mikroICD®

User manual

mikroICD debugger is a highly effective tool for real-time debugging at hardware level. It enables you to view program variable values, Special Function Registers (SFRs) and EEPROM while the program is running. This manual contains practical example on how to create a new project, write and compile code and test the results.

Debugger

MikroElektronika
SOFTWARE AND HARDWARE SOLUTIONS FOR EMBEDDED WORLD ...making it simple

TO OUR VALUED CUSTOMERS

I want to express my thanks to you for being interested in our products and for having confidence in mikroElektronika.
The primary aim of our company is to design and produce high quality electronic products and to constantly improve the performance thereof in order to better suit your needs.

Nebojsa Matic
General Manager

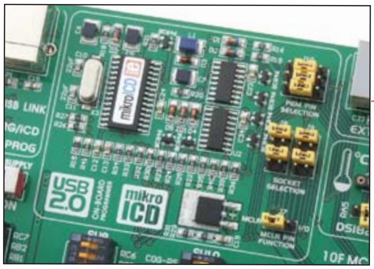
TABLE OF CONTENTS

1.0. mikroICD® Overview

The *mikroICD* (In-Circuit Debugger) is a hardware tool designed for testing and debugging programs on most PIC microcontrollers. It also enables you to monitor the state of all registers within the microcontroller which operates in real environment. In order for the *mikroICD* debugger to be used, it is necessary to have the appropriate hardware as well as to install additional software.

Hardware

The *mikroICD* is an integral part of the *PICflash* programmer intended for use with PIC16, PIC18, PIC24, dsPIC30 and dsPIC33 microcontrollers. It is built into all PIC® development systems designed by MikroElektronika such as *EasyPIC6, EasyPIC5, BigPIC5*, EasydsPIC4, LV 24-33A etc. Thanks to mikroICD support, the *PICflash*® programmer is a multifunctional device as it may be used for programming PIC microcontrollers as well as for debugging programs executed in real time. Besides, the *PICflash* programmer is also available as a stand-alone device used for programming chips built into (soldered on) the target device.



PICflash programmer built into the EasyPIC6 development system

PIC microcontrollers are connected to the programmer through the PGC, PGD and MCLR pins. In case that such programmer is used for the programming only, its hardware will automatically break connection with these pins after loading the HEX code, thus enabling them to be used for other purposes.

In case that the PICflash programmer is also used for debugging (mikroICD is enabled), these pins will be used for communication with the PC and cannot be used for other purposes.

The process of testing and debugging programs in real environment is performed by monitoring the state of all registers within the microcontroller. The mikroICD debugger also offers functions such as running a program step by step (single stepping), pausing the program execution to examine the state of currently active registers using breakpoints, tracking the values of some variables etc. In this case the *mikroICD* debugger is connected to the PC all the time so that the PGC, PGD and MCLR/Vpp pins cannot be used for the operation of the target device.

Software

The *mikroICD* debugger needs the additional software to be installed on the PC for its operation. Such software includes:

PICflash v7.02 (or later version) is a program used along with the *PICflash* programmer's hardware. It enables you to select the microcontroller to be programmed and to set up its mode. You can download it for free from our website at *www.mikroe.com*.

Drivers necessary for the proper operation of the *PICflash* programmer enable communication between the PC and the *PICflash* programmer's hardware.

Compilers are programs used for compiling programs written in high-level programming languages into executable file (HEX code). Here is a list of compilers providing the mikroICD support:

mikroC PRO® 2009;
mikroBasic PRO® 2009;
mikroPascal PRO® 2009;
mikroC® (dsPIC30/33 & PIC24);
mikroBasic® (dsPIC30/33 & PIC24); and
mikroPascal® (dsPIC30/33 & PIC24).

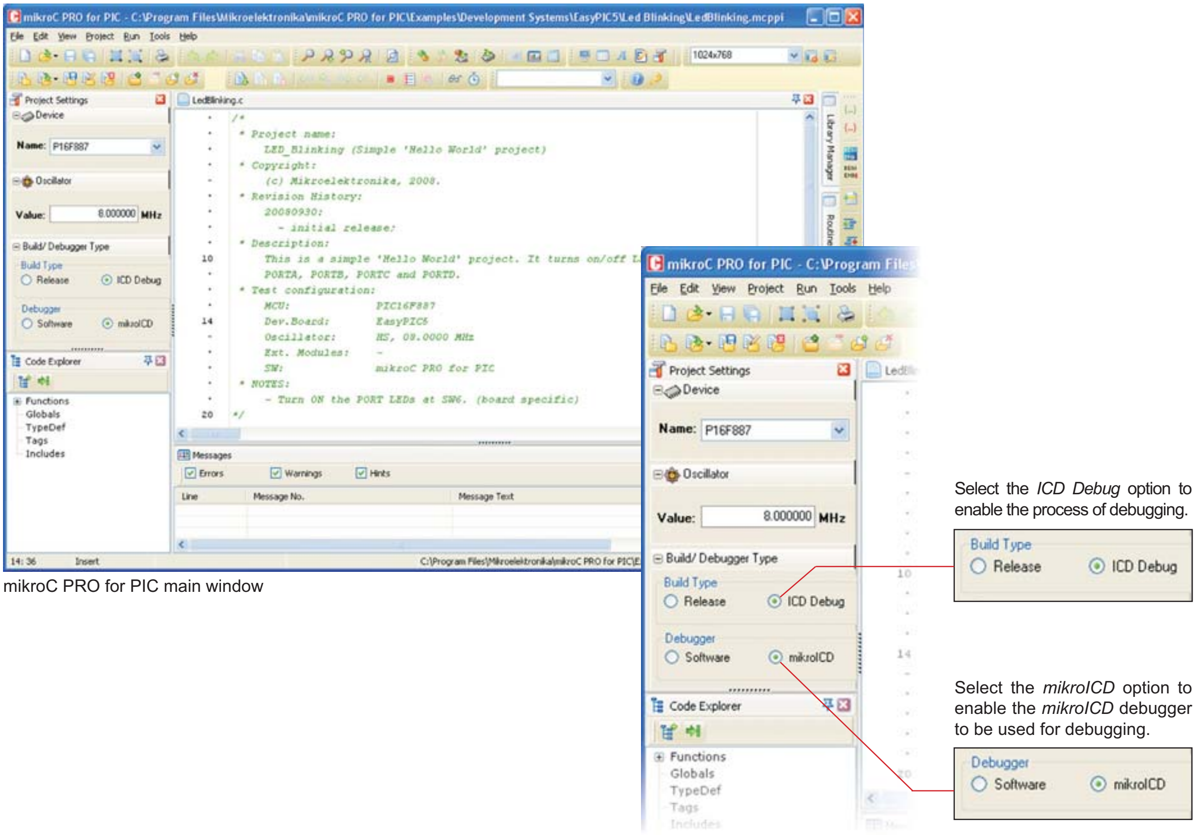
The compilers' demo versions can be downloaded for free from our website at *www.mikroe.com*

2.0. Using mikroICD

The *mikroICD* debugger comes with all PIC and dsPIC compilers designed by MikroElektronika. This manual illustrates and describes its operation in the *mikroC PRO for PIC* compiler. The principle of the operation is the same for *mikroBasic* and *mikroPascal* compilers as well.

Step 1: Writing the Program and Setting up the Project for Debugging

Creating a new project and writing a program in the compiler's main window should be done first. The next step is to set up the project for debugging using the mikroICD debugger. To perform this, it is necessary to select the following options in the *Project Settings* window:



mikroC PRO for PIC - C:\Program Files\Mikroelektronika\mikroC PRO for PIC\Examples\Development Systems\EasyPIC5\Led Blinking\LedBlinking.mcppi

LedBlinking.c

```
/*
 * Project name:
     LED_Blinking (Simple 'Hello World' project)
 * Copyright:
     (c) Mikroelektronika, 2008.
 * Revision History:
     20080930:
       - initial release;
 * Description:
     This is a simple 'Hello World' project. It turns on/off L
     PORTA, PORTB, PORTC and PORTD.
 * Test configuration:
     MCU:             PIC16F887
     Dev.Board:       EasyPIC5
     Oscillator:      HS, 08.0000 MHz
     Ext. Modules:    -
     SW:              mikroC PRO for PIC
 * NOTES:
     - Turn ON the PORT LEDs at SW6. (board specific)
*/
```

Project Settings

Device — Name: P16F887

Oscillator — Value: 8.000000 MHz

Build/ Debugger Type

Build Type: ○ Release ● ICD Debug

Debugger: ○ Software ● mikroICD

mikroC PRO for PIC main window

Select the *ICD Debug* option to enable the process of debugging.

