Micro–IDE

Integrated Development Environment

6811 Simulator

Quick Start Guide

Document Revision: 1.03
Date: 30 May 2004

BiPOM Electronics, Inc.
16301 Blue Ridge Road, Missouri City, Texas 77489 USA
Telephone: 1-713-283-9900 Fax: 1-281-416-2806
E-mail: info@bipom.com
Web: www.bipom.com

1. Introduction

The 6811 Simulator simulates the operation of the Motorola 68HC11 microcontroller with up to 64K program memory and 1024 bytes internal data memory.

Powerful, yet easy to use and affordable 68HC11 simulator for Micro-IDE, which is a Windows-based Integrated Development Environment for micro-controllers, the 68HC11 Simulator simplifies code development with Micro C and Micro-IDE. Errors in user programs can be found and fixed quickly in simulation mode by avoiding time consuming downloads to the target board.

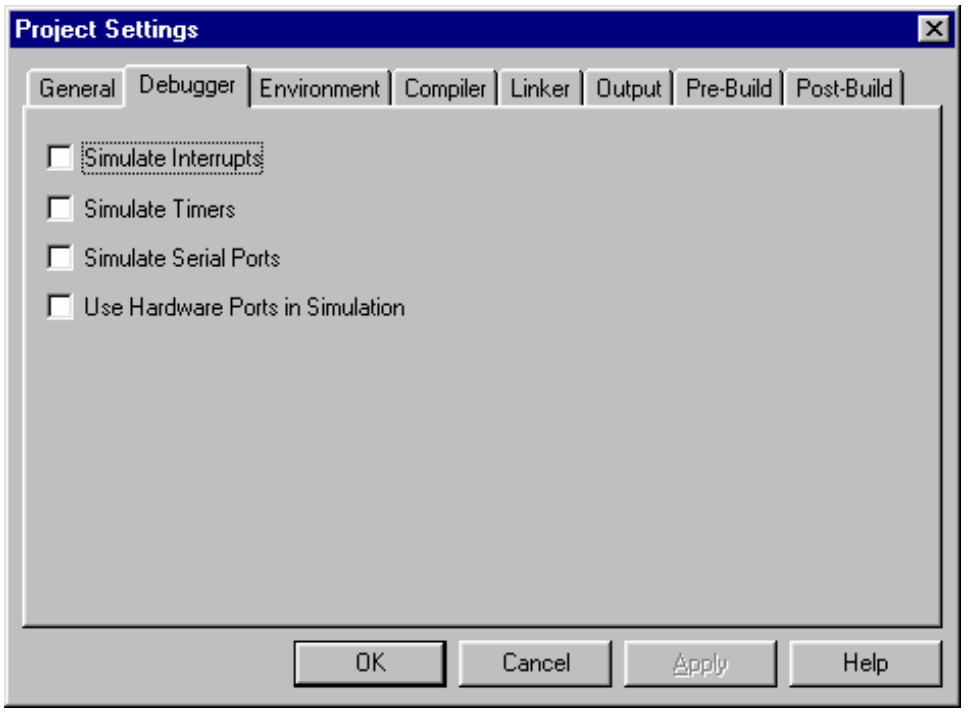
68HC11 Simulator is part of the 68HC11 Development System that can be downloaded from http://www.bipom.com/6811dev.htm

68HC11 Simulator supports both assembly and C source level simulation (using Micro C compiler).

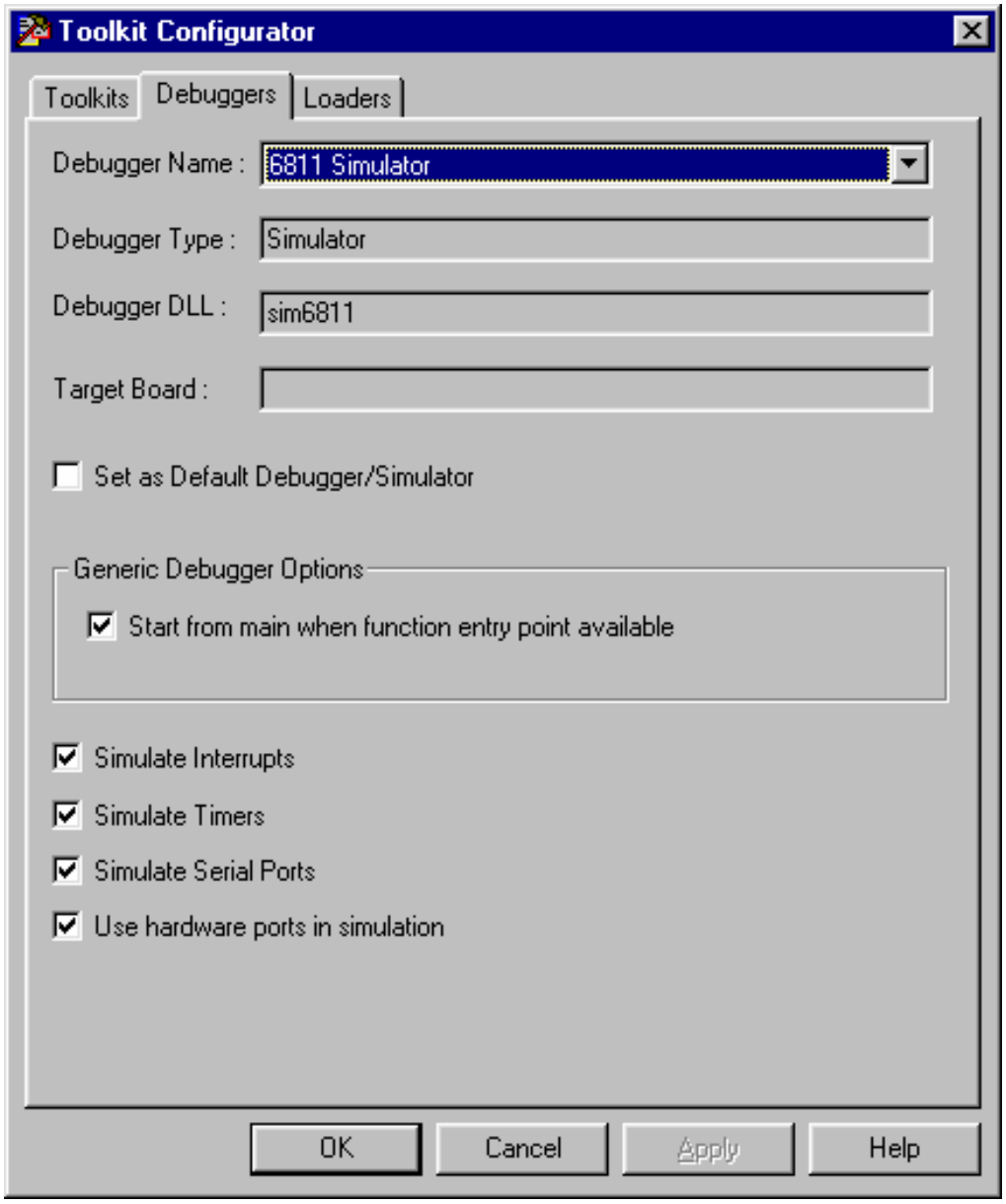
2. Preparation for simulation

Some example projects that come with Micro-IDE have settings to enable the simulator.

New projects are also created with the simulator/debugger options enabled. For every Micro-IDE project you can specify individual simulator/debugger settings using "Settings" command under "Project" menu. Simulator/debugger options are on the debugger tab:



Default simulator/debugger settings can be set using Toolkit Configurator (which can be started from Programs->Micro-IDE->Toolkit Configurator under the Start menu):



If you have older projects that do not have the simulator/debugger options, you can enable simulator/debugger support by adding command line options. These options are tool-specific.

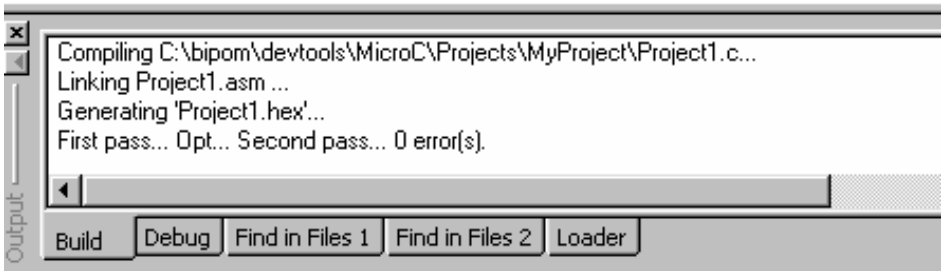
For example, for Micro C compiler, use the following:

Under Project->Settings:

- On the compiler command line, add **-C -S -L**
- On the Output Generator command line, add **-F -S**

These settings apply to Micro C compiler only. These settings will generate a listing file (.LST) which contains the information needed for the simulator/debugger.

Press “Build All” icon button on toolbar of Micro-IDE and check Output\Build window.



3. Starting and Stopping the 6811 simulator

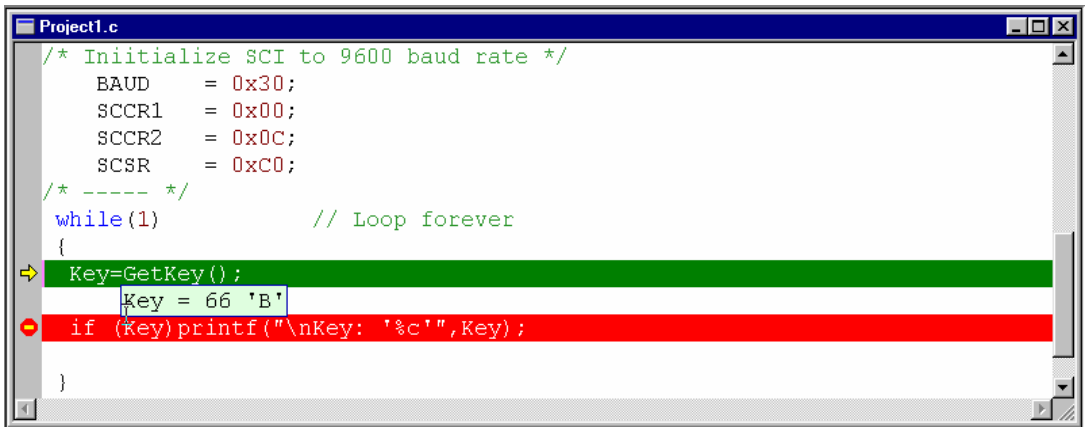
To start simulator, use "Go" button (F5), "Step into" button (F11), "Step Over" button (F10) or "Step out" button. To stop the simulator, you can use "Stop Debugging" button (Shift+F5).

4. 6811 simulator windows

Simulator uses several windows to display the debug information: Source Window, Output Window, Registers Window, Memory Window, Variables Window and Terminal Window.

4.1. Source Window

Source window is the window where you can create and edit your programs. Source window is also used by the simulator to set/remove/display breakpoints and current execution line.



