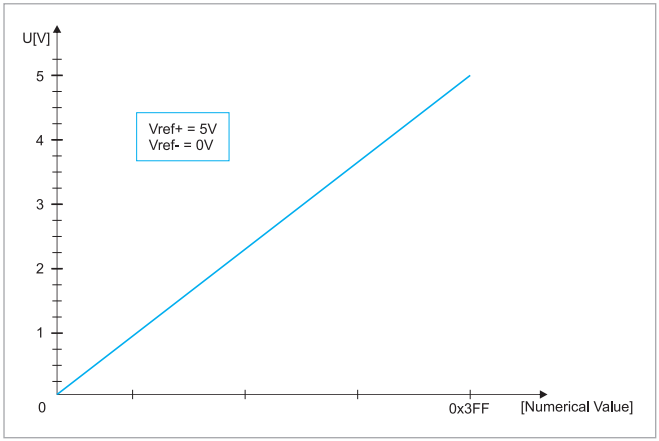
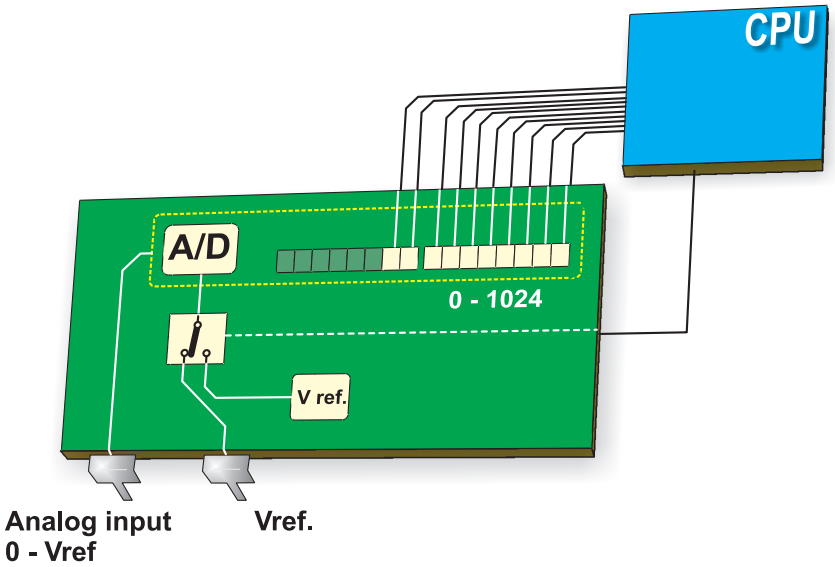
A/D CONVERTER

External signals are usually fundamentally different from those the microcontroller recognizes (0V and 5V only) and therefore have to be converted into recognizable values. An analog to digital converter is an electronic circuit which converts continuous signals to discrete digital numbers. In other words, this circuit converts an analogue value into a binary number and forwards it to the CPU for further processing. This module is thus used for input pin voltage (analogue value) measurement.



