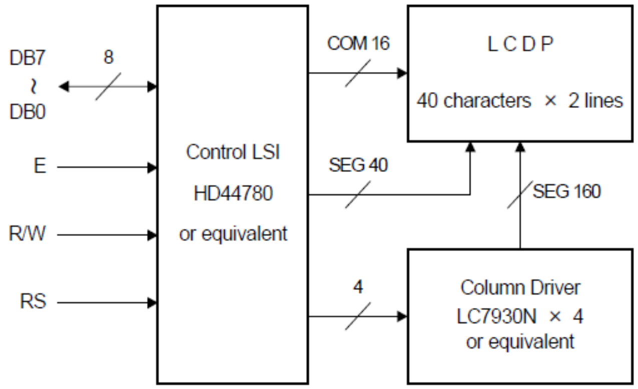
PICDEMO



Interfacing LCD

- Driver HD77480
  - Three control signals:
    - RS – Register Select (RA3)
    - R/W – Read/Write (RA2)
    - E – Enable (RA1)
  - Three power connections
    - Power, ground, and the variable register to control the brightness





Interfacing LCD

- Can be interfaced either in the 8-bit mode or the 4-bit mode
  - In the 8-bit mode, all eight data lines are connected for data transfer
  - In the 4-bit mode, only four data lines (DB7-DB4 or DB3-DB0) are connected and two transfers per character (or instruction) are needed
- Driver (HD77480) has two 8-bit internal registers
  - Instruction Register (IR) to write instructions to set up LCD
  - Data Register (DR) to write data (ASCII characters)

IR REGISTER

DR REGISTER

Command and Instruction set for LCD type HD44780

Command	Code										Description	Execution Time
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0		
Clear Display	0	0	0	0	0	0	0	0	0	1	Clears the display and returns the cursor to the home position (address 0).	82μs~1.64ms
Return Home	0	0	0	0	0	0	0	0	1	*	Returns the cursor to the home position (address 0). Also returns a shifted display to the home position. DD RAM contents remain unchanged.	40μs~1.64ms
Entry Mode Set	0	0	0	0	0	0	0	1	I/D	S	Sets the cursor move direction and enables/disables the display.	40μs
Display ON/OFF Control	0	0	0	0	0	0	1	D	C	B	Turns the display ON/OFF (D), or the cursor ON/OFF (C), and blink of the character at the cursor position (B).	40μs
Cursor & Display Shift	0	0	0	0	0	1	S/C	R/L	*	*	Moves the cursor and shifts the display without changing the DD RAM contents.	40μs
Function Set	0	0	0	0	1	DL	N$	F	*	#	Sets the data width (DL), the number of lines in the display (L), and the character font (F).	40μs
Set CG RAM Address	0	0	0	1	A_{CG}						Sets the CG RAM address. CG RAM data can be read or altered after making this setting.	40μs
Set DD RAM Address	0	0	1	A_{DD}							Sets the DD RAM address. Data may be written or read after making this setting.	40μs
Read Busy Flag & Address	0	1	BF	AC							Reads the BUSY flag (BF) indicating that an internal operation is being performed and reads the address counter contents.	1μs
Write Data to CG or DD RAM	1	0	Write Data								Writes data into DD RAM or CG RAM.	46μs
Read Data from CG or DD RAM	1	1	Read Data								Reads data from DD RAM or CG RAM.	46μs

I/D = 1: Increment I/D = 0: Decrement
S = 1: Accompanies display shift.
S/C= 1: Display shift S/C = 0: cursor move
R/L= 1: Shift to the right. R/L= 0: Shift to the left.
DL = 1: 8 bits DL = 0: 4 bits
N = 1: 2 lines N = 0: 1 line
F = 1: 5x10 dots F = 0: 5 x 7 dots
BF = 1: Busy BF = 0: Can accept data
 # Set to 1 on 24x4 modules
$ With KS0072 is Address Mode.

DD RAM: Display data RAM
CG RAM: Character generator RAM
A_{CG}: CG RAM Address
A_{DD}: DD RAM Address Corresponds to cursor address.
AC: Address counter Used for both DD and CG RAM address.

Execution times are typical. If transfers are timed by software and the busy flag is not used, add 10% to the above times.

Interfacing LCD

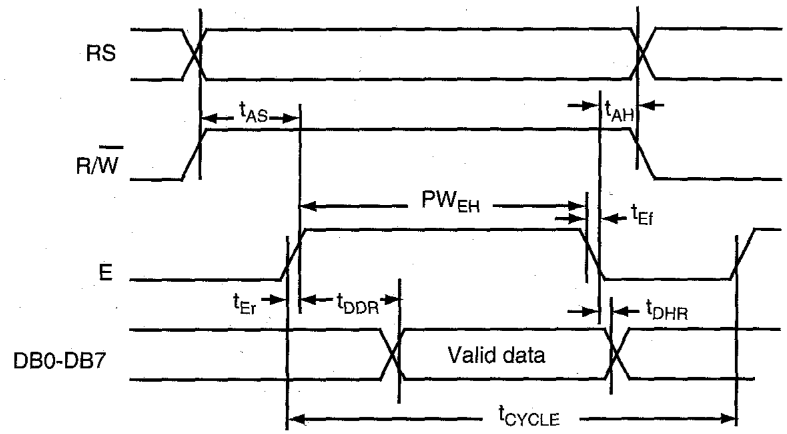
- LCD Operation
  - When the MPU writes an instruction to IR or data to DR, the controller:
    - Sets the data line DB7 high as a flag indicating that the controller is busy completing the operation
    - Sets the data line DB7 low after the completion of the operation
  - The MPU should always check whether DB7 is low before sending an instruction or a data byte
  - After the power up, DB7 cannot be checked for the first two initialization instructions.