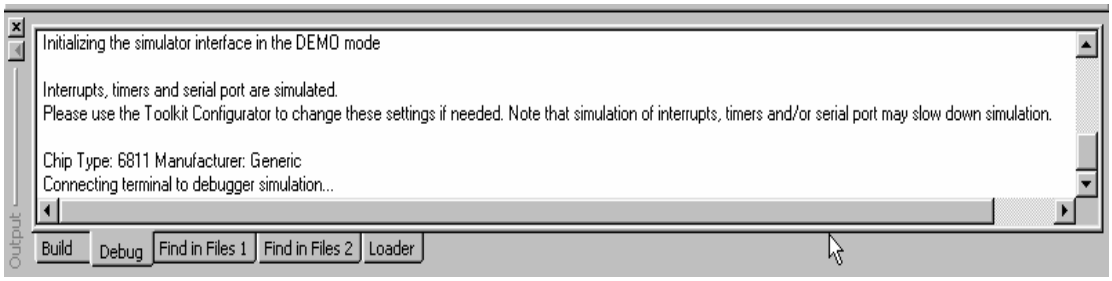
In debugging mode, the line that is about to be executed is colored green and contains a small yellow arrow on the left side.

You can toggle a breakpoint using F9 key. The line with an active breakpoint is colored red.

When debugging, you can view a variable value by simply placing mouse cursor over variable name.

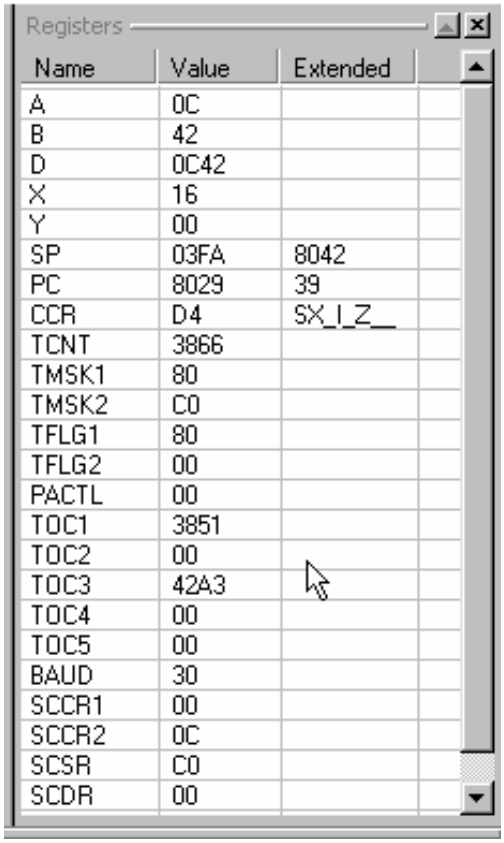
4.2. Output Window

Output Window displays messages about the current debugging session. To view debug messages, select the Debug tab on the Output Window. Any errors and information messages encountered during debugging are displayed here.



4.3. Registers Window

Register window displays the contents of the registers of the processor being debugged.

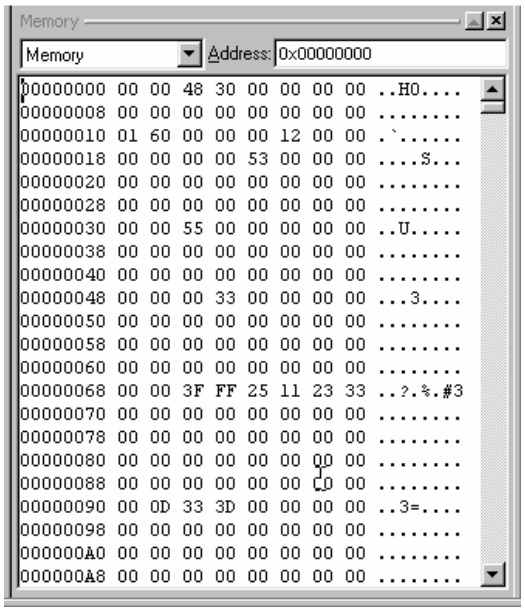


Register Window is updated as you single-step through your program. If your program is running continuously (for example, using the Go command), then the Register Window is updated when the simulated program stops due to a breakpoint or other cause.

You can edit register contents in this window by simply typing new values in the window cells.

4.4. Memory Window

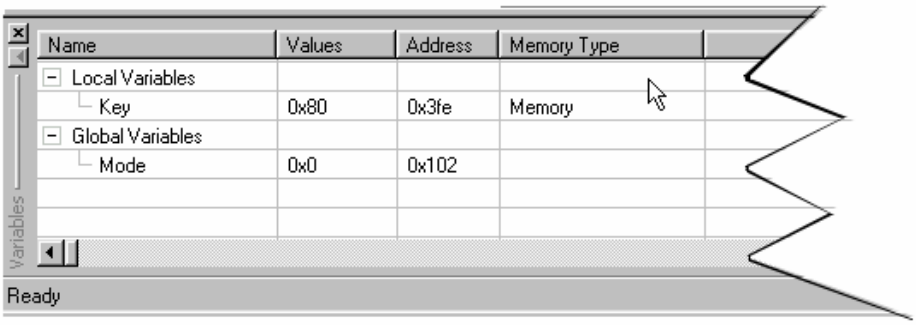
Memory window displays the contents of the memory of the target processor. Memory Window is updated as you single-step through your program. If your program is running continuously (for example, using the Go command), then the Memory Window is updated when the simulated program stops due to a breakpoint or other cause.



You can edit memory contents in this window by simply typing new values in the window cells.

4.5. Variables Window

Variables window displays contents of the global and local variables during debugging.



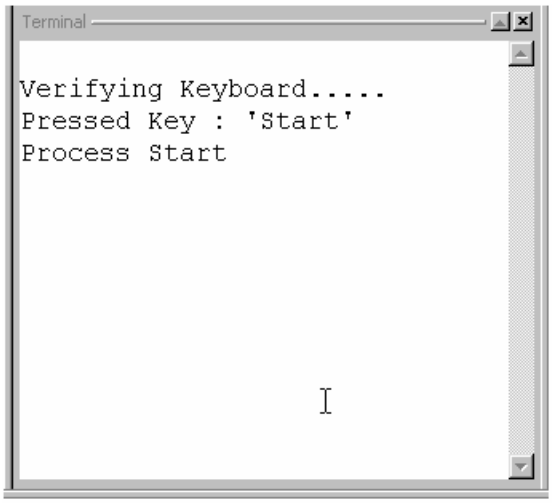
You can edit variables in this window by simply typing new values in the window cells.

Right-click on a variable and click "Extend" to edit variable in binary mode.

Right-click in this window and click "Hexadecimal display" to view variable contents in decimal or hexadecimal format.

4.6. Terminal Window

Terminal window is used for exchanging information with RS232 serial port of the simulated processor. Terminal window displays information that is sent to the serial port (for example, using C functions such as **printf**). Terminal window also accepts information from the keyboard and sends this information to the serial port of the simulated processor (for example, using C functions such as **getc** or **getch**).



5. Examples for use Smi6811 Simulator.

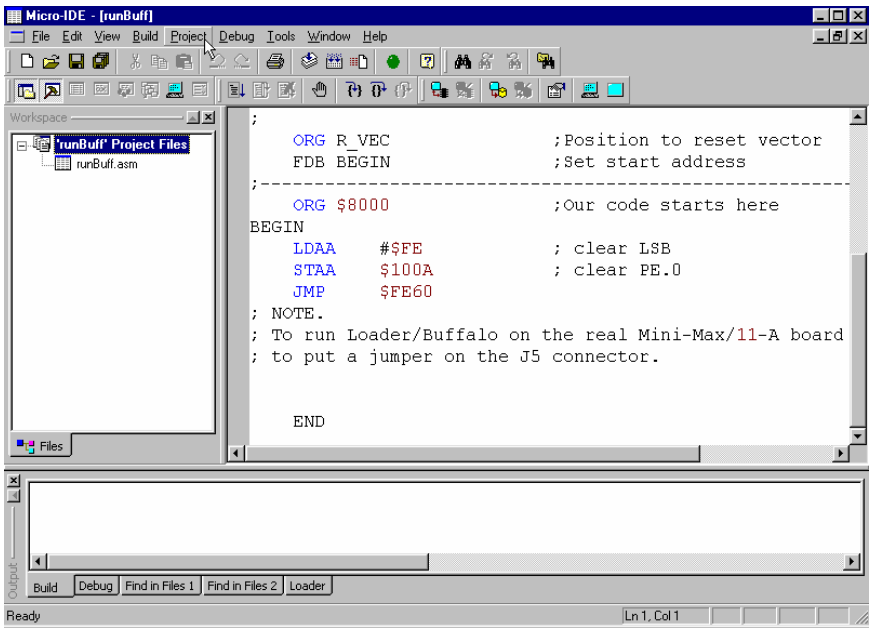
With these small examples you can quickly get acquainted with the Sim6811 simulator and ways to use it for the program execution tracing, timer system tracing and interrupts.

5.1. Run Loader/Buffalo.

Sim6811 not only simulates your projects but executes everything as real MINI-MAX/11-A board. Because the real board has pre-programmed BUFFALO and LOADER programs the simulator downloads and executes both these programs too. You can start these pre-programmed programs using the RunBuff program.

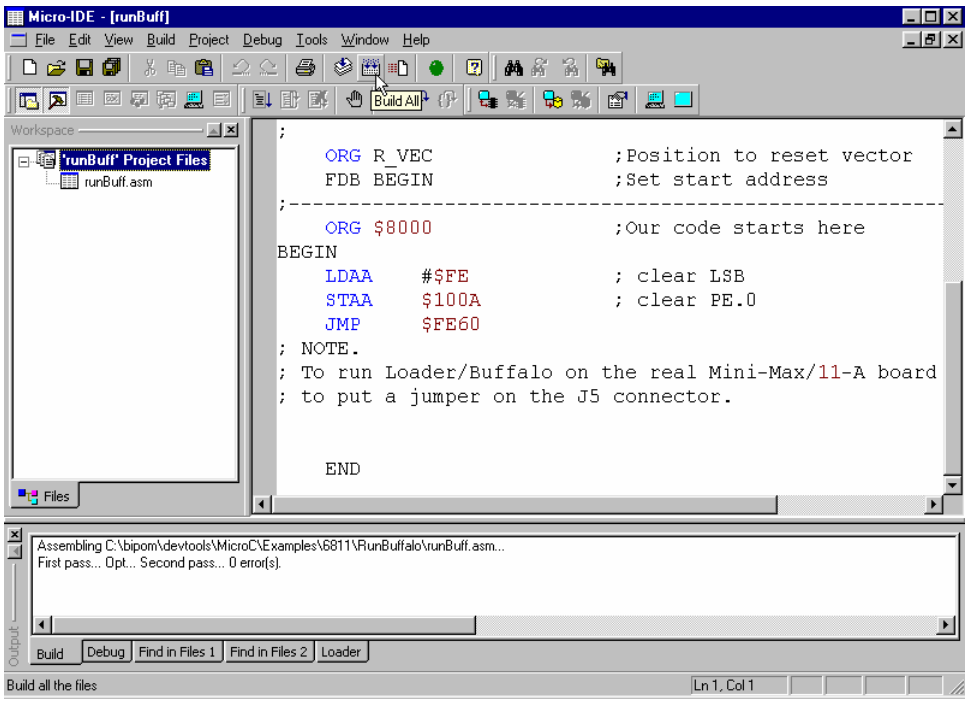
5.1.1. Step 1 – Open the project.

Open the project "RunBuff .prj" using "Open Project" command under "Project" menu.