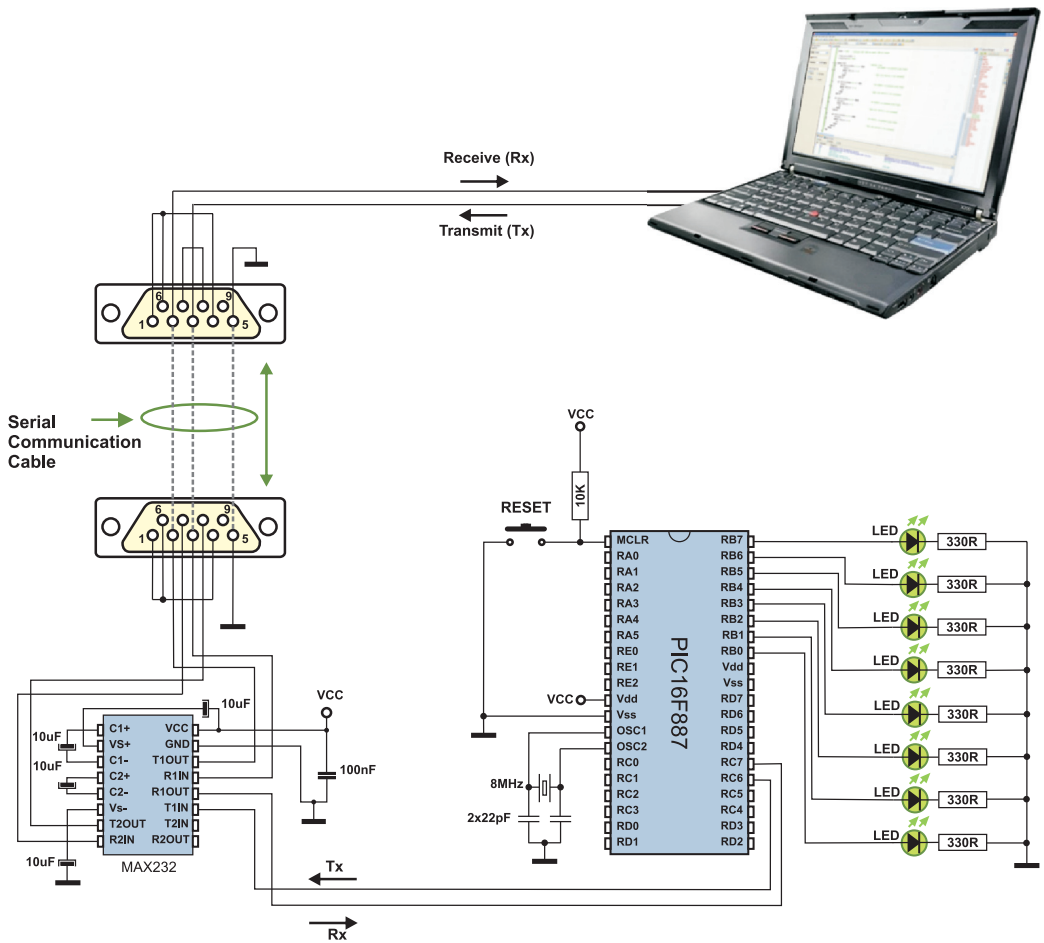
The result of measurement is a number (digital value) used and processed later in the program.



