
Topview Debugger

Software User Guide

FRONTLINE
E L E C T R O N I C S

Copyright © 2002 Frontline Electronics Pvt Ltd. All Rights Reserved.

Information in this document is subject to change without notice. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose without the express written permission of Frontline Electronics Pvt Ltd.

FRONTLINE
ELECTRONICS

Topview Debugger Software User Guide

Contents

Chapter 1 - Introduction to Topview Debugger

| | | |
|-----|--|---|
| 1.1 | Introduction to Topview Debugger | 1 |
| 1.2 | System Requirements | 2 |

Chapter 2 - Connecting the Topview Trainer with the Host Computer

| | | |
|-----|---|---|
| 2.1 | Introduction | 3 |
| 2.2 | Connecting the Topview Trainer with the Host Computer | 3 |

Chapter 3 - Topview Debugger GUI Environment

| | | |
|-------|-----------------------------------|----|
| 3.1 | Introduction | 7 |
| 3.2 | Debugger Toolbar | 7 |
| 3.3 | Tool Bar Portion | 8 |
| 3.4 | ClearView Window Structure | 11 |
| 3.5 | Normal Window | 12 |
| 3.6 | Register Window | 12 |
| 3.7 | Program Window | 14 |
| 3.7.1 | Edit Instruction | 16 |
| 3.7.2 | Enter Program | 17 |
| 3.7.3 | Execute BreakPoint | 17 |
| 3.7.4 | Execute | 17 |
| 3.7.5 | Set Address | 17 |
| 3.7.6 | Set PC | 18 |
| 3.7.7 | Set BreakPoint | 18 |
| 3.7.8 | Remove BreakPoint | 18 |
| 3.8 | Internal Data Memory Window | 18 |
| 3.9 | External Memory Window | 19 |
| 3.10 | SFR Bit Status Window | 21 |
| 3.11 | Memory Bit Status Window | 21 |

Chapter 4 - Developing Target Program Code

| | | |
|-----|--------------------------------------|----|
| 4.1 | Introduction | 23 |
| 4.2 | Developing Target Program Code | 23 |

Chapter 5 - Loading a Program from Disk to the Memory of the Trainer

| | | |
|-----|--|----|
| 5.1 | Introduction | 27 |
| 5.2 | Loading a Program from Disk to the Memory of the Trainer | 27 |

Chapter 6 - Storing the Memory Contents to Disk

| | | |
|-----|---|----|
| 6.1 | Introduction | 31 |
| 6.2 | Storing the Memory Contents to Disk | 31 |

Chapter 7 - Using Single Line Assembler

| | | |
|-----|-----------------------------------|----|
| 7.1 | Introduction | 33 |
| 7.2 | Using Single Line Assembler | 33 |

Chapter 8 - Filling a Fixed Data in Internal/External Data Memory

| | | |
|-----|---|----|
| 8.1 | Introduction | 35 |
| 8.2 | Filling a Fixed Data in Internal/External Data Memory | 35 |

Chapter 9 - Copying Block of Data From One Location to Other Place

| | | |
|-----|--|----|
| 9.1 | Introduction | 37 |
| 9.2 | Copying Block of Data From One Location to Other Place | 37 |

Chapter 10 - Testing the Memory Area of the Trainer

| | | |
|------|--|----|
| 10.1 | Introduction | 39 |
| 10.2 | Testing the Memory Area of the Trainer | 39 |

Chapter 11 - Program Execution in Full Speed

| | |
|--|----|
| 11.1 Introduction | 41 |
| 11.2 Program Execution in Full Speed | 41 |
| 11.3 GoTo Command | 44 |

Chapter 12 - Program Execution Using BreakPoints

| | |
|--|----|
| 12.1 Introduction | 47 |
| 12.2 Program Execution Using BreakPoints | 47 |

Chapter 13 - Program Execution Using SingleStep Command

| | |
|---|----|
| 13.1 Introduction | 51 |
| 13.2 Program Execution Using SingleStep Command | 51 |
| 13.3 SingleStep Setting | 52 |