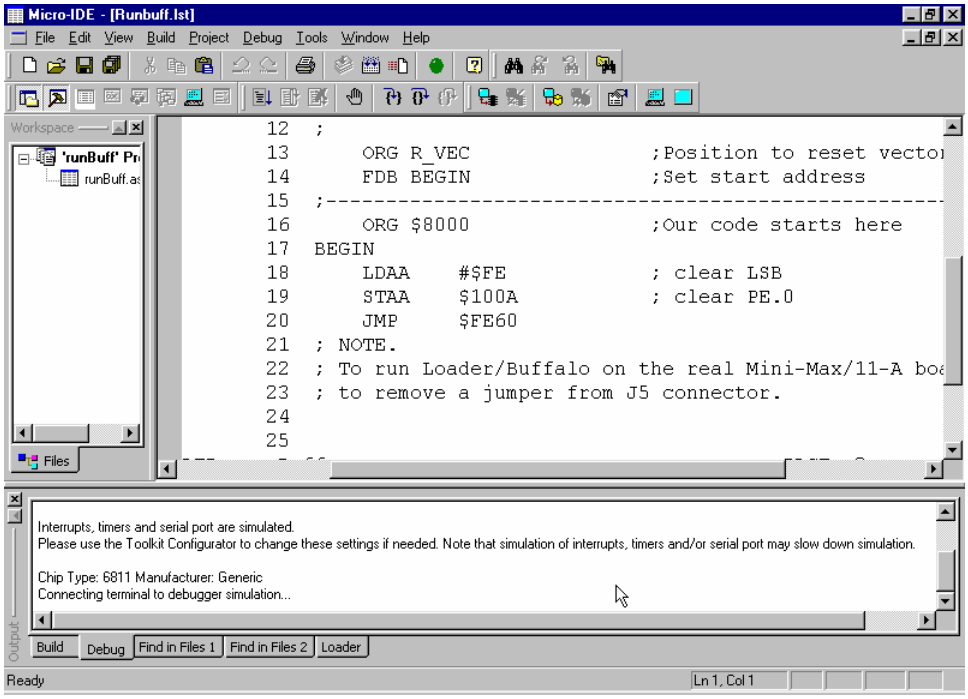
5.1.2. Step 2 – Build the project.

Press “Build All” icon button on toolbar of Micro-IDE and check Output\Build window.

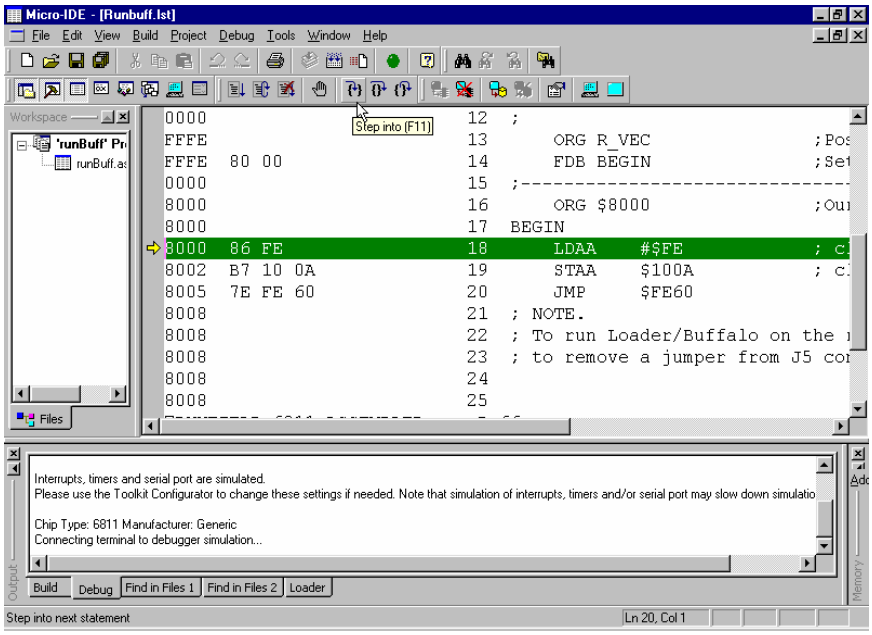


5.2.3. Step 3 – Start simulator.

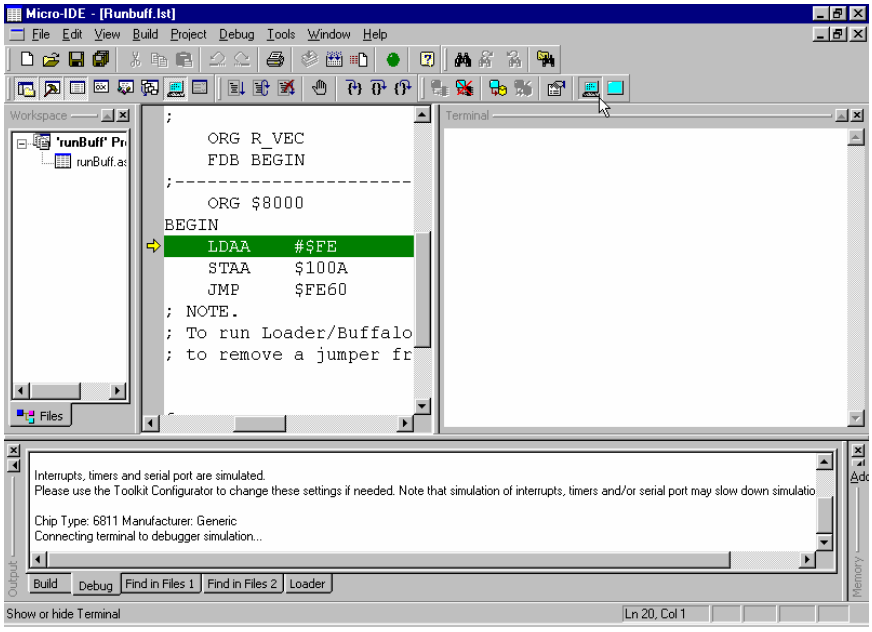
Open 'RunBuff.lst' file and trace step by step executing the program using "Open ..." command under "File" menu.



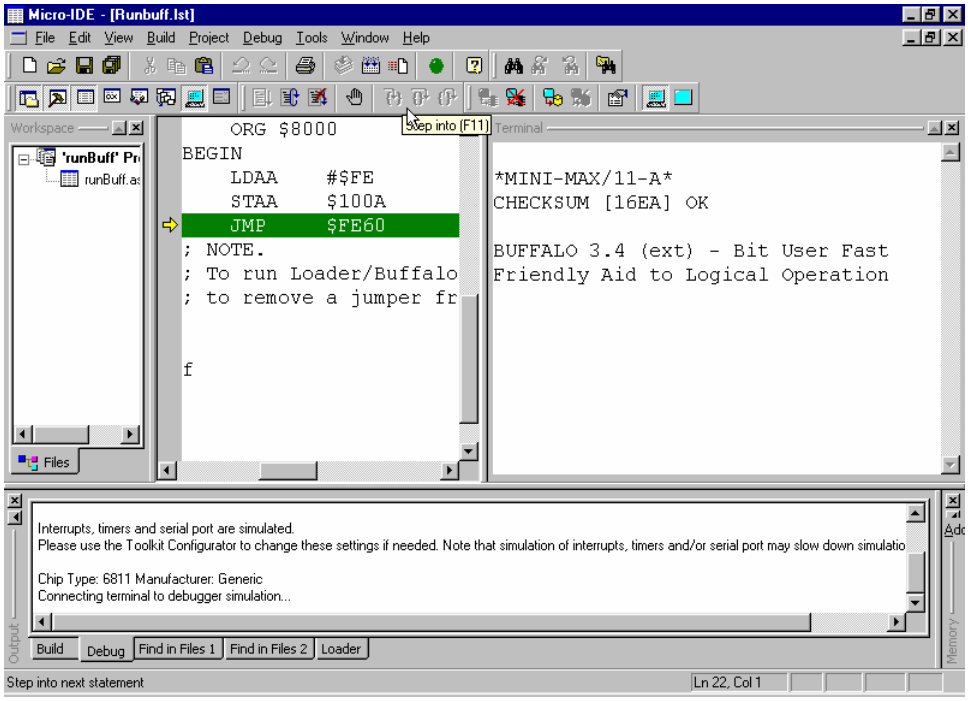
Start the simulating process by clicking "Step Into" command under "Debug" menu (or F11) a few times. Please note, simulator should step through some startup code (a few instructions of loader/buffalo) on the entry. Then, one of the lines will be colored green. Simulator is stopped at this line. It will be executed when you press one of the “Step Over”, “Step Into”, etc. buttons.



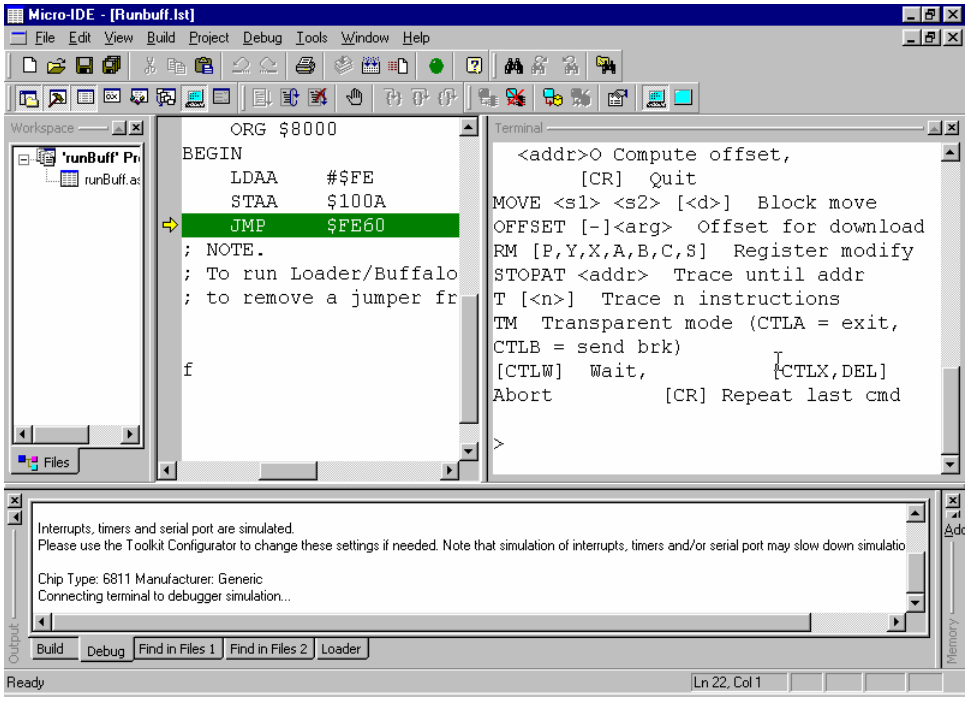
Open the Terminal window, using "Terminal" command under "View" menu.



The first two lines after BEGIN (LDAA #$FE and STAA $100A) are used to clear PORTE.0. This corresponds putting a jumper on J5 (1-3) connector on real board. The next line (JMP $FE60) gives control to Loader/Buffalo.



Now you can send any command to Buffalo. For example, you can write '?' (HELP) and confirm with <ENTER> button in Terminal Window.