EXAMPLE 11

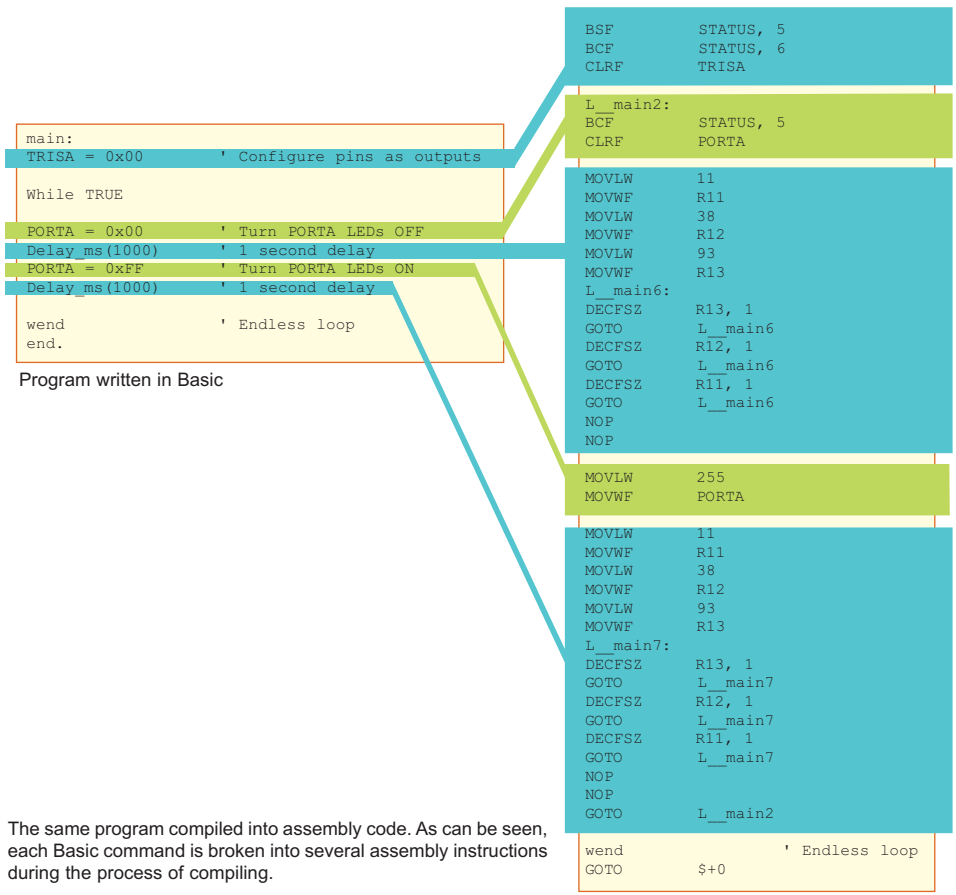
RS232 serial communication

This example illustrates the use of the microcontroller's EUSART module. Connection between the microcontroller and a PC is established in compliance with the RS232 communication standard. The program works as follows. Every byte received via serial communication is displayed using LED diodes connected to PORTB and is automatically sent back to the sender thereupon. The easiest way to test the program operation is by using a standard Windows program called *Hyper Terminal*.



ADVANTAGES OF HIGH-LEVEL PROGRAMMING LANGUAGES

If you have any experience in writing programs for PIC microcontrollers in assembly language, then you are probably familiar with the other side of the medal of RISC architecture - the lack of instructions. For example, there is no appropriate instruction for multiplying two numbers. Of course, there is a way to solve this issue owing to mathematics which enables you to perform complex operations by breaking them into a number of simple ones. Accordingly, multiplication can be easily substituted by successive addition (a x b = a + a + a + ... + a). And here we are, just at the beginning of a very long story... Still there is no reason to be worried about as far as you use one of the high-level programming languages, such as Basic, as the compiler will automatically find a solution to these and similar issues. Simply write a*b.



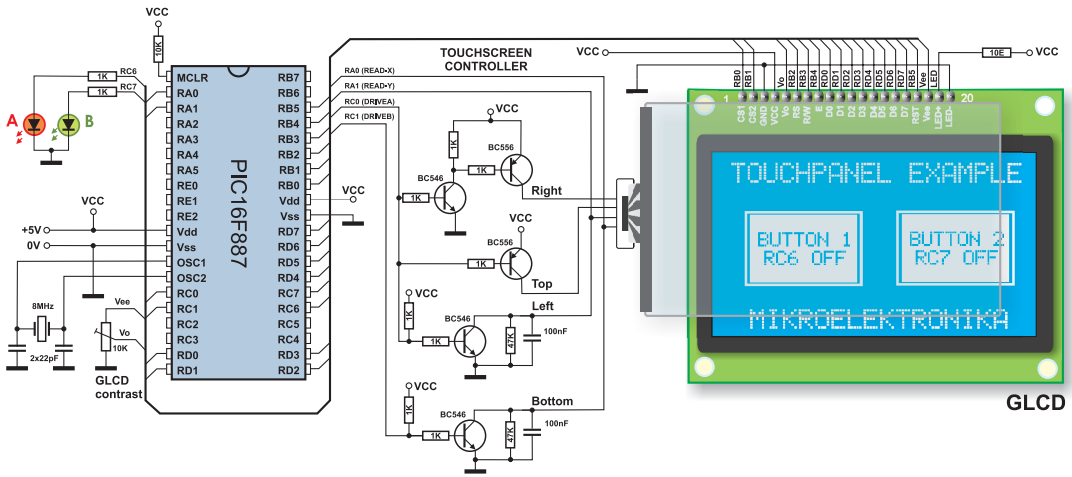
Program written in Basic

The same program compiled into assembly code. As can be seen, each Basic command is broken into several assembly instructions during the process of compiling.