Chapter 14 - Program Execution Using StepOver Command

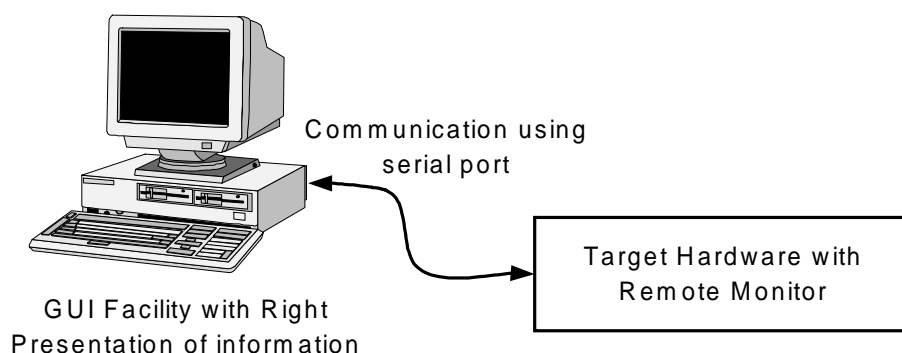
| | |
|---|----|
| 14.1 Introduction | 55 |
| 14.2 Program Execution Using StepOver Command | 55 |

1.1 - Introduction to Topview Debugger

Topview Debugger is the important facility meant for developing 8031 microcontroller based embedded solutions. Debugging is an inevitable part in any tool suite required to develop applications in real time. A right debugging tool may save a lot of development time in any product development process.

An exclusive version of Topview Debugger is made available for the Topview Trainer that gives the required development power to enable you to face real time challenges with ease and confidence.

This debugger is a two part program in which the major part stays inside of the trainer and keeps track of internal operation of microcontroller. During program execution, it catches information on various register contents, internal /external memory areas and also various peripherals of the microcontroller. This information is later transferred to the host computer to which it is connected. Since it is residing in the target hardware, sometimes it is also called, 'Remote Monitor'.



Second part of the debugger operates in the host computer and is responsible for presenting the information received from the remote monitor in a most useful format using a GUI environment.

When you establish a reliable communication link between the trainer and the host personal computer, the Topview Debugger automatically comes into action.

1.2 - System Requirements

The Topview Debugger operates in Windows 95/98 environments. The required hard disk space is about 3MB.

Host Computer with colour monitor and 800 X 600 pixel setting is preferred because contents of many windows are displayed in different colours for your visual convenience.

2.1 - Introduction

This part guides you to connect the Topview Trainer with the host computer for the debugging purpose. The trainer comes with a Power supply, Serial Port Cable, and a CDROM containing the Topview Debugger and the Assembler package.

2.2 - Connecting the Topview Trainer with the Host Computer

First, connect the power supply to the trainer through the connector CON2 marked as POWER.

Then connect the given serial port cable between CON3 marked as SERIAL PORT and the host computer's serial port.

Now install the Topview Debugger in the computer. You can do this very easily by executing the file, **Setup.exe** in the Topview Debugger folder in the CDROM. The installation program will guide you in remaining portion of the installation.

Switch on the power to the trainer. The display in the trainer will indicate a message " - - 8031 ". Execute the Topview Debugger in the host computer. Now host should establish the required communication link with the trainer.

If the selected serial port is not available or used by any other application, this will be indicated as shown here:



Connecting the Topview Trainer with the Host Computer

The default serial port is COM2.

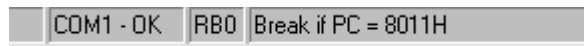
Select the serial port that is connected with the trainer and click **Retry** button to continue or **Quit** to quit the debugger.

If there is any error in the communication even after selecting the correct serial port, then the error will be informed as shown here:

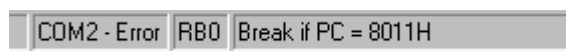


Press the **RST** key on the trainer and try again to establish the communication with the host computer.