Build Type: ○ Release ● ICD Debug

Select the *mikroICD* option to enable the *mikroICD* debugger to be used for debugging.

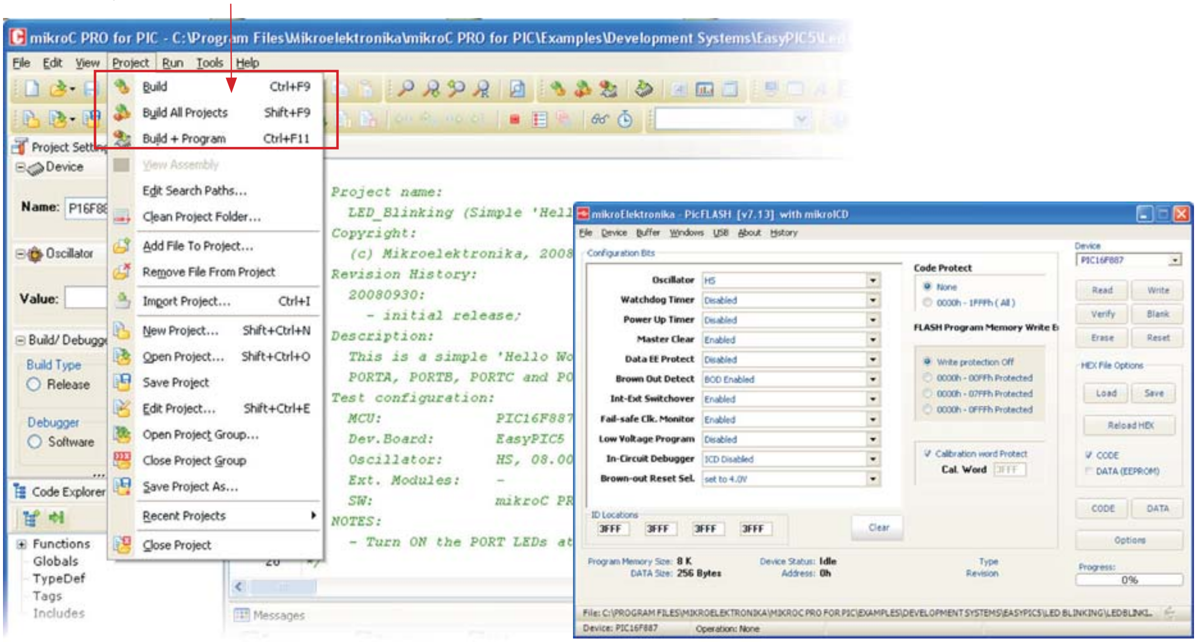
Debugger: ○ Software ● mikroICD

Step 2: Compiling the Program and Dumping It into the Microcontroller

The program has to be compiled into the machine code before it is downloaded into the microcontroller. In order to start the process of compiling, click one of the appropriate shortcut icons or select the following option from the compiler's *Project* drop-down menu:

Build+Program [Ctrl+F11]

By clicking on this command, the *PICflash* programmer will be automatically activated after completing the process of compiling and the compiled program (HEX code) will be immediately loaded into the microcontroller's program memory. The programming progress will be shown in the *PICflash* programmer's window to appear on the screen:



Project drop-down menu - Build options

```
Project name:
  LED_Blinking (Simple 'Hell
Copyright:
  (c) Mikroelektronika, 2008
Revision History:
  20080930:
   - initial release;
Description:
  This is a simple 'Hello Wo
  PORTA, PORTB, PORTC and PO
Test configuration:
  MCU:             PIC16F887
  Dev.Board:       EasyPIC5
  Oscillator:      HS, 08.00
  Ext. Modules:    -
  SW:              mikroC PR
NOTES:
  - Turn ON the PORT LEDs at
```

Device: PIC16F887

Oscillator: HS
Watchdog Timer: Disabled
Power Up Timer: Disabled
Master Clear: Enabled
Data EE Protect: Disabled
Brown Out Detect: BOD Enabled
Int-Ext Switchover: Enabled
Fail-safe Clk. Monitor: Enabled
Low Voltage Program: Disabled
In-Circuit Debugger: ICD Disabled
Brown-out Reset Sel.: set to 4.0V

Code Protect
● None
○ 0000h - 1FFFh (All)

FLASH Program Memory Write E
● Write protection Off
○ 0000h - 00FFh Protected
○ 0000h - 07FFh Protected
○ 0000h - 0FFFh Protected

☑ Calibration word Protect
Cal. Word 3FFF

☑ CODE
□ DATA (EEPROM)

ID Locations: 3FFF 3FFF 3FFF 3FFF

Program Memory Size: 8 K
DATA Size: 256 Bytes
Device Status: Idle
Address: 0h
Type
Revision
Progress: 0%

File: C:\PROGRAM FILES\MIKROELEKTRONIKA\MIKROC PRO FOR PIC\EXAMPLES\DEVELOPMENT SYSTEMS\EASYPIC5\LED BLINKING\LEDBLINKI...
Device: PIC16F887 Operation: None

PICflash programmer's main window

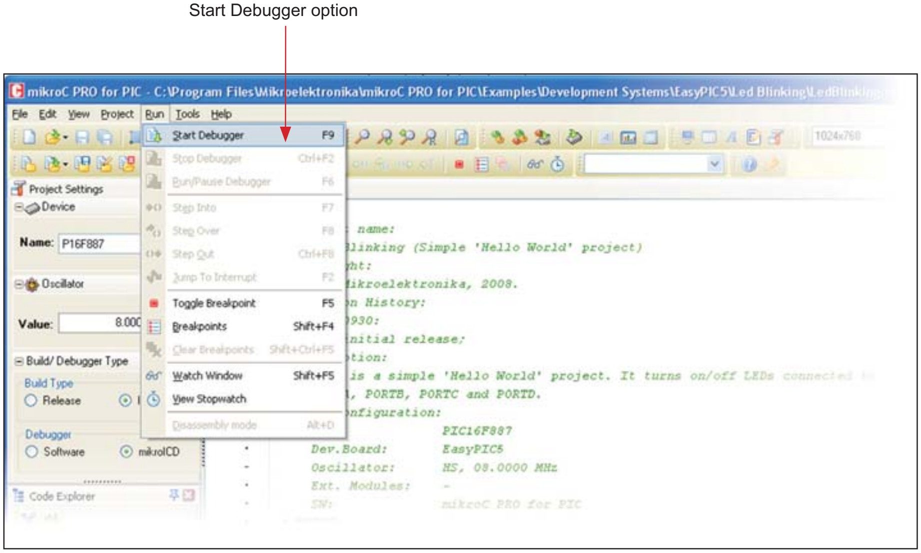
NOTE: In addition to the aforementioned build option which causes the program to be automatically compiled and loaded into the microcontroller memory, there are two other build options in the *Project* drop-down menu:

Build [Ctrl+F9]	If the project consists of one file; and
Build All [Shift+F9]	If the project consists of several files;

These options are intended for compiling only and do not start up the programming process. Accordingly, when these are used, the HEX code has to be loaded into the microcontroller from within the PICflash program using the *Load* and *Write* options. More information on MCU programming using the PICflash program may be found in the *PICflash Programmer* manual.

Step 3: Starting up the mikroICD Debugger

After the microcontroller has been successfully programmed, it is time to start up the *mikroICD* by selecting the *Start Debugger* option from the *Run* drop-down menu.