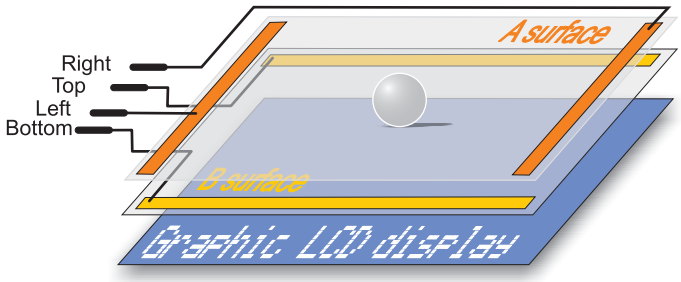
EXAMPLE 15

Use a touch panel

A touch panel is a thin, self-adhesive transparent panel placed over the screen of a graphic LCD. It is very sensitive to pressure so that even a soft touch causes some changes on the output signal. There are a few types of touch panel. The simplest one is a resistive touch panel.



It consists of two transparent rigid foils, forming a 'sandwich' structure, that have resistive layers on their inner sides. The resistance of these layers usually does not exceed 1K. The opposite sides of these foils have contacts available for use via a flat cable.

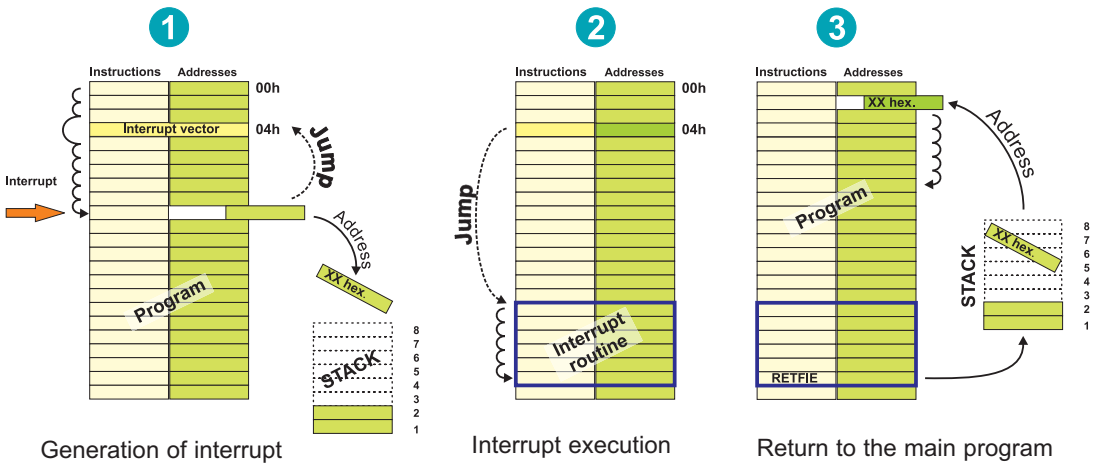


The process of determining coordinates of the point in which the touch panel is pressed can be broken into two steps. The first one is the determination of the X coordinate and the second one is the determination of the Y coordinate of the point.

INTERRUPT SYSTEM

The first thing to be done by the microcontroller, when an interrupt request arrives, is to execute the current instruction, then to stop the regular program execution. The current program memory address is automatically pushed onto the stack and the default address (predefined by the manufacturer) is written to the program counter. The location from where the program proceeds with execution is called an interrupt vector. For the PIC16F887 microcontroller, the address is 0004h. As seen in figure below, the interrupt vector should be skipped during regular program execution.