The selected COM port is indicated on the status bar as shown:

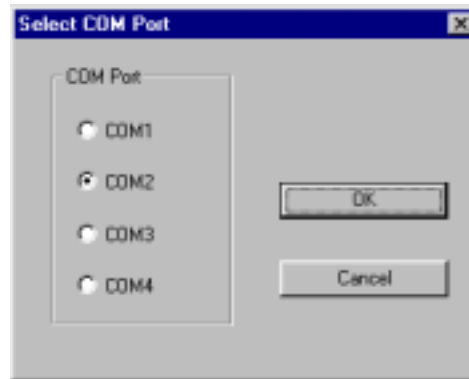


If there is any error, the status bar also indicates this error condition:



Connecting the Topview Trainer with the Host Computer

To select a different serial port, click **Serial Port** from the **Setting** menu. You may get a dialog box to indicate your selection:



Select the correct COM port and click over **OK** button or press **Enter** key.

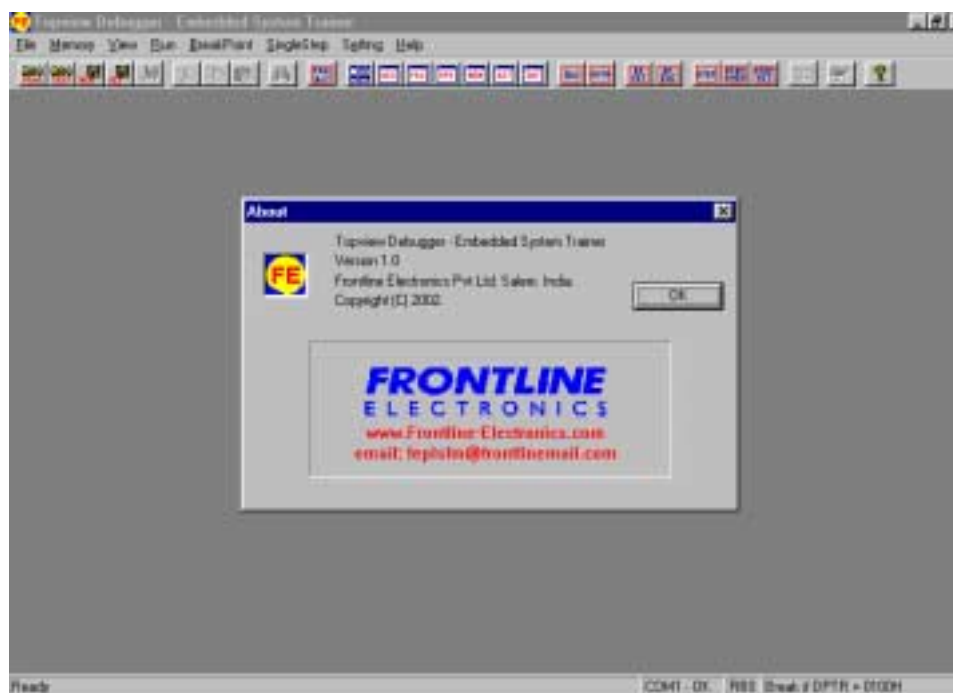
3.1 - Introduction

Topview Debugger sports an extensive and user friendly GUI environment using many windows to present the information in a more productive way. There are short cut ways to combine many operations in single step to enable the users implement repeated tasks in less time.

The details are given in subsequent chapters and now this chapter introduces this GUI environment to you.

3.2 - Debugger Toolbar

Topview Debugger uses many user friendly windows to implement an effective GUI interface. When you activate the debugger, an opening screen comes alive in your computer that presents an **About** window to display version details and Frontline Electronics' contact information



Click over **OK** button or press **Enter** key to close this window.

Now you can see a blank screen with a Tool Bar and a Menu Bar. Also at the

Debugger Toolbar

bottom, you can see a status bar that gives useful information when the debugger is in action.

For your convenience, the tool bar is given below and a short description on each command buttons. The debugger gives you same explanation about each of these buttons whenever you keep mouse cursor over that button.

3.3 - Tool Bar Portion



This command loads the program from the disk into the memory of the trainer. The file may be either in Hex or Binary format. It is same as activating **Load Program** from the