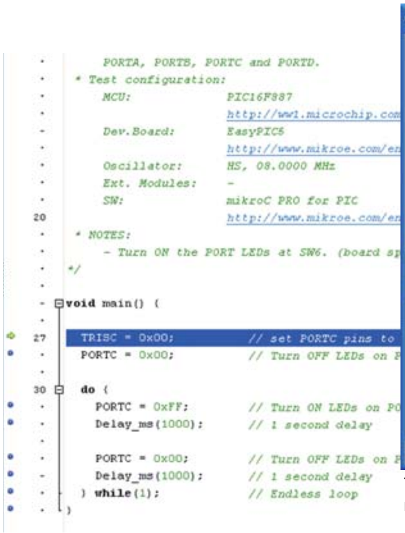


Name: P16F887

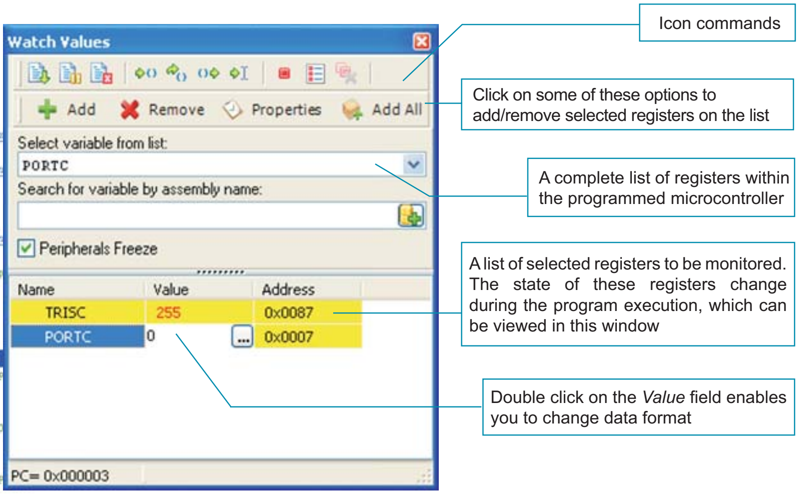
Value: 8.000

```
name:
Blinking (Simple 'Hello World' project)
ht:
ikroelektronika, 2008.
n History:
0930:
nitial release;
tion:
is a simple 'Hello World' project. It turns on/off LEDs connected to
, PORTB, PORTC and PORTD.
nfiguration:
                 PIC16F887
Dev.Board:       EasyPIC5
Oscillator:      HS, 08.0000 MHz
Ext. Modules:    -
SW:              mikroC PRO for PIC
```

As mentioned before, the *mikroICD* debugger enables you to directly monitor the state of all registers within the microcontroller. Some of the most frequently used debugger options are: *Step Into, Step Over, Run to Cursor* and *Step Out*. For the *Watch Values* window to appear on the screen, select the *View > Debug Windows > Watch Window* option.



```
     PORTA, PORTB, PORTC and PORTD.
 * Test configuration:
     MCU:              PIC16F887
                       http://ww1.microchip.com
     Dev.Board:        EasyPIC5
                       http://www.mikroe.com/en
     Oscillator:       HS, 08.0000 MHz
     Ext. Modules:     -
     SW:               mikroC PRO for PIC
                       http://www.mikroe.com/en
 * NOTES:
     - Turn ON the PORT LEDs at SW6. (board sp
*/

void main() {

  TRISC = 0x00;           // set PORTC pins to
  PORTC = 0x00;           // Turn OFF LEDs on P

  do {
    PORTC = 0xFF;         // Turn ON LEDs on PO
    Delay_ms(1000);       // 1 second delay

    PORTC = 0x00;         // Turn OFF LEDs on P
    Delay_ms(1000);       // 1 second delay
  } while(1);             // Endless loop
}
```



Select variable from list: PORTC

Search for variable by assembly name:

☑ Peripherals Freeze

Name	Value	Address
TRISC	255	0x0087
PORTC	0	0x0007

PC= 0x000003

Icon commands

Click on some of these options to add/remove selected registers on the list

A complete list of registers within the programmed microcontroller

A list of selected registers to be monitored. The state of these registers change during the program execution, which can be viewed in this window

Double click on the *Value* field enables you to change data format

The *Watch Values* window showing the state of the microcontroller's registers and program variables

3.0. Practical Example of Using mikroICD

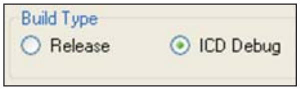
Here is a step-by-step illustration of the operation of the *mikroICD*:

Step 1: Writing the Program and Setting up the Project for ICD Debugging

Program Example

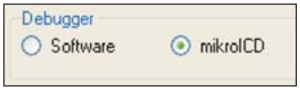
```
/* Here is a simple program to demonstrate the operation of the microcontroller. The PORTC port’s
pins are configured as digital outputs and their logic state changes once per second. Establishing
connection between such port and LEDs will cause LEDs to blink simultaneously */

void main() {

  TRISC = 0x00;            // Configure PORTC pins as outputs
  PORTC = 0x00;            // Turn OFF LEDs on PORTC

  do {
    PORTC = 0xFF;          // Turn ON LEDs on PORTC
    Delay_ms(1000);        // 1 second delay

    PORTC = 0x00;          // Turn OFF LEDs on PORTC
    Delay_ms(1000);        // 1 second delay
  } while(1);              // Endless loop
}
```

When the program has been written, it is necessary to select the appropriate debugging mode before it is compiled into the HEX code, in order to perform debugging using the *mikroICD* debugger.



Build Type
○ Release ● ICD Debug

First, to debug program, select the *ICD Debug* option from the *Project Settings* window.

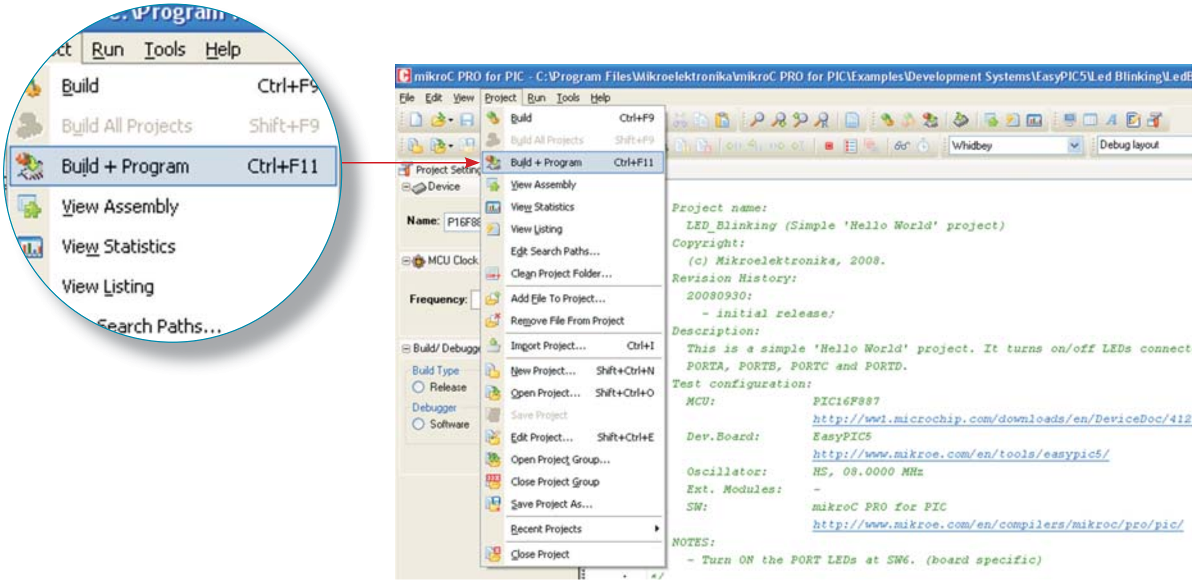


Debugger
○ Software ● mikroICD

Then, select the *mikroICD* option to enable the *mikroICD* debugger to be used for debugging.

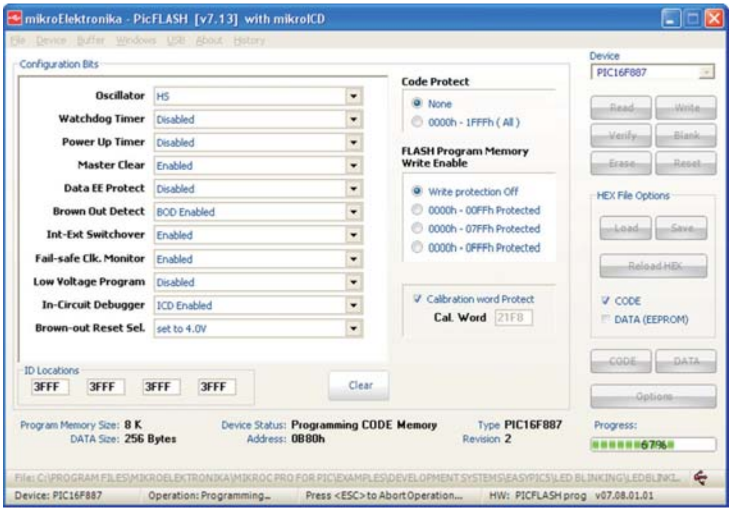
Step 2: Compiling the Program and Dumping It into the Microcontroller