Interfacing LCD

- Writing to or reading from LCD
- The MPU:
  - Asserts RS low to select IR
  - Reads from LCD by asserting the R/W signal high
  - Asserts the E signal high and then low (toggles) to latch a data byte or an instruction

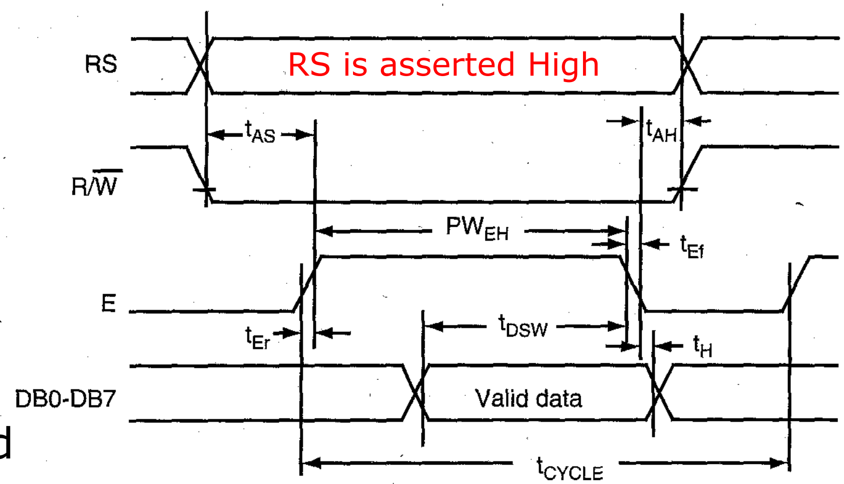
  - Asserts RS high to select DR
  - Writes into LCD by asserting the R/W signal low
  - Asserts the E signal high and then low (toggles) to latch a data byte or an instruction

HD44780 Bus Timing



Read timing diagram

Symbol	Meaning	Min	Typ	Max.	Unit
t_{CYCLE}	Enable cycle time	1000	-	-	ns
PW_{EH}	Enable pulse width (high level)	450	-	-	ns
t_{Er}, t_{Ef}	Enable rise and decay time	-	-	25	ns
t_{AS}	Address setup time, RS, R/$\overline{W}$, E	60	-	-	ns
t_{DDR}	Data delay time	-	-	360	ns
t_{DSW}	Data setup time	195	-	-	ns
t_H	Data hold time (write)	10	-	-	ns
t_{DHR}	Data hold time (read)	5	-	-	ns
t_{AH}	Address hold time	20	-	-	ns



Write timing diagram

- Asserts RS high to select DR
- Writes into LCD by asserting the R/W signal low
- Asserts the E signal high and then low (toggles) to latch a data byte or an instruction

Interfacing LCD (Write)

- Software
  - To write into the LCD, the program should:
    - Send the initial instructions (commands) before it can check DB7 to set up the LCD in the 4-bit or the 8-bit mode.
    - Check DB7 and continue to check until it goes low.
    - Write instructions to IR to set up the LCD parameters such as the number of display lines and cursor status.
    - Write data to display a message.

Resetting LCD

- In 4-bit mode the data is sent in nibbles
  - First we send the higher nibble and then the lower nibble.
- To enable the 4-bit mode of LCD, we need to follow special sequence of initialization that tells the LCD controller that user has selected 4-bit mode of operation:
  - Wait for about 20mS
  - Send the first init value (0x30)
  - Wait for about 10mS
  - Send second init value (0x30)
  - Wait for about 1mS
  - Send third init value (0x30)
  - Wait for 1mS
  - Select bus width (0x30 - for 8-bit and 0x20 for 4-bit
  - Wait for 1mS

http://www.8051projects.net/lcd-interfacing/commands.php http://www.youtube.com/watch?v=tTym5apZwCE

http://video.google.com/videoplay?docid=7437543675646211278#

Organic LED

- Organic light-emitting diodes - OLEDs - emit light when a current flows through them
- Unlike conventional LEDs, OLEDs are made from layers of plastic and other organic (carbon-based) materials
  - Very flexible!
- Applications: displays in MP3 players and phones
- Advantages:
  - cheaper than the techniques required to make conventional LEDs.
  - inherently thin
  - can be made on flexible plastic substrates
  - all colors, and multi-colors, are possible
- Disadvantages
  - incredibly sensitive to moisture which leads to short life - glass blocks all moisture, so displays made on a glass substrate and covered by a second glass sheet can have a long life, particularly if the edges are hermetically sealed

http://en.wikipedia.org/wiki/Organic_LED

Organic LED

- OLEDs are generally made of several layers
- A typical stack (variations are possible):
  - Anode
  - Electron donor
  - Electron transport
  - Emitter
  - Hole transport
  - Hole donor.
  - Cathode

References

- http://home.iae.nl/users/pouweha/lcd/lcd0.shtml
- Huang