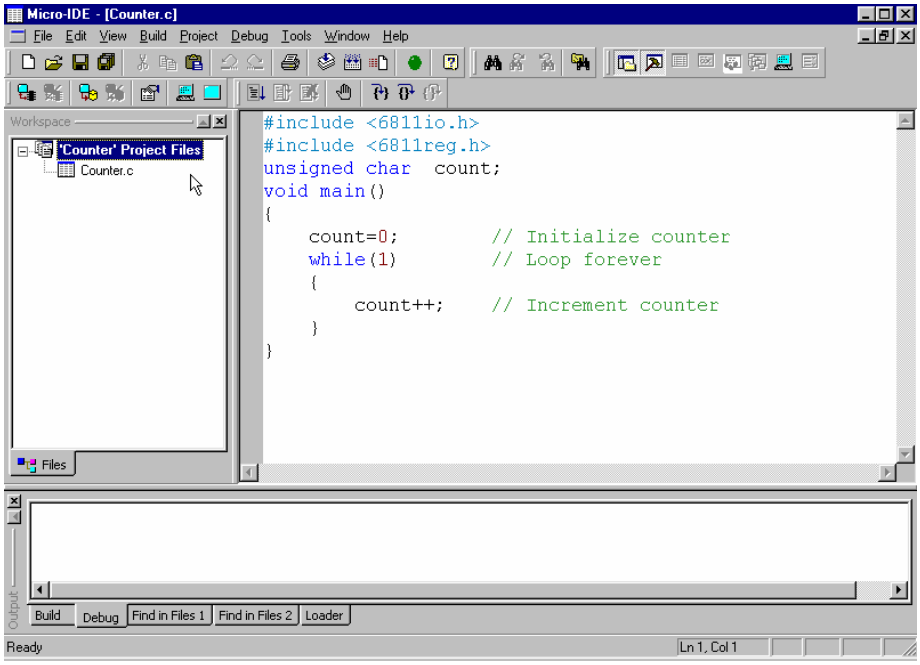
NOTE: Because Loader and Buffalo are not included in your project, you can't trace both them step-by-step executing.

5.2. Counter.

The example increments a variable in a loop. You will learn how to watch variable increment using Micro-IDE windows.

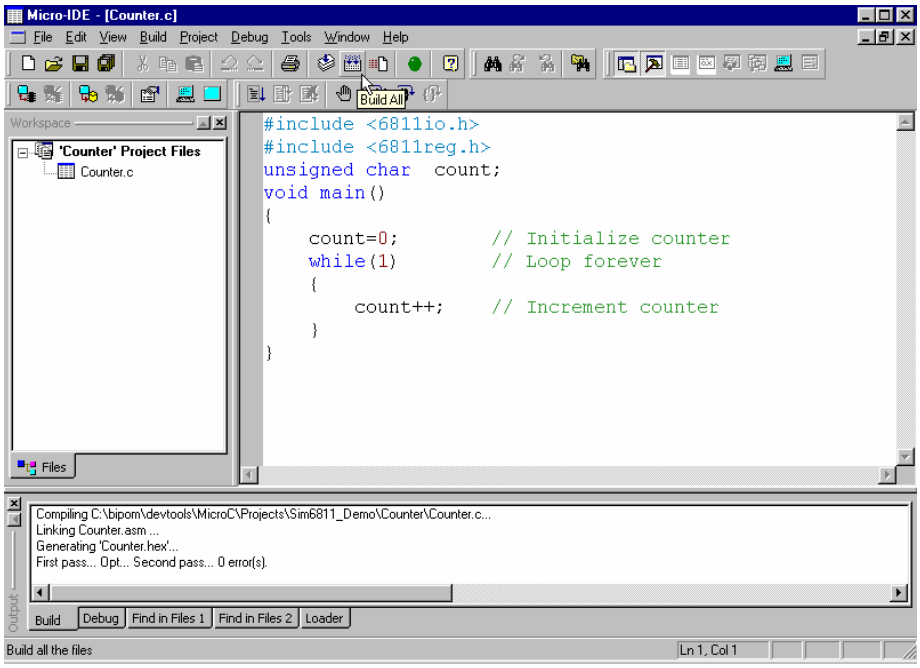
5.2.1. Step 1 – Open the project.

Open the project "Counter.prj", using "Open Project" command under "Project" menu.



5.2.2. Step 2 – Build the project.

Press “Build All ” icon button on toolbar of Micro-IDE and check Output\Build window.

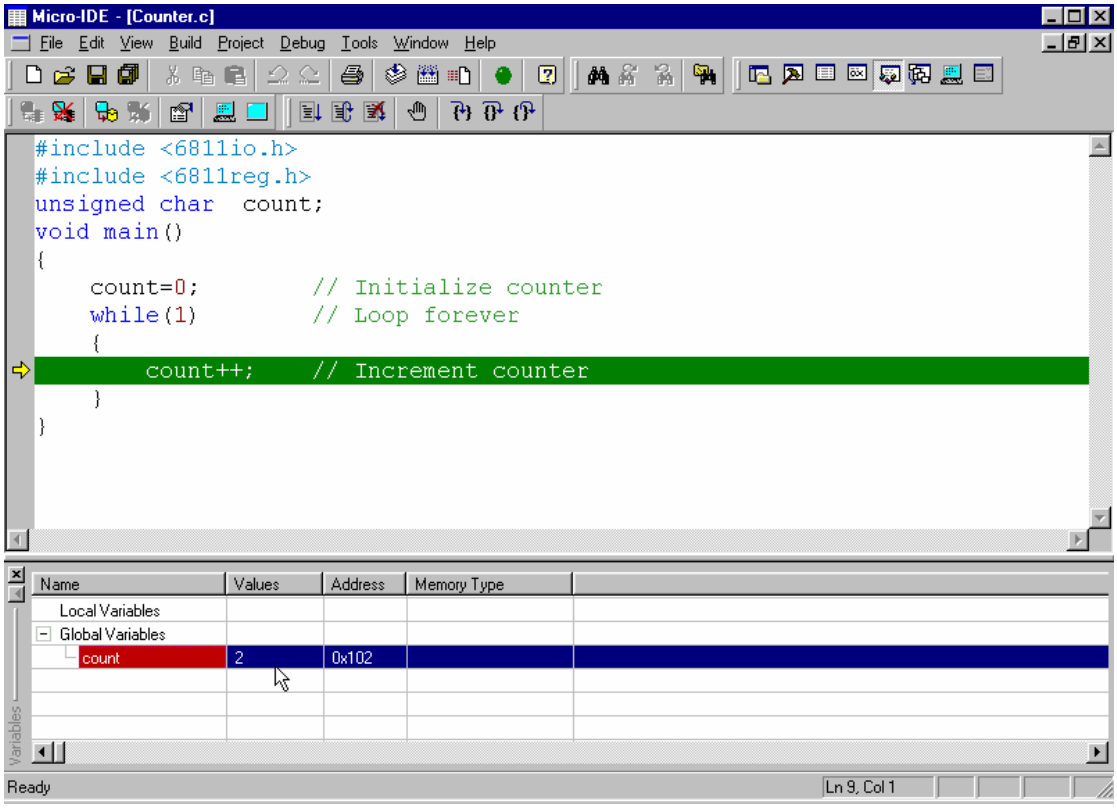


5.2.3. Step 3 – Start simulator.

Start the simulating process by clicking "Step Into" command under "Debug" menu (or F11). One of the lines will be colored green. This is the line the simulator stopped at. It will be executed when you press one of the “Step Over”, “Step Into”, etc. buttons.

5.2.4. Step 4 – Watch variable.

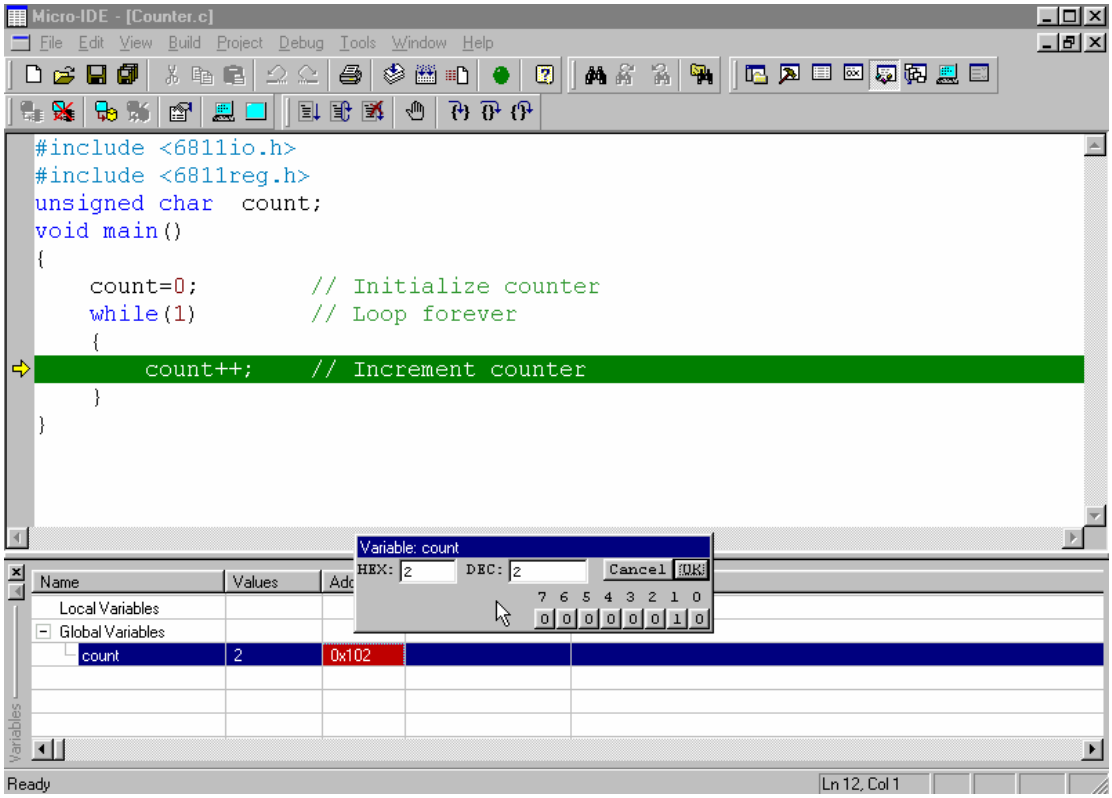
Open Variables Window using “Variables” command in “View” menu. In Variables Window you will see one global variable “count”. Run the program by clicking “Step Into” command (or F11) and you will see how “count” variable value is changed.



Please note that “Address” column contains “0x0102” value. It is the address of the variable in memory. On the next steps you will see that you can watch the variable in the memory window directly.