In order to compile the program into the HEX code and automatically dump it into the microcontroller, select the *Build+program* option [Ctrl+F11] from the *Project* drop-down menu.



```
Project name:
  LED_Blinking (Simple 'Hello World' project)
Copyright:
  (c) Mikroelektronika, 2008.
Revision History:
  20080930:
    - initial release;
Description:
  This is a simple 'Hello World' project. It turns on/off LEDs connect
  PORTA, PORTB, PORTC and PORTD.
Test configuration:
  MCU:             PIC16F887
                   http://ww1.microchip.com/downloads/en/DeviceDoc/412
  Dev.Board:       EasyPIC5
                   http://www.mikroe.com/en/tools/easypic5/
  Oscillator:      HS, 08.0000 MHz
  Ext. Modules:    -
  SW:              mikroC PRO for PIC
                   http://www.mikroe.com/en/compilers/mikroc/pro/pic/
NOTES:
  - Turn ON the PORT LEDs at SW6. (board specific)
```

Immediately after completing the compiling process, the PICflash programmer's window will appear on the screen. In the right bottom corner thereof, there is the *Progress* bar showing the programming progress. If the *Tools > Options > Tools > PICflash Options > Close when finished* option is ticked off the PIC flash programmer's window will be automatically closed after programming.



mikroElektronika - PicFLASH [v7.13] with mikroICD
File Device Buffer Windows USB About History
Read
Write
Verify
Blank
Erase
Reset
Load
Save
Reload HEX
CODE
DATA
Options
Clear

Configuration Bits

Oscillator	HS
Watchdog Timer	Disabled
Power Up Timer	Disabled
Master Clear	Enabled
Data EE Protect	Disabled
Brown Out Detect	BOD Enabled
Int-Ext Switchover	Enabled
Fail-safe Clk. Monitor	Enabled
Low Voltage Program	Disabled
In-Circuit Debugger	ICD Enabled
Brown-out Reset Sel.	set to 4.0V

ID Locations: 3FFF 3FFF 3FFF 3FFF

Code Protect
- ● None
- ○ 0000h - 1FFFh (All)

FLASH Program Memory Write Enable
- ● Write protection Off
- ○ 0000h - 00FFh Protected
- ○ 0000h - 07FFh Protected
- ○ 0000h - 0FFFh Protected

☑ Calibration word Protect

Cal. Word 21F8

Device: PIC16F887

HEX File Options
- ☑ CODE
- □ DATA (EEPROM)

Program Memory Size: 8 K
DATA Size: 256 Bytes

Device Status: Programming CODE Memory
Address: 0B80h

Type PIC16F887
Revision 2

Progress: 67%

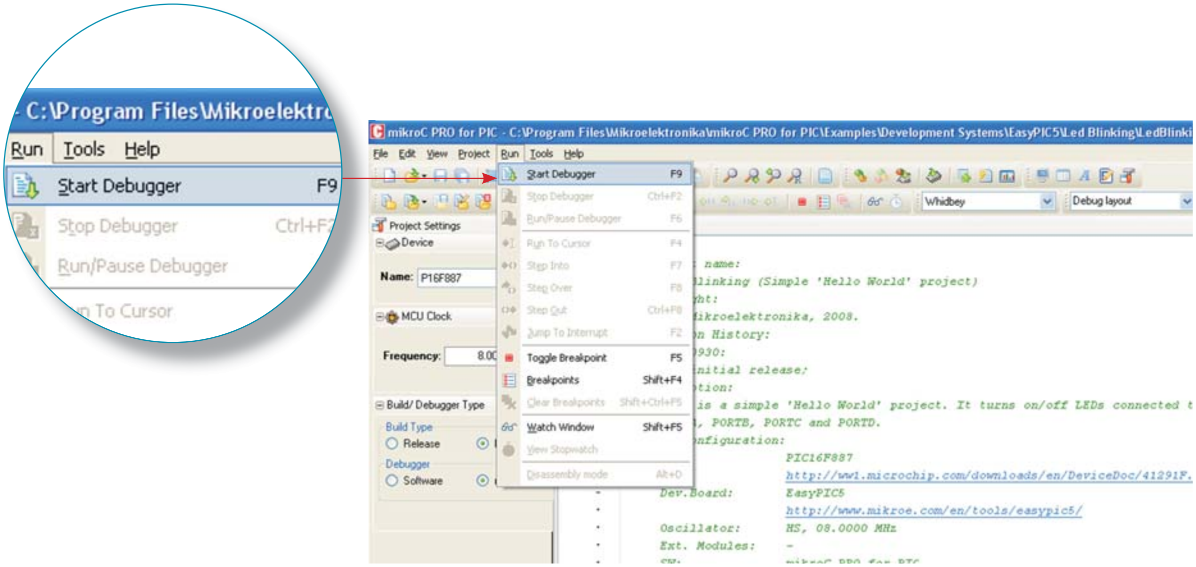
File: C:\PROGRAM FILES\MIKROELEKTRONIKA\MIKROC PRO FOR PIC\EXAMPLES\DEVELOPMENT SYSTEMS\EASYPIC5\LED BLINKING\LEDBLINKI...

Device: PIC16F887
Operation: Programming...
Press <ESC> to Abort Operation...
HW: PICFLASH prog v07.08.01.01

PICflash programmer's main window

Step 3: Starting up mikroICD and Line-by-Line Program Execution

When the program is loaded into the microcontroller, its execution in real time can be monitored by using the *mikroICD* debugger. To start it up, select the *Start Debugger* option from the *Run* drop-down menu or click the [F9] button.



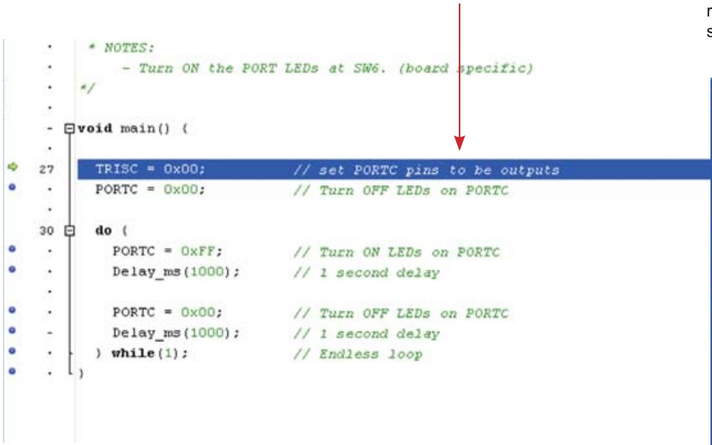
Name: P16F887

Frequency: 8.00

```
 name:
linking (Simple 'Hello World' project)
ht:
ikroelektronika, 2008.
n History:
930:
nitial release;
tion:
is a simple 'Hello World' project. It turns on/off LEDs connected t
, PORTB, PORTC and PORTD.
nfiguration:
                    PIC16F887
                    http://ww1.microchip.com/downloads/en/DeviceDoc/41291F.
Dev.Board:          EasyPIC5
                    http://www.mikroe.com/en/tools/easypic5/
Oscillator:         HS, 08.0000 MHz
Ext. Modules:       -
```

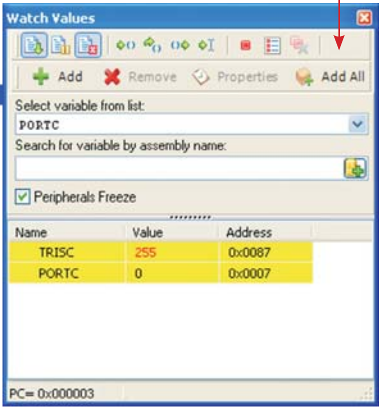
Now, the program within the microcontroller will be executed line by line by pressing the [F8] button.

During program execution, the program line to be next executed is highlighted in blue by default



```
 * NOTES:
     - Turn ON the PORT LEDs at SW6. (board specific)
*/

void main() {

  TRISC = 0x00;              // set PORTC pins to be outputs
  PORTC = 0x00;              // Turn OFF LEDs on PORTC

  do {
    PORTC = 0xFF;            // Turn ON LEDs on PORTC
    Delay_ms(1000);          // 1 second delay

    PORTC = 0x00;            // Turn OFF LEDs on PORTC
    Delay_ms(1000);          // 1 second delay
  } while(1);                // Endless loop
}
```

Only two registers are selected here to be monitored. Use the *Add All* option to add all registers within the microcontroller to the list of selected registers to be monitored.