A part of the program to be executed when an interrupt request arrives is called an interrupt routine (it is a subroutine in fact). The first instruction of the interrupt routine is located at the interrupt vector. How long will it take to execute the subroutine and what it will be like, depends on the skills of the programmer as well as on the interrupt source itself. Some microcontrollers have a couple of interrupt vectors (every interrupt request has its vector), whereas this microcontroller has only one. This is why the first part of every interrupt routine should be interrupt source detection. When the interrupt source is known and interrupt routine is executed, the microcontroller reaches the `RETFIE` instruction, pops the address from the stack and proceeds with program execution from where it left off.



Generation of interrupt — Interrupt execution — Return to the main program

MikroBasic recognizes an interrupt routine to be executed by means of the *interrupt* keyword. The interrupt routine should be written by the user.

Example

```
sub procedure interrupt    ' Interrupt routine
cnt = cnt + 1 ;        ' Interrupt causes variable cnt to be incremented by 1
end sub
```

Example 10

```
 'Header*******************************************************

program example_10              ' Program name

dim LCD_RS as sbit  at RB4_bit                ' Lcd module connections
    LCD_EN as sbit  at RB5_bit
    LCD_D4 as sbit  at RB0_bit
    LCD_D5 as sbit  at RB1_bit
    LCD_D6 as sbit  at RB2_bit
    LCD_D7 as sbit  at RB3_bit

    LCD_RS_Direction as sbit at TRISB4_bit
    LCD_EN_Direction as sbit at TRISB5_bit
    LCD_D4_Direction as sbit at TRISB0_bit
    LCD_D5_Direction as sbit at TRISB1_bit
    LCD_D6_Direction as sbit at TRISB2_bit
    LCD_D7_Direction as sbit at TRISB3_bit   ' End Lcd module connections

dim text as string [16]         ' Variable text is of string type
dim ch, adc_rd as word          ' Variables ch and adc_rd are of word type
dim tlong as longword           ' Variable tlong is of longword type

main:                           ' Start of program
  TRISB = 0                     ' All port PORTB pins are configured as outputs
  PORTB = 0xFF
  INTCON = 0                    ' All interrupts disabled
  ANSEL  = 0x04                 ' Pin RA2 is configured as an analog input
  TRISA  = 0x04
  ANSELH = 0                    ' Rest of pins is configured as digital
  Lcd_Init()                    ' LCD display initialization
  Lcd_Cmd(_LCD_CURSOR_OFF)      ' LCD command (cursor off)
  Lcd_Cmd(_LCD_CLEAR)           ' LCD command (clear LCD)

  text = "mikroElektronika"     ' Define the first message
  Lcd_Out(1,1,text)             ' Write the first message in the first line
  text = "LCD example"          ' Define the second message
  Lcd_Out(2,1,text)             ' Write the second message in the second line

  ADCON1      = 0x80            ' A/D voltage reference is VCC
  TRISA       = 0xFF            ' All PORTA pins are configured as inputs
  Delay_ms(2000)
  text  = "Voltage="            ' Define the third message

  while 1                       ' Endless loop
    adc_rd  = ADC_Read(2)       ' A/D conversion. Pin RA2 is an input.
    Lcd_Out(2,1,text)           ' Write result in the second line

    tlong = adc_rd * 5000       ' Convert the result in millivolts
    tlong = tlong / 1023        ' 0..1023 -> 0-5000mV

    ch = (tlong / 1000) mod 10        ' Extract volts (thousands of millivolts)
                                      ' from result
    Lcd_Chr(2,9,48+ch)                ' Write result in ASCII format

    Lcd_Chr_CP(".")                   ' Write the decimal pint

    ch = (tlong / 100) mod 10         ' Extract hundreds of millivolts
    Lcd_Chr_CP(48+ch)                 ' Write result in ASCII format

    ch = (tlong / 10) mod 10          ' Extract tens of millivolts
    Lcd_Chr_CP(48+ch)                 ' Write result in ASCII format

    ch = tlong mod 10                 ' Extract digits for millivolts
    Lcd_Chr_CP(48+ch)                 ' Write result in ASCII format

    Lcd_Chr_CP("V")                   ' Write a mark for voltage "V"

    Delay_ms(1)                       ' 1mS delay
  wend

       end.                           ' End of program
```