5.2.5. Step 5 – Change variable value.

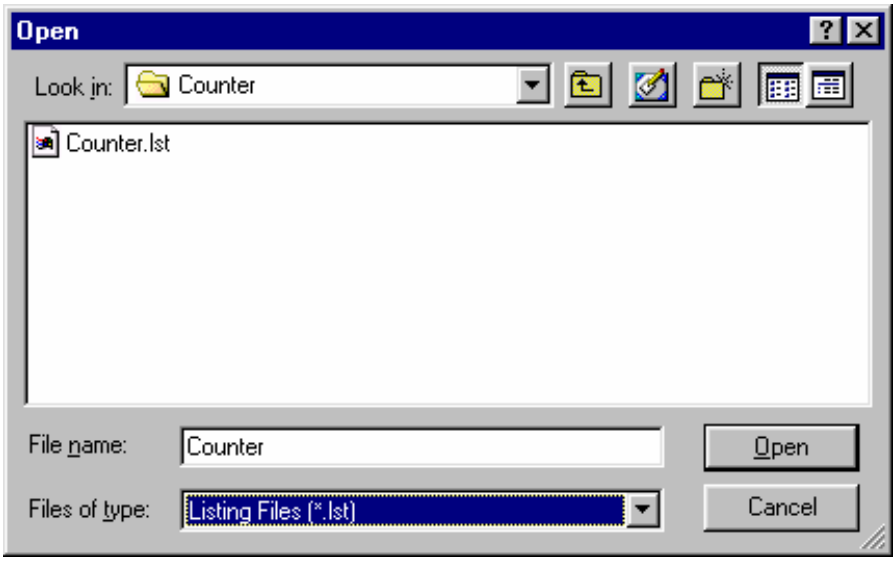
You can change value of the variable editing value in Values column. You can also view and change bits of the variable value. Double-click on the variable row in the Variables Window or select “Extend” command from context menu. A popup window will appear:



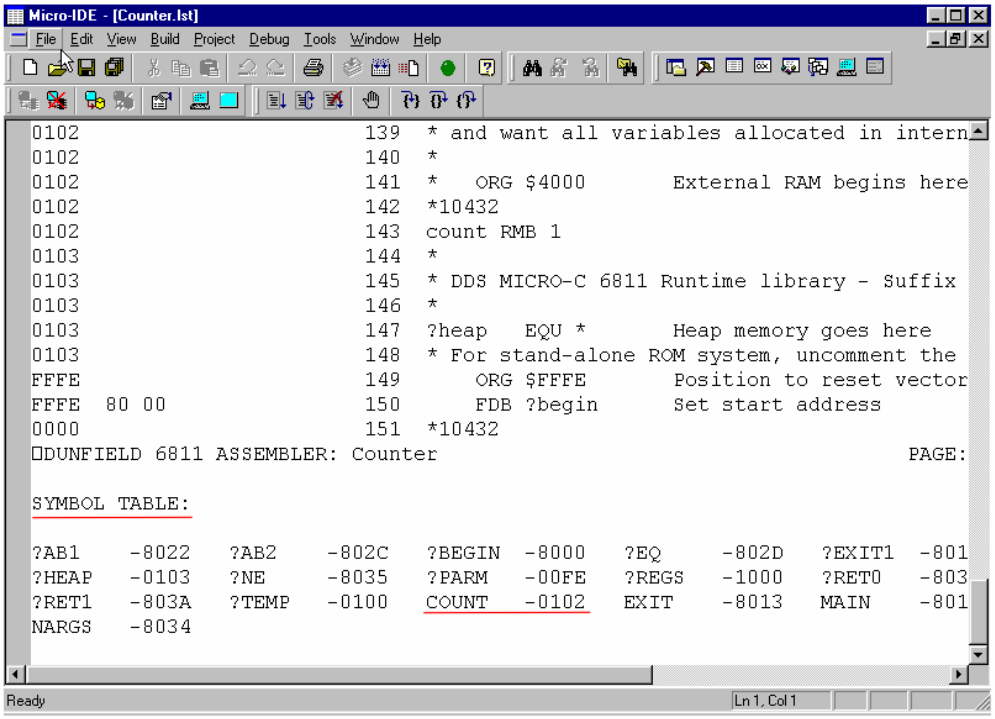
You can edit value in binary, HEX or DEC here.

5.2.6. Step 6 – Variable address in .LST file.

Open the listing file "Counter.lst" using "Open …" command under "File" menu.



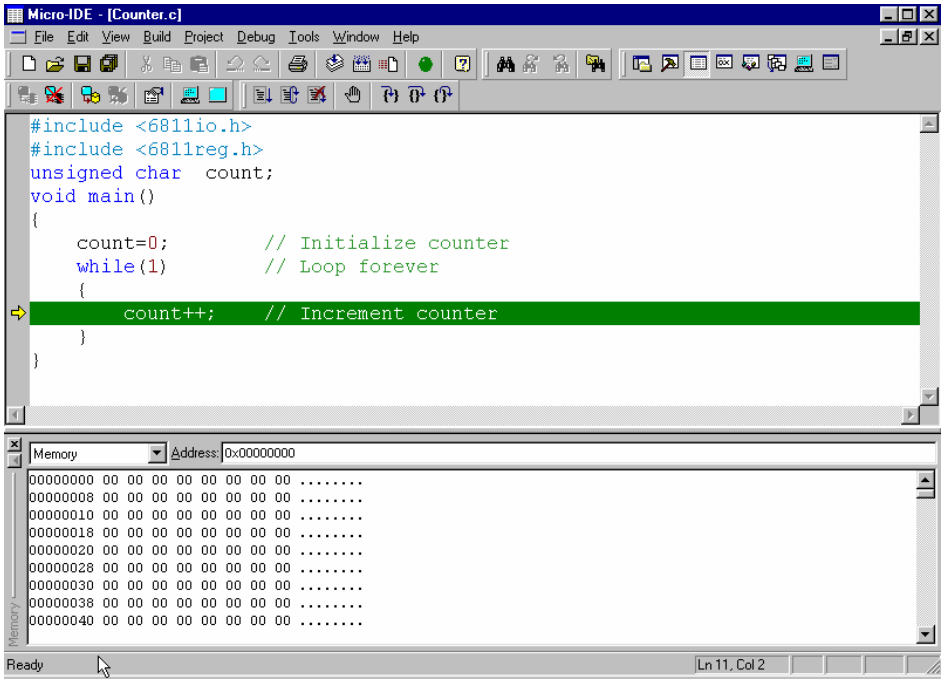
In the end of the listing file find 'SYMBOL TABLE :'. Here you can see address in RAM of the 'count' variable (for this example, 0102). It is the location Micro-C placed this variable to.



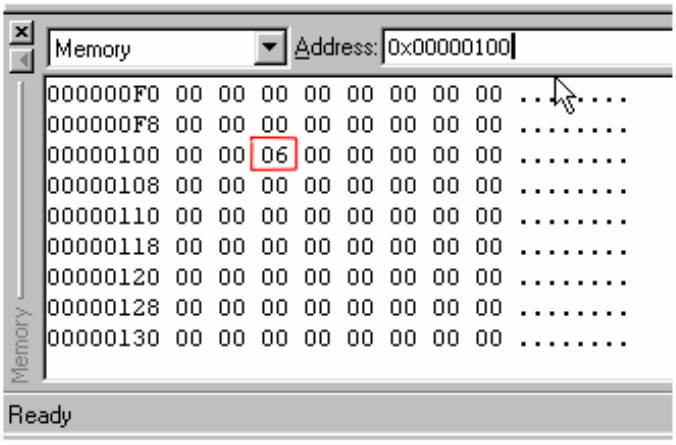
Close the Listing File, using the <Close> button. You will see C-source view again.

5.2.7. Step 7 – Variable in the Memory Window

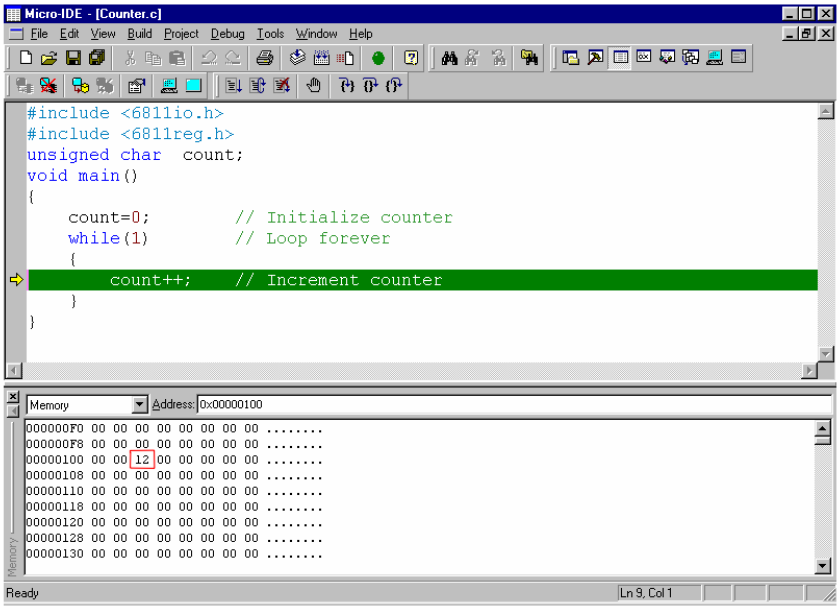
Open the Memory Window, using "Memory" command under "View" menu.



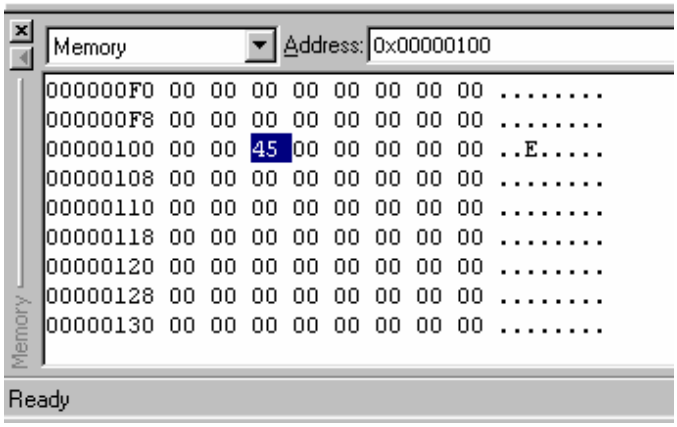
In field "Address:" enter 0x00000100 value and press Enter. This will scroll Memory Window to address 0x0100.



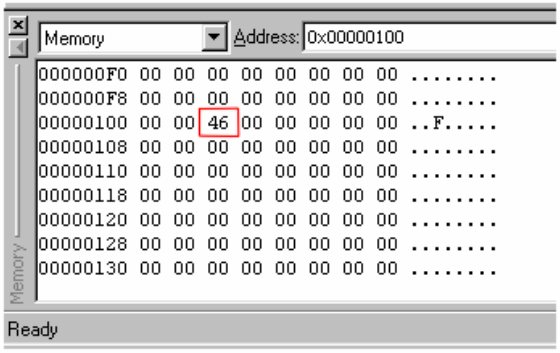
Now you can see 0102 memory location which is address of the “counter” variable. Use F11 button to step through the program and you will see that value at 0102 address is changed.



Also, you can change the 'count' variable value directly in memory. For this purpose, you can mark and enter new value (for example 45).



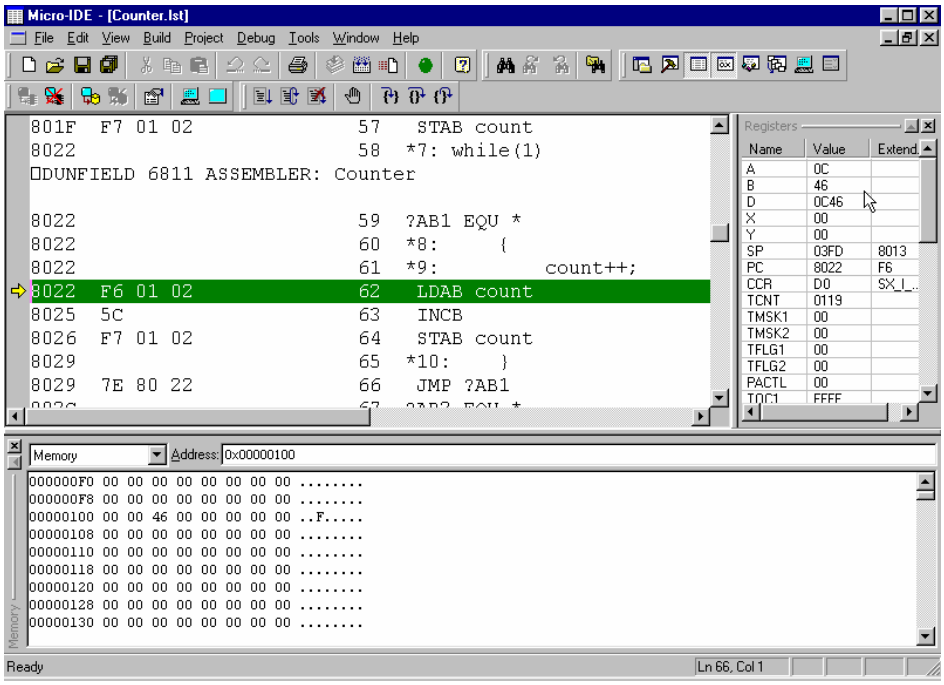
Of course, you can continue program executing, after changing the variable value in memory, use the F11 button.