Select variable from list: PORTC

Search for variable by assembly name:

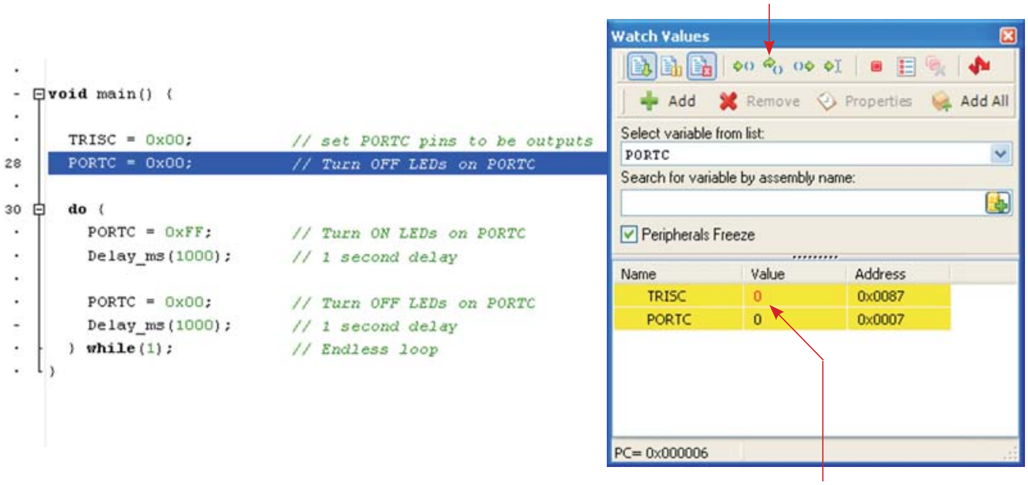
☑ Peripherals Freeze

Name	Value	Address
TRISC	255	0x0087
PORTC	0	0x0007

PC= 0x000003

The *Watch Values* window enables you to monitor the state of selected registers and to view how their states change during program execution.
In this example, the first instruction is executed using the *Step Over* option. In higher programming languages such option executes the whole program line regardless of how many assembly instructions it consists of.

Step Over
Command



```
   void main() {

     TRISC = 0x00;          // set PORTC pins to be outputs
28   PORTC = 0x00;          // Turn OFF LEDs on PORTC

30   do {
       PORTC = 0xFF;        // Turn ON LEDs on PORTC
       Delay_ms(1000);      // 1 second delay

       PORTC = 0x00;        // Turn OFF LEDs on PORTC
       Delay_ms(1000);      // 1 second delay
     } while(1);            // Endless loop
   }
```

Select variable from list: PORTC

Search for variable by assembly name:

☑ Peripherals Freeze

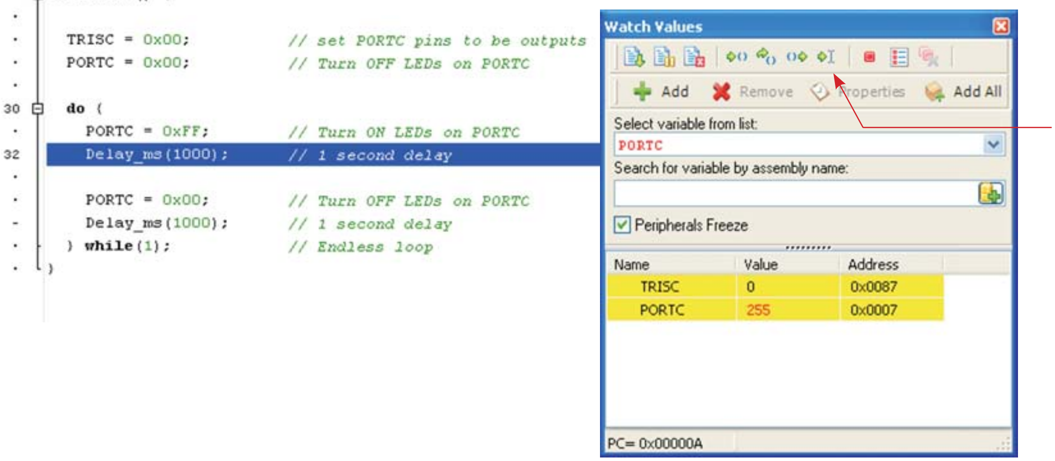
Name	Value	Address
TRISC	0	0x0087
PORTC	0	0x0007

PC= 0x000006

The most recent state of registers is highlighted in red

By executing the same instruction (*Step Over* [F8]) two more times, the 32nd program line which contains the Delay_ms(1000) command will be reached. To perform its execution, it is advisable to use the *Run to Cursor* [F4] option as it executes the program at full speed.

The *Run to Cursor* command will cause the program to be executed at full speed until it encountes the program line with cursor placed.



```
   void main() {

     TRISC = 0x00;          // set PORTC pins to be outputs
     PORTC = 0x00;          // Turn OFF LEDs on PORTC

30   do {
       PORTC = 0xFF;        // Turn ON LEDs on PORTC
32     Delay_ms(1000);      // 1 second delay

       PORTC = 0x00;        // Turn OFF LEDs on PORTC
       Delay_ms(1000);      // 1 second delay
     } while(1);            // Endless loop
   }
```

Select variable from list: PORTC

Search for variable by assembly name:

☑ Peripherals Freeze

Name	Value	Address
TRISC	0	0x0087
PORTC	255	0x0007

PC= 0x00000A

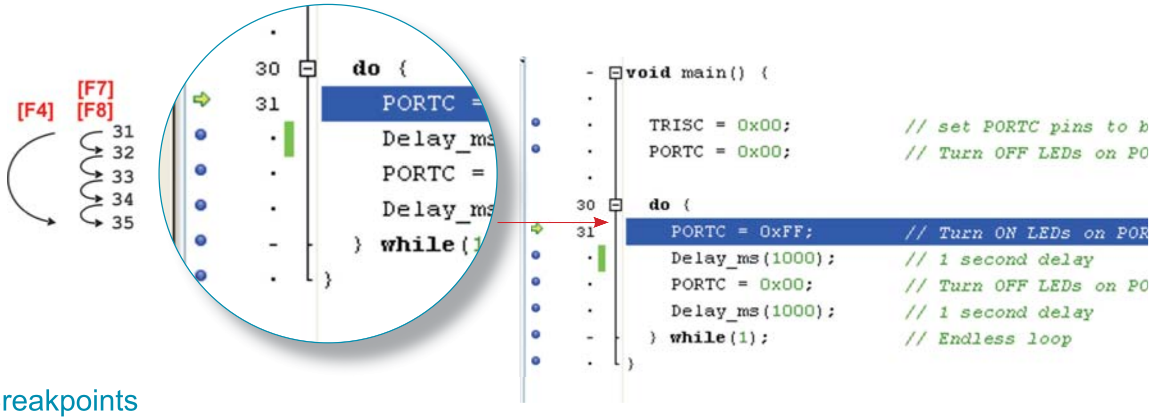
Run to Cursor
Command

4.0. mikroICD Debugger Options for Advanced Users

The following text describes the advanced options offered by the *mikroICD* debugger.

Real-Time Debugging

The *Step Into [F7]* and *Step Over [F8]* commands enable the program to be executed line by line. Program execution is a slow process in this case, and as such, is suitable for short programs. Unlike them, the *Run/Pause Debugger [F6]* and *Run To Cursor [F4]* commands enable the program to be executed in real time and therefore be much faster. The speed of program execution depends on the microcontroller's own clock. By pressing [F6] or selecting the *Run/Pause Debugger* option, the *mikroICD* is temporarily halted and the microcontroller executes the loaded program at full speed. Another pressure on the same button reactivates the *mikroICD* and the program execution stops at reached location. By pressing [F4], the microcontroller will proceed with program execution at high speed until it reaches the line selected by the cursor.



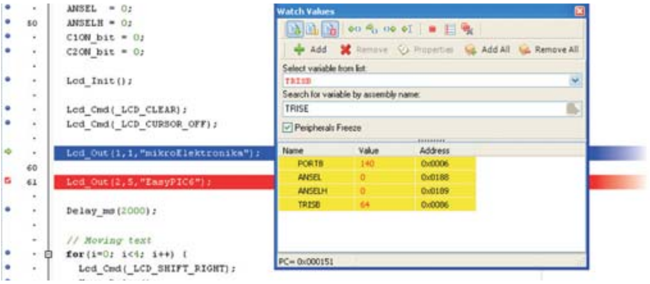
```
void main() {

  TRISC = 0x00;           // set PORTC pins to b
  PORTC = 0x00;           // Turn OFF LEDs on PO

  do {
    PORTC = 0xFF;         // Turn ON LEDs on POR
    Delay_ms(1000);       // 1 second delay
    PORTC = 0x00;         // Turn OFF LEDs on PO
    Delay_ms(1000);       // 1 second delay
  } while(1);             // Endless loop
}
```

Breakpoints

The *mikroICD* enables each program line to be marked with a breakpoint. The breakpoint is an intentional stopping or pausing place in the program used for the purpose of debugging. Breakpoints are placed in the program by clicking the space to the left of the program line or by pressing [F5]. By selecting the *Run* command [F6], the microcontroller will execute the program from the current location (highlighted in blue) until it reaches a breakpoint (highlighted in red). The debugger halts after reaching the breakpoint.