5.2.8. Step 8 – Disassembly.

In this step we'll trace how CPU changes the variable value using the registers. First, you have to open disassembly window by clicking "Disassembly" command under "View" menu.

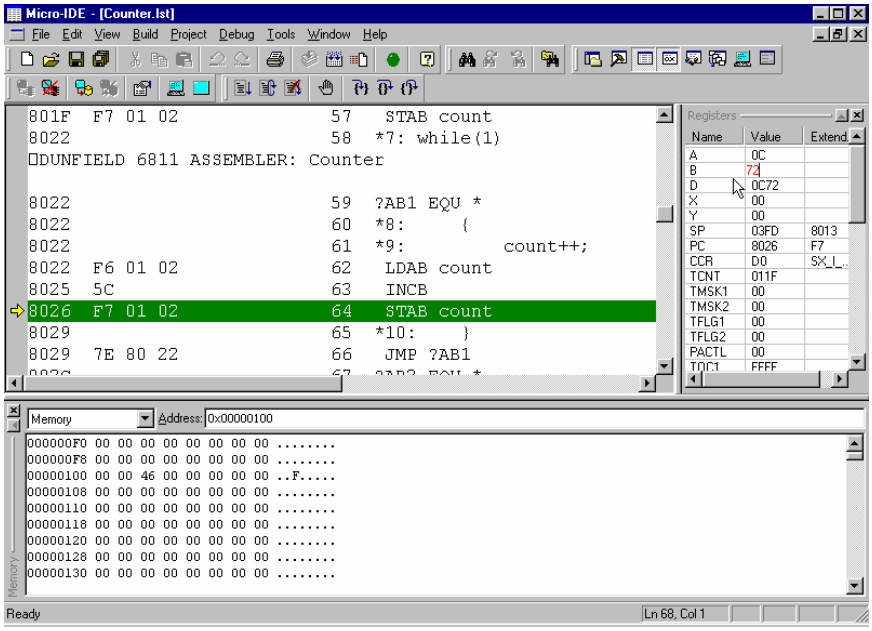
In this window, you can see and trace the program in the assembly and code view. After that, you have to open registers window, using "Registers" command under "View" menu.



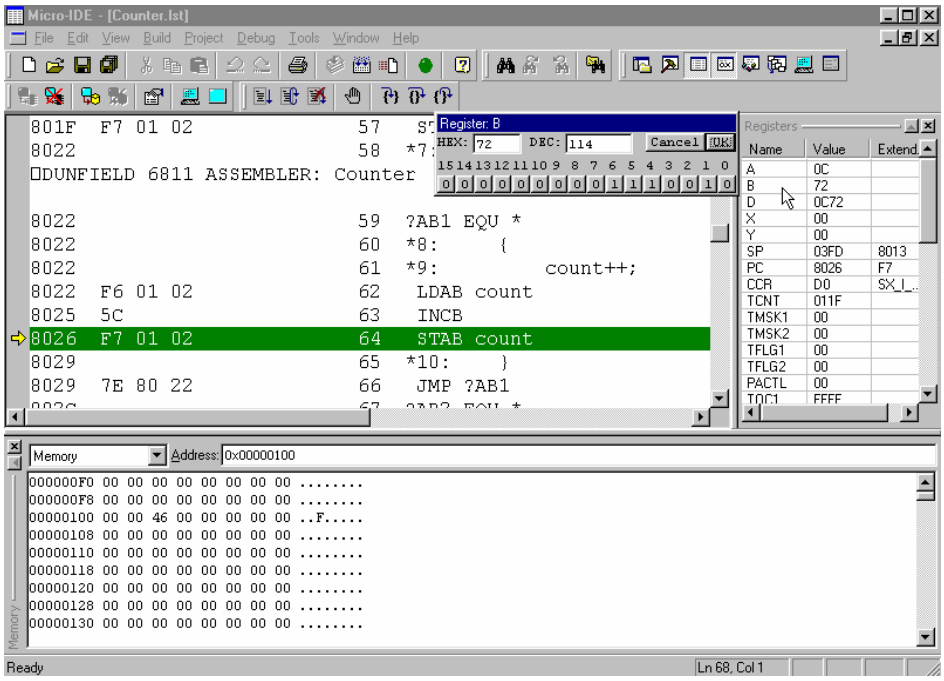
In disassembly view you can see how CPU loads the accumulator 'B' from 'count' ("LDAB count" instruction), increments the accumulator 'B' ("INCB" instruction) and stores the accumulator 'B' in 'count' ("STAB count" instruction). You can step through these instructions using F11 button.

In Registers Window you also can change the accumulator 'B' value (for example, to 72). To do it you have to click on the Value column, enter new value, and press 'ENTER' button.

(NOTE: you have to change accumulator 'B' value after "INCB" instruction and before "STAB count" instruction.)



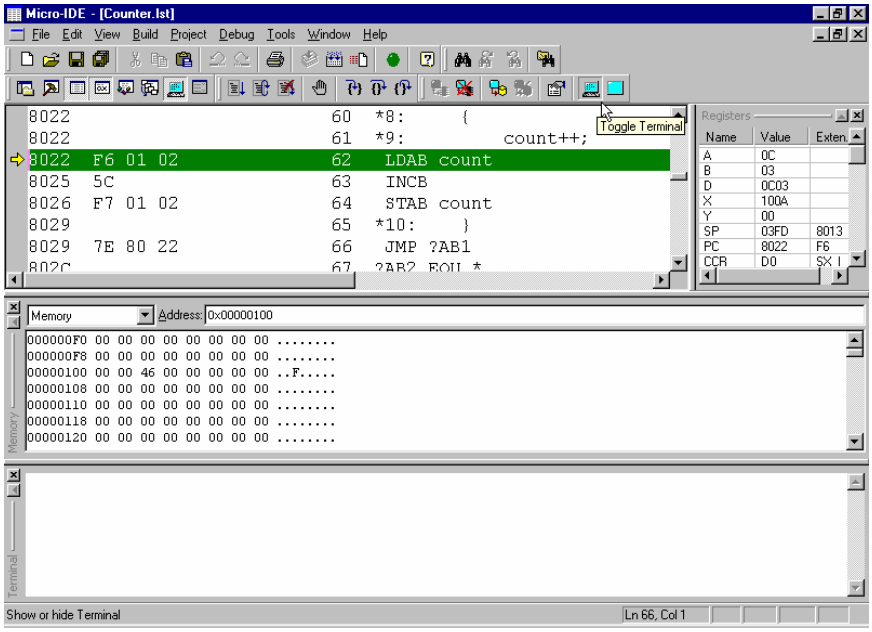
If you want to see accumulator 'B' (or other register) contents in binary mode, you can click the right mouse button, after positioning the cursor over the accumulator (or other register) name (for example, 'B').



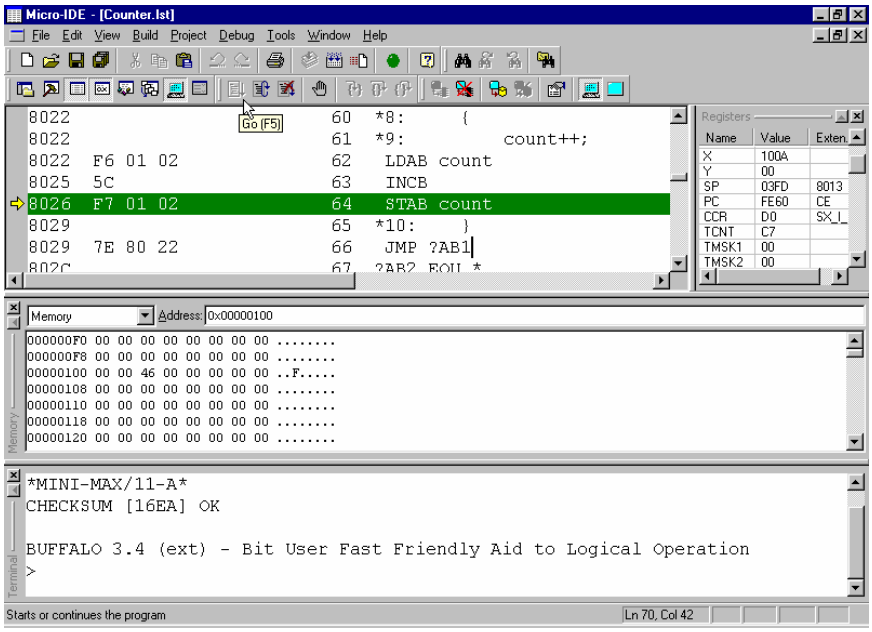
A popup window appears. In this window you can change the register value in binary, decimal or hex format. After you enter needed value you can confirm by clicking 'OK' button or cancel the changes with 'Cancel' button.

5.2.9. Step 9 – Using Buffalo and Loader.

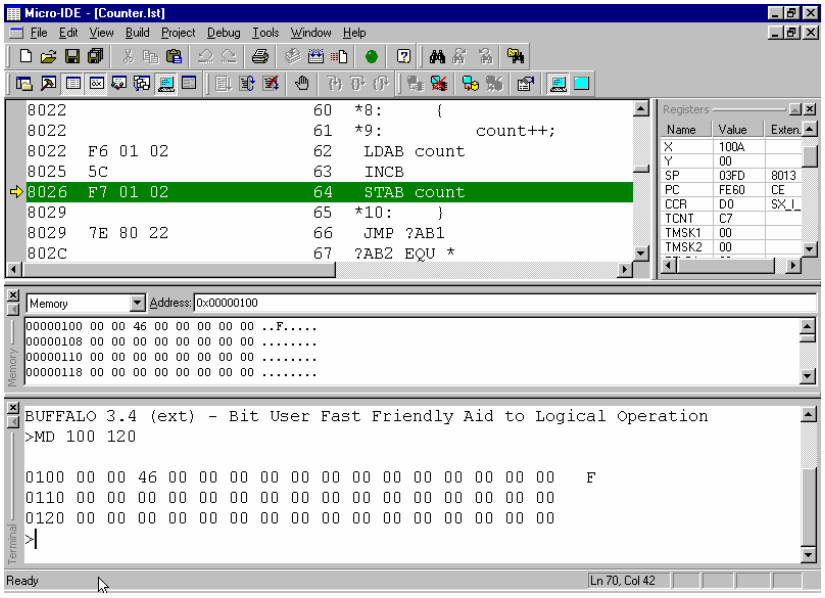
You can break executing your program and start Buffalo and Loader at any time. For this purpose you can open the Terminal window using "Terminal" command under "View" menu.



After that you must set PORTE=0xFE and PC=0xFE60 using the Register Window and start the program using <GO> button (F5).

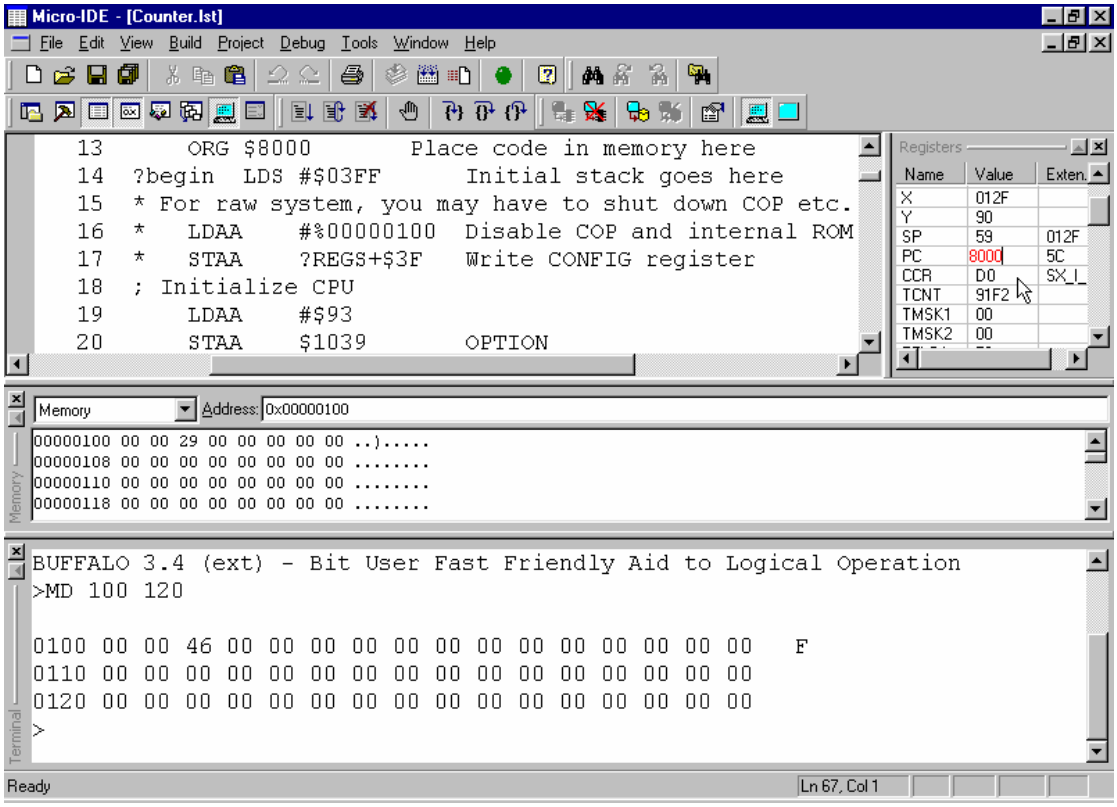


Now, for example, you can see memory area in Terminal Window using the Buffalo command:
MD 100 120.



Of course, you can go back to your program at any time, but you must break non-stop executing the Buffalo monitor program using "Break" command under "Debug" menu.

Then, you can start your program again from every address but please have in mind the accumulator, registers, Program Counter, and Stack Pointer are random. The correct way is to run your program from address $8000 or manually set up all registers to correct values.



5.3. Timer.

This example demonstrates simulation of timers and interrupts.

First, this program prepares RAM vectors for interrupts (if you are using Buffalo) and synchronizes the TCNT value using the Timer Overflow interrupt (to provide equally executing on real board and simulator). Then, it sets up timer output compare 1 (TOC1) and timer output compare 2 (TOC2) to generate regular interrupts. When interrupt occurs the corresponding interrupt service routine (ISR) sends 'TOCx' string to the serial port. When program waits for interrupts it sends char '.' to the serial port.

NOTE: When you begin executing this example, make sure that the file 6811int.h contains the following line (Fig.2). Use "Open" command under "File" menu (Fig.1) to view the file.

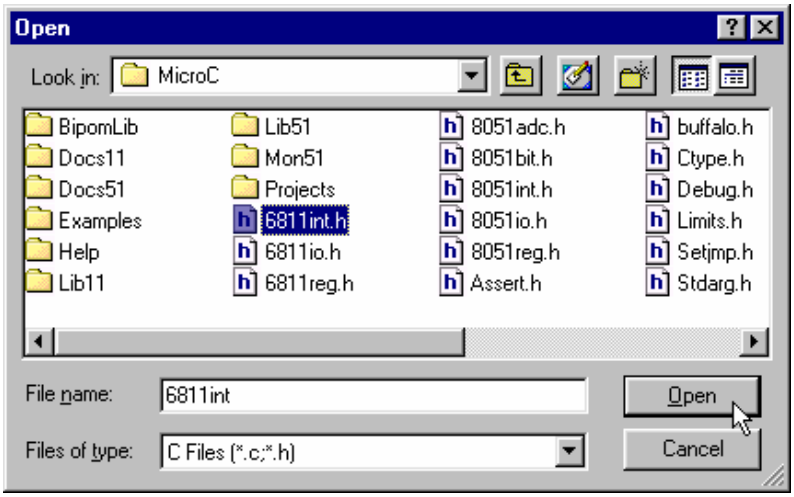


Fig.1

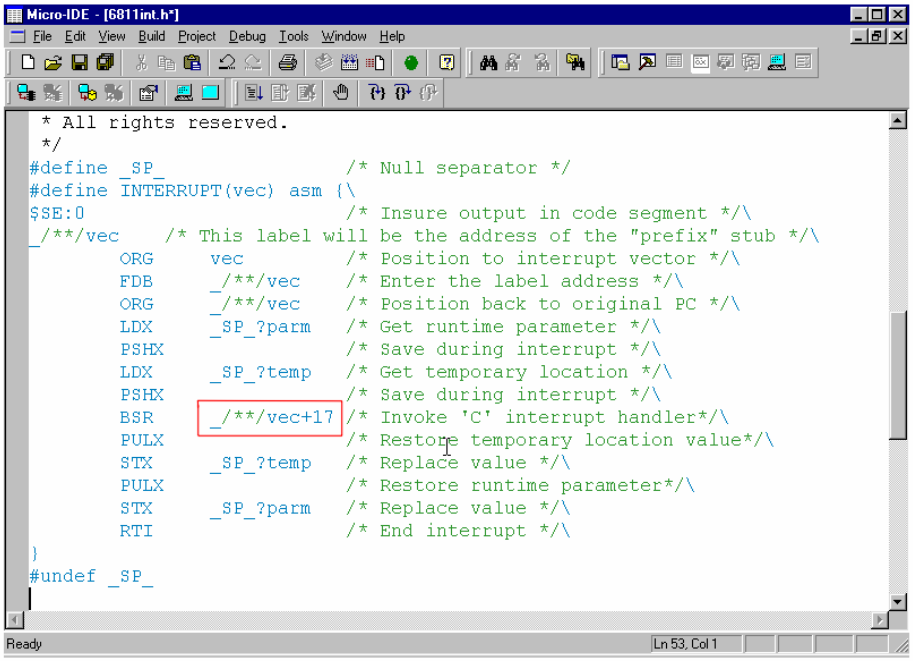
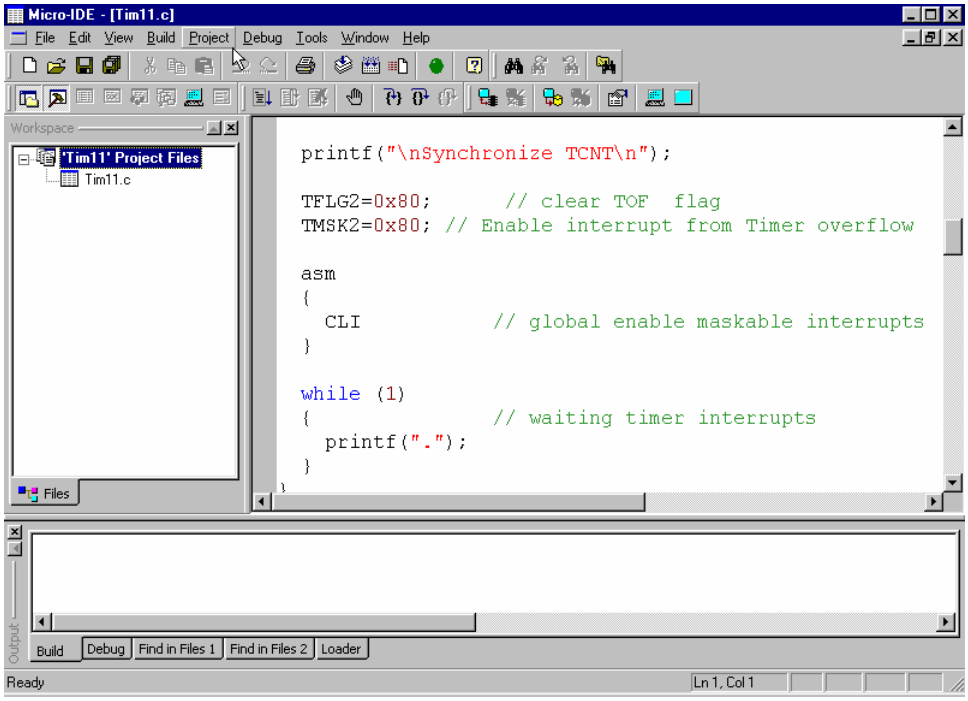


Fig.2

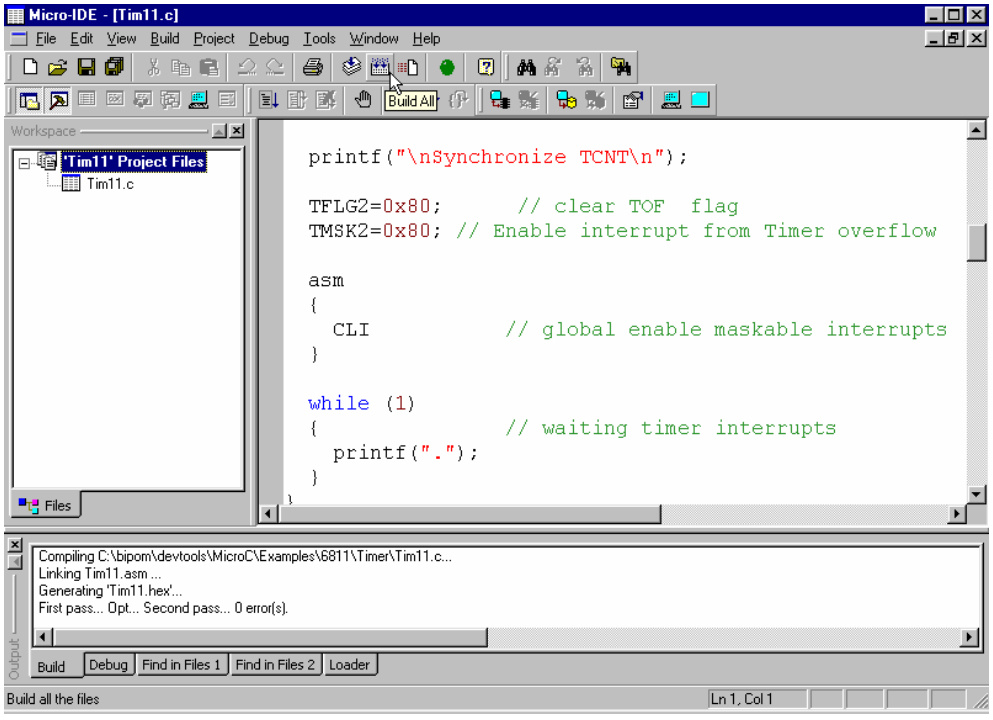
5.3.1. Step 1 – Open the project.