There are two kinds of breakpoints - hardware and software breakpoints. The only visible difference between them is in the speed of program execution before it reaches the specified program line. Hardware breakpoints are placed within the microcontroller chip and provide considerably faster program execution. The number of hardware breakpoints is limited, whereas the total number of software breakpoints is unlimited. For example, 16-bit PIC microcontrollers have only one, whereas 18-bit PIC microcontrollers have up to 3 hardware breakpoints. When all hardware breakpoints are used, then remaining breakpoints in the program will be used as software breakpoints.

Click here to convert program line into breakpoint



```
  ANSEL  = 0;
  ANSELH = 0;
  C1ON_bit = 0;
  C2ON_bit = 0;

  Lcd_Init();

  Lcd_Cmd(_LCD_CLEAR);
  Lcd_Cmd(_LCD_CURSOR_OFF);

  Lcd_Out(1,1,"mikroElektronika");

  Lcd_Out(2,5,"EasyPIC6");

  Delay_ms(2000);

  // Moving text
  for(i=0; i<4; i++) {
    Lcd_Cmd(_LCD_SHIFT_RIGHT);
```

Select variable from list: TRISB

Search for variable by assembly name: TRISE

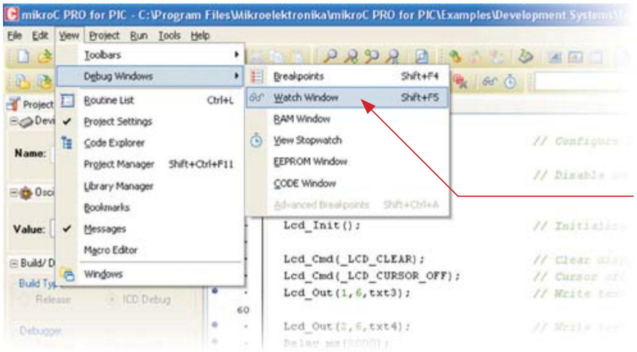
☑ Peripherals Freeze

Name	Value	Address
PORTB	140	0x0006
ANSEL	0	0x0188
ANSELH	0	0x0189
TRISB	64	0x0086

PC= 0x000151

Watch Window Option

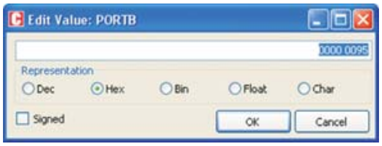
The *Watch Window* option allows you to monitor the values of program variables as well as the contents of SFRs while the program is running. As soon as the program is loaded into the microcontroller, the *Watch Values* window appears on the screen. To reopen this window, when removed, select the option *View* > *Debug Windows* > *Watch Window*.



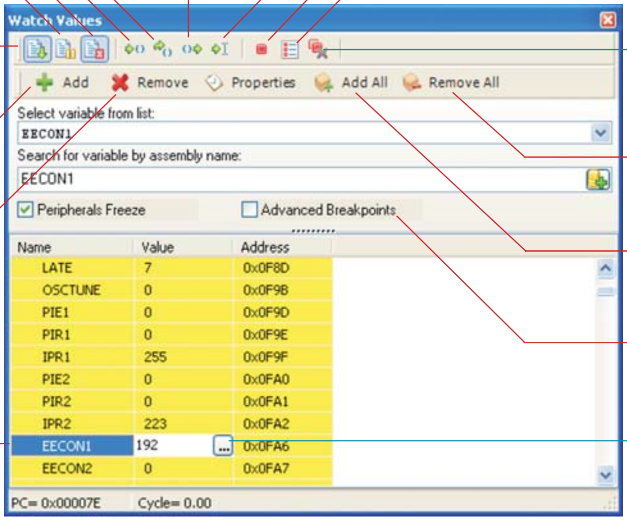
```
Lcd_Init();

Lcd_Cmd(_LCD_CLEAR);
Lcd_Cmd(_LCD_CURSOR_OFF);
Lcd_Out(1,6,txt3);

Lcd_Out(2,6,txt4);
Delay_ms(2000);
```

Watch Window option

The *Watch Values* window displays data in three columns: register or variable names, their values and memory addresses. Double click on any variable opens the *Edit Value* window which allows you to assign it a new value. It is also possible to change data format (decimal, hexadecimal, binary, floating or character) in this window.



Edit Value window



Name	Value	Address
LATE	7	0x0F8D
OSCTUNE	0	0x0F9B
PIE1	0	0x0F9D
PIR1	0	0x0F9E
IPR1	255	0x0F9F
PIE2	0	0x0FA0
PIR2	0	0x0FA1
IPR2	223	0x0FA2
EECON1	192	0x0FA6
EECON2	0	0x0FA7

Step Into [F7]

Step Over [F8]

Run to Cursor [F4]

Toggle Breakpoint [F5]

Stop Debugger

Step Out [Ctrl+F8]

Show/Hide Breakpoints Shift + [F4]

Run/Pause Debugger

Start Debugger

Clear Breakpoints Shift +Ctrl + F5]

Add selected variable to the list

Remove all variables from the list

Remove selected variable from the list

Add all variables to the list

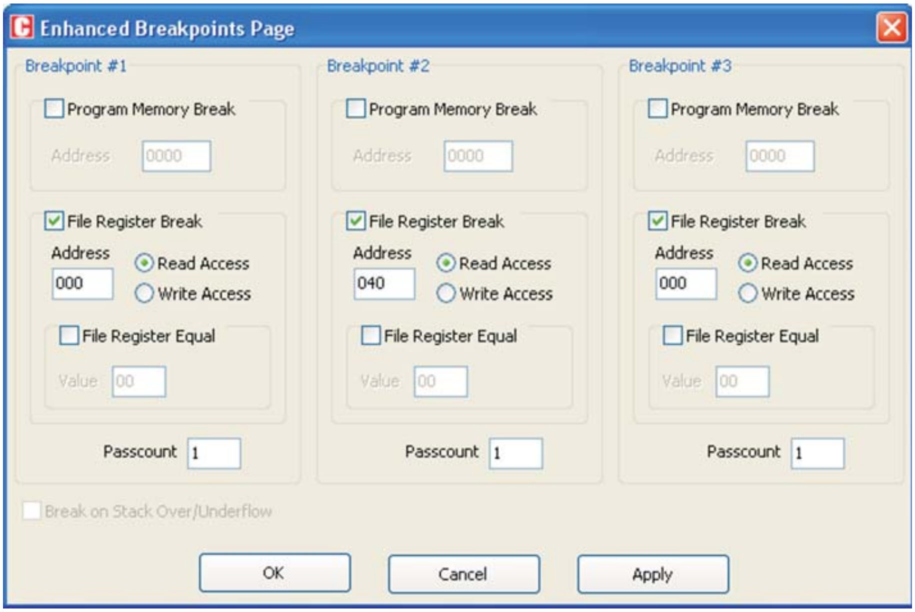
Advanced Breakpoints option

Selected variable

Change format of the selected variable

Advanced Breakpoints Option

The *mikroICD* provides the means for using the *Advanced Breakpoints* option with PIC18 and PIC18FJ microcontrollers. To enable it, tick the *Advanced Breakpoints* checkbox within the *Watch Values* window. To configure the *Advanced Breakpoints* option it is necessary to start up *mikroICD* [F9] and select the *View › Debug Windows › Advanced Breakpoints* option or to use the [Ctrl+Shift+A] shortcut icon.



Enhanced Breakpoints Page

Breakpoint #1
- □ Program Memory Break — Address 0000
- ☑ File Register Break — Address 000 — ● Read Access ○ Write Access
- □ File Register Equal — Value 00
- Passcount 1

Breakpoint #2
- □ Program Memory Break — Address 0000
- ☑ File Register Break — Address 040 — ● Read Access ○ Write Access
- □ File Register Equal — Value 00
- Passcount 1

Breakpoint #3
- □ Program Memory Break — Address 0000
- ☑ File Register Break — Address 000 — ● Read Access ○ Write Access
- □ File Register Equal — Value 00
- Passcount 1

□ Break on Stack Over/Underflow

Program Memory Break Option

The *Program Memory Break* option is used for placing breakpoints at specified addresses in the program memory. The value entered in the *Address* field must be in the .hex format.

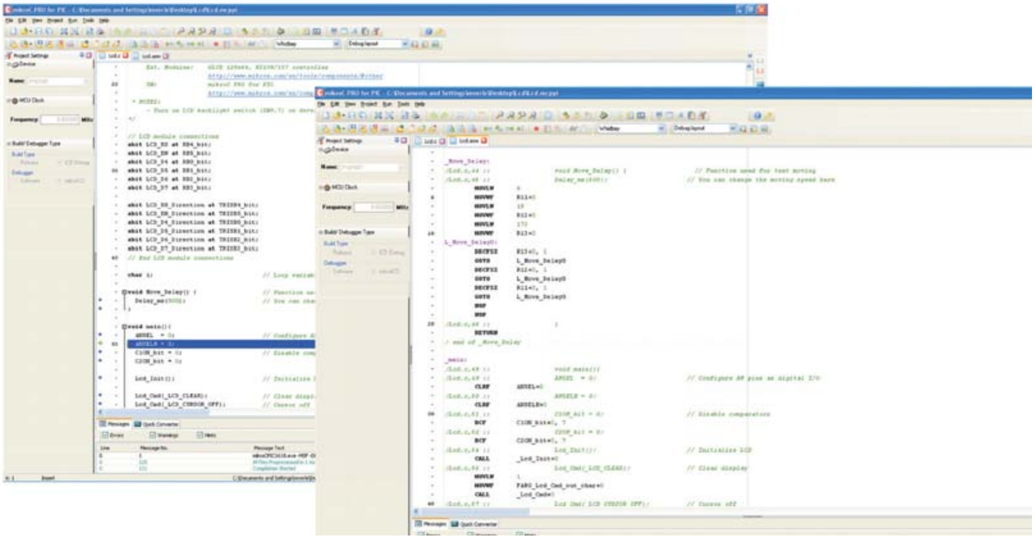
File Register Break Option

The *File Register Break* option is used for stopping code execution when read/write access to the specified data memory location occurs. If the *Read Access* option is selected, the *File Register Equal* option can be used for setting the appropriate value in the *Value* field. The program execution will be stopped when the value read from the specified data memory location matches the value written in the *Value* field. All the values entered in the *Value* field must be in the .hex format.

When the *Advanced Breakpoints* option is enabled, *mikroICD* operates in real-time mode, thus supporting only the following set of commands: Start Debugger [F9], Run/Pause Debugger [F6] and Stop Debugger [Ctrl+F2]. After reaching the first breakpoint, the *Advanced Breakpoints* option can be disabled and the process of debugging can be continued with a full set of commands. The number of advanced breakpoints is equal to the number of hardware breakpoints and depends on the microcontroller in use.

View Assembly Option

During the process of compiling, each program line written in a high-level programming language is replaced with one or more assembly instructions. To display program in the assembly language, select the *View Assembly* option from the *Project* drop-down menu. In this case, the process of simulating and debugging is performed in the same way as if the program is written in a high-level programming language.



Program written in the high level programming language...

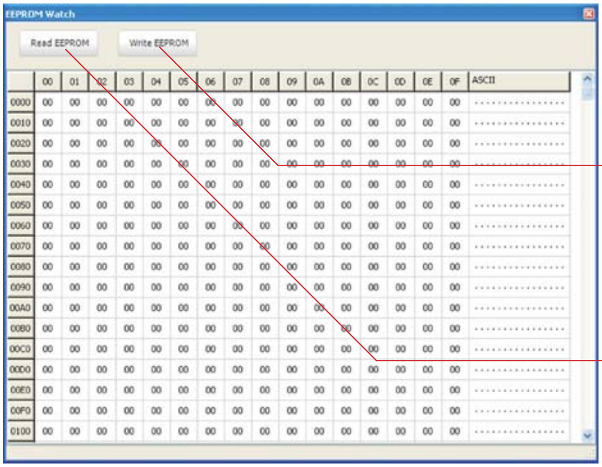
```
// LCD module connections
sbit LCD_RS at RB4_bit;
sbit LCD_EN at RB5_bit;
sbit LCD_D4 at RB0_bit;
sbit LCD_D5 at RB1_bit;
sbit LCD_D6 at RB2_bit;
sbit LCD_D7 at RB3_bit;

sbit LCD_RS_Direction at TRISB4_bit;
sbit LCD_EN_Direction at TRISB5_bit;
sbit LCD_D4_Direction at TRISB0_bit;
sbit LCD_D5_Direction at TRISB1_bit;
sbit LCD_D6_Direction at TRISB2_bit;
sbit LCD_D7_Direction at TRISB3_bit;
// End LCD module connections

char i;                              // Loop variab

void Move_Delay() {                  // Function us
  Delay_ms(500);                     // You can cha
}

void main(){
  ANSEL  = 0;                        // Configure A
  ANSELH = 0;
  C1ON_bit = 0;                      // Disable com
  C2ON_bit = 0;

  Lcd_Init();                        // Initialize 

  Lcd_Cmd(_LCD_CLEAR);               // Clear displ
  Lcd_Cmd(_LCD_CURSOR_OFF);          // Cursor off
```

```
_Move_Delay:
;Lcd.c,44 ::          void Move_Delay() {          // Function used for text moving
;Lcd.c,45 ::          Delay_ms(500);               // You can change the moving speed here
        MOVLW     6
        MOVWF     R11+0
        MOVLW     19
        MOVWF     R12+0
        MOVLW     173
        MOVWF     R13+0
L_Move_Delay0:
        DECFSZ    R13+0, 1
        GOTO      L_Move_Delay0
        DECFSZ    R12+0, 1
        GOTO      L_Move_Delay0
        DECFSZ    R11+0, 1
        GOTO      L_Move_Delay0
        NOP
        NOP
;Lcd.c,46 ::          }
        RETURN
; end of _Move_Delay

_main:
;Lcd.c,48 ::          void main(){
;Lcd.c,49 ::          ANSEL  = 0;                  // Configure AN pins as digital I/O
        CLRF      ANSEL+0
;Lcd.c,50 ::          ANSELH = 0;
        CLRF      ANSELH+0
;Lcd.c,52 ::          C1ON_bit = 0;                // Disable comparators
        BCF       C1ON_bit+0, 7
;Lcd.c,53 ::          C2ON_bit = 0;
        BCF       C2ON_bit+0, 7
;Lcd.c,55 ::          Lcd_Init();                  // Initialize LCD
        CALL      _Lcd_Init+0
;Lcd.c,56 ::          Lcd_Cmd(_LCD_CLEAR);         // Clear display
        MOVLW     1
        MOVWF     FARG_Lcd_Cmd_out_char+0
        CALL      _Lcd_Cmd+0
;Lcd.c,57 ::          Lcd_Cmd(_LCD_CURSOR_OFF);    // Cursor off
```

...the same program compiled in the assembly language

EEPROM Watch Window

The *EEPROM Watch* window will appear by selecting the *View › Debug Windows > EEPROM Window* option. It shows the values currently stored in the PIC internal EEPROM memory.



	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	ASCII
0000	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
0010	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
0020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
0030	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
0040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
0050	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
0060	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
0070	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
0080	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
0090	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
0100	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....

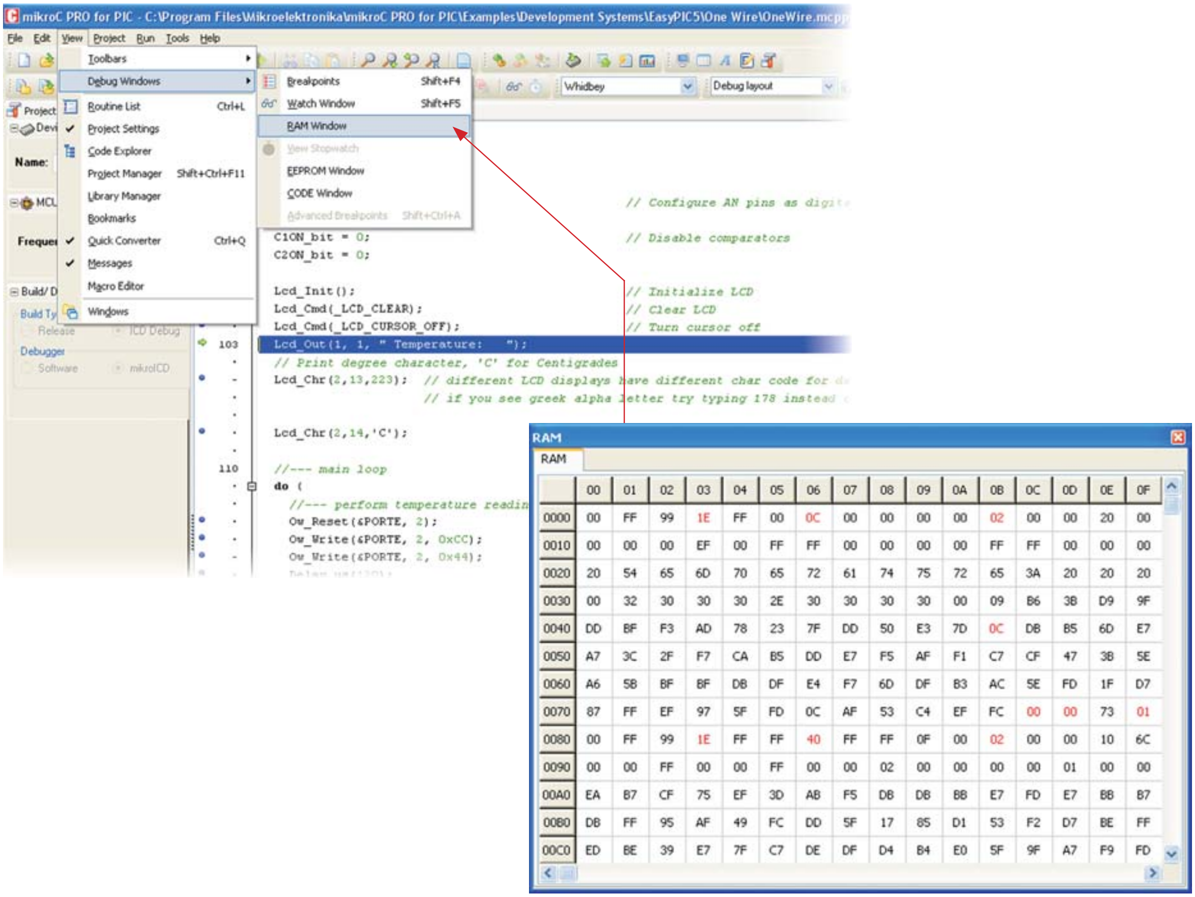
Pressing the *Write EEPROM* button causes data from the *EEPROM Watch* window to be loaded into internal EEPROM memory of the microcontroller.

Pressing the *Read EEPROM* button causes the contents of the EEPROM memory to be read and shown in the *EEPROM Watch window*.

EEPROM Watch window

RAM Window

The *mikroICD* allows you to view the contents of the microcontroller’s RAM memory in the *RAM* window by clicking the *View* > *Debug Windows* > *RAM Window* option. Unlike the *Watch Window* option, all memory locations are displayed in a table. The content of each RAM location is displayed in the hexadecimal format and may be changed at any time during the operation of the microcontroller. Changed values are directly written into the microcontroller by clicking *Enter*.



```
                                                       // Configure AN pins as digit

C1ON_bit = 0;                                          // Disable comparators
C2ON_bit = 0;

Lcd_Init();                                            // Initialize LCD
Lcd_Cmd(_LCD_CLEAR);                                   // Clear LCD
Lcd_Cmd(_LCD_CURSOR_OFF);                              // Turn cursor off
Lcd_Out(1, 1, " Temperature:   ");
// Print degree character, 'C' for Centigrades
Lcd_Chr(2,13,223);  // different LCD displays have different char code for d
                    // if you see greek alpha letter try typing 178 instead

Lcd_Chr(2,14,'C');

//--- main loop
do {
  //--- perform temperature readin
  Ow_Reset(&PORTE, 2);
  Ow_Write(&PORTE, 2, 0xCC);
  Ow_Write(&PORTE, 2, 0x44);
  Delay_us(120);
```

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
0000	00	FF	99	1E	FF	00	0C	00	00	00	00	02	00	00	20	00
0010	00	00	00	EF	00	FF	FF	00	00	00	00	FF	FF	00	00	00
0020	20	54	65	6D	70	65	72	61	74	75	72	65	3A	20	20	20
0030	00	32	30	30	30	2E	30	30	30	30	00	09	B6	3B	D9	9F
0040	DD	BF	F3	AD	78	23	7F	DD	50	E3	7D	0C	DB	B5	6D	E7
0050	A7	3C	2F	F7	CA	B5	DD	E7	F5	AF	F1	C7	CF	47	3B	5E
0060	A6	5B	BF	BF	DB	DF	E4	F7	6D	DF	B3	AC	5E	FD	1F	D7
0070	87	FF	EF	97	5F	FD	0C	AF	53	C4	EF	FC	00	00	73	01
0080	00	FF	99	1E	FF	FF	40	FF	FF	0F	00	02	00	00	10	6C
0090	00	00	FF	00	00	FF	00	00	02	00	00	00	00	01	00	00
00A0	EA	B7	CF	75	EF	3D	AB	F5	DB	DB	BB	E7	FD	E7	BB	B7
00B0	DB	FF	95	AF	49	FC	DD	5F	17	85	D1	53	F2	D7	BE	FF
00C0	ED	BE	39	E7	7F	C7	DE	DF	D4	B4	E0	5F	9F	A7	F9	FD

RAM window

Here is a list of the most frequently used *mikroICD* options:

Name	Description	Function key
Start Debugger	Start up debugger	[F9]
Run/Pause Debugger	Run or pause debugger	[F6]
Stop Debugger	Stop debugger	[Ctrl+F2]
Step Into	Execute the current program line, then halts. If the program line executed calls another routine, the debugger steps into the routine and halts after executing the first instruction within it.	[F7]
Step Over	Execute the current program line, then halts. If the program line executed calls another routine, the debugger will not step into it. The whole routine will be executed and the debugger halts at the first instruction following the call.	[F8]
Step Out	Execute all remaining program lines within the subroutine. The debugger halts immediately upon exiting the subroutine. This option is provided with the PIC18 microcontroller family, but not with the PIC16 microcontroller family.	[Ctrl+F8]
Run To Cursor	Execute the program until reaching the cursor position.	[F4]
Toggle Breakpoint	During the process of debugging, the program executes until reaching a breakpoint. The *Toogle Breakpoints* option sets new breakpoints or removes those already set at the current cursor position.	[F5]
Show/Hide Breakpoints	To view all the breakpoints in the program, select the *Show/Hide Breakpoints* option from the *Run* drop-down menu or use the *Shift +F4* shortcut. Double click a breakpoint from the list to locate it.	[Shift+F4]
Clear Breakpoints	Clear all breakpoints from the program.	[Ctrl+Shift+F5]

DISCLAIMER

All the products owned by MikroElektronika are protected by copyright law and international copyright treaty. Therefore, this manual is to be treated as any other copyright material. No part of this manual, including product and software described herein, may be reproduced, stored in a retrieval system, translated or transmitted in any form or by any means, without the prior written permission of MikroElektronika. The manual PDF edition can be printed for private or local use, but not for distribution. Any modification of this manual is prohibited.

MikroElektronika provides this manual 'as is' without warranty of any kind, either expressed or implied, including, but not limited to, the implied warranties or conditions of merchantability or fitness for a particular purpose.

MikroElektronika shall assume no responsibility or liability for any errors, omissions and inaccuracies that may appear in this manual. In no event shall MikroElektronika, its directors, officers, employees or distributors be liable for any indirect, specific, incidental or consequential damages (including damages for loss of business profits and business information, business interruption or any other pecuniary loss) arising out of the use of this manual or product, even if MikroElektronika has been advised of the possibility of such damages. MikroElektronika reserves the right to change information contained in this manual at any time without prior notice, if necessary.

HIGH RISK ACTIVITIES

The products of MikroElektronika are not fault–tolerant nor designed, manufactured or intended for use or resale as on–line control equipment in hazardous environments requiring fail–safe performance, such as in the operation of nuclear facilities, aircraft navigation or communication systems, air traffic control, direct life support machines or weapon systems in which the failure of Software could lead directly to death, personal injury or severe physical or environmental damage ("High Risk Activities"). MikroElektronika and its suppliers specifically disclaim any expressed or implied warranty of fitness for High Risk Activities.

MikroElektronika
SOFTWARE AND HARDWARE SOLUTIONS FOR EMBEDDED WORLD ...making it simple

If you want to learn more about our products, please visit our website at www.mikroe.com

If you are experiencing some problems with any of our products or just need additional information, please place your ticket at www.mikroe.com/en/support

If you have any questions, comments or business proposals, do not hesitate to contact us at office@mikroe.com