Open the project "Tim11.prj" using "Open Project" command under "Project" menu.



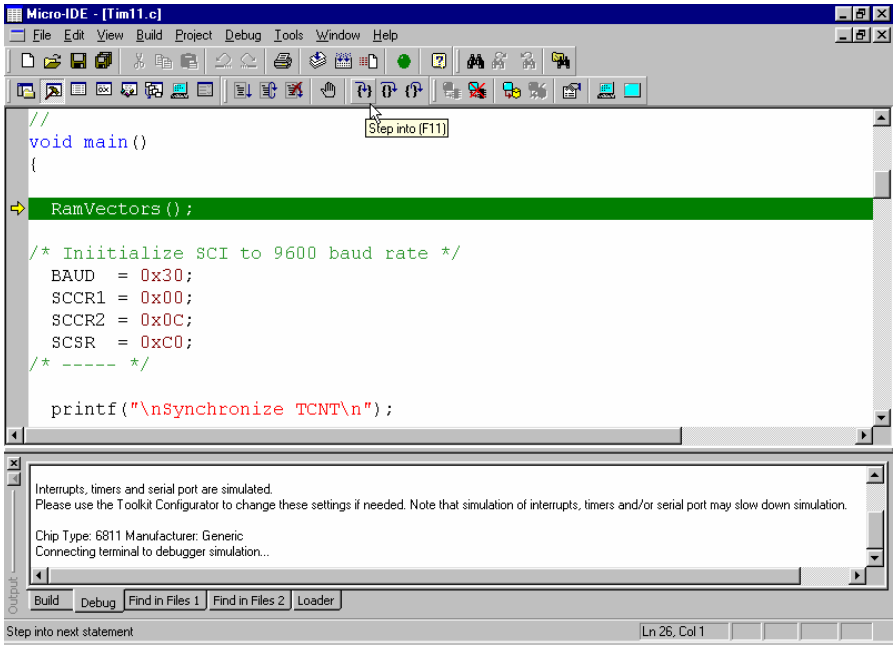
5.3.2. Step 2 – Build the project.

Click “Build All” icon button on the toolbar of Micro-IDE and make sure there are no errors reported in Output\Build window.



5.3.3. Step 3 – Start simulator.

Start the simulation using the F11 button.

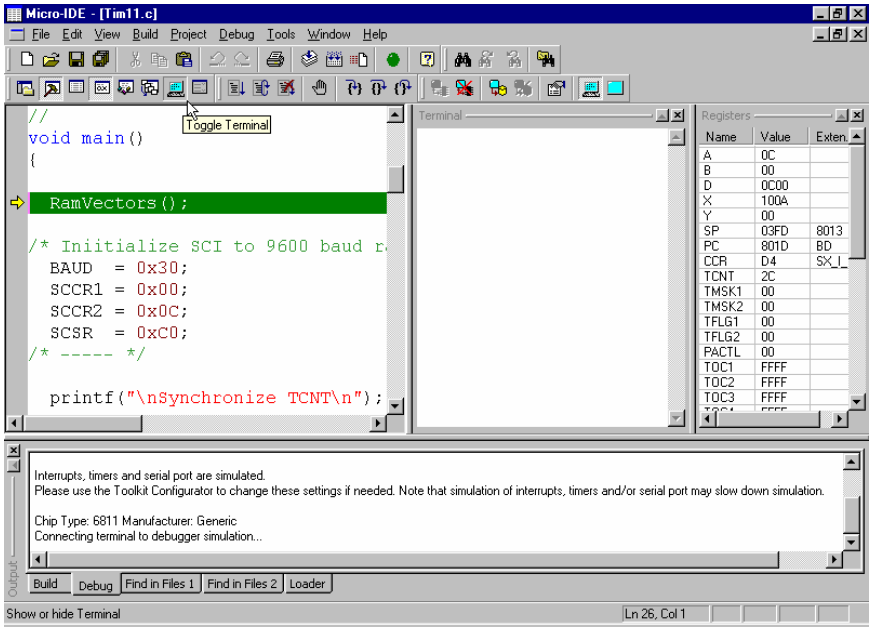


First of all, the program fills the JMP Table for interrupts (function RamVectors) because the Buffalo monitor redefines interrupt vectors.

VECTOR JUMP TABLE:

00DF:7E 81 15 JMP FUNC_1 (Interrupt from TOC1)
00DC:7E 81 27 JMP FUNC_2 (Interrupt from TOC2)
00D0:7E 81 39 JMP FUNC_3 (Interrupt from TOVF)

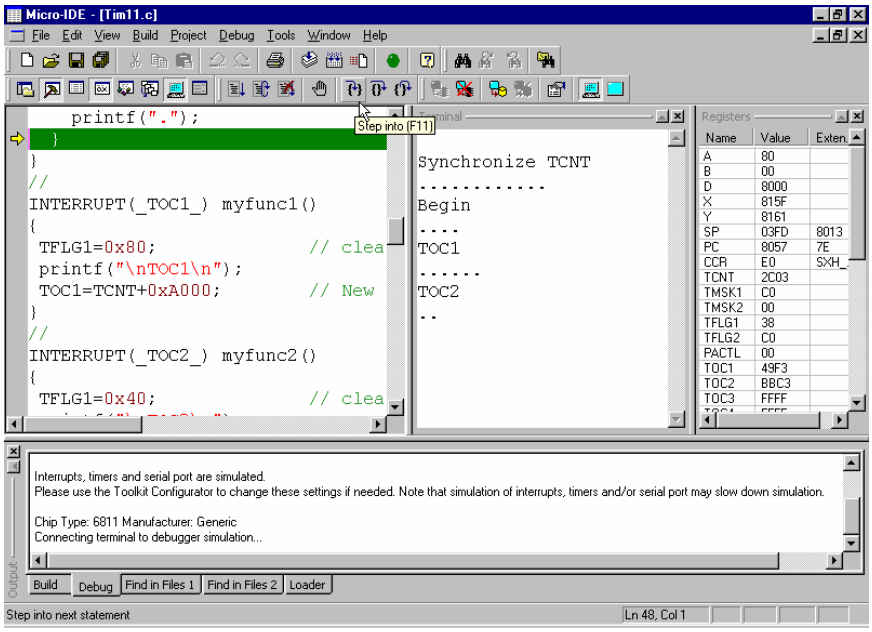
Now you can open the Registers and Terminal windows using “Registers” and "Terminal" commands under "View" menu.



5.3.4. Step 4 – Run the sample.

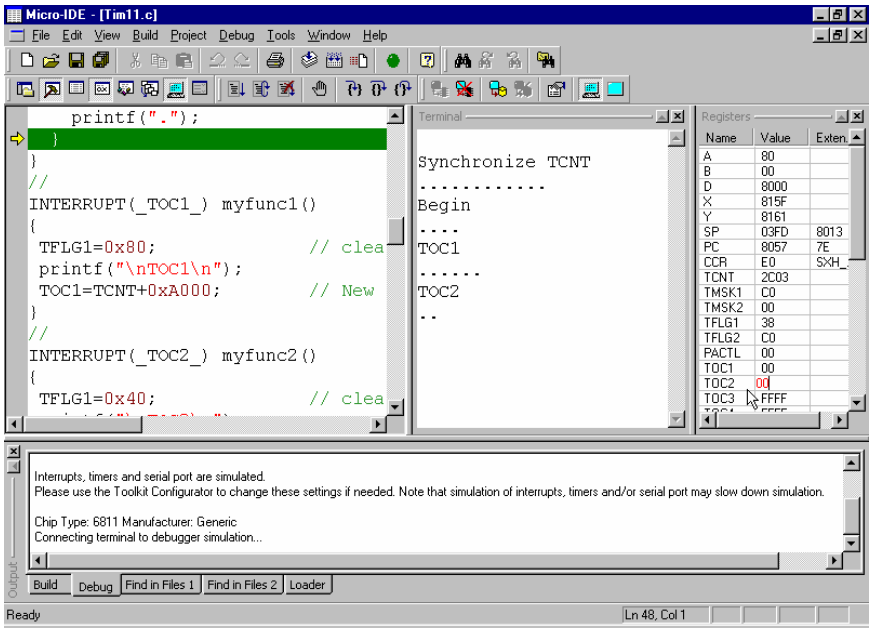
The program begins with synchronizing TCNT using the Timer Overflow interrupt (TOVF). When interrupt TOVF occurs the program disables TOVF interrupt and sets up timer output compare 1 (TOC1) and timer output compare 2 (TOC2) to generate regular interrupts.

Trace the synchronizing TCNT, main program execution and see how TOC1 and TOC2 interrupts are handled. Use F11 button to step.

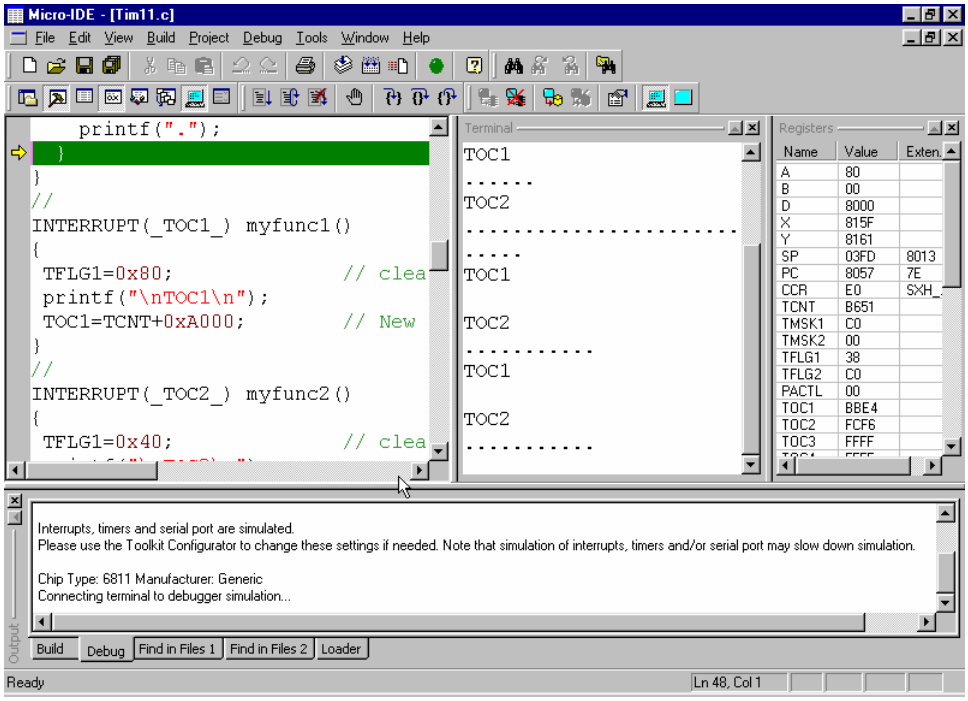


5.3.5. Step 5 – Changing registers values.

Now you can change the TOC1 and TOC2 value (for example, set both them to 0x0000) so that interrupts from TOC1 and TOC2 happen at the same time. To change values of the registers you can use the Registers Window.



Keep on pressing F11 and trace interrupts handling. Because TOC1 has higher priority level, these interrupts are executed first.



5.3.5. Step 5 – Running in non-step mode.

You can start the program using the F5 button or by clicking “Go” toolbar button to execute program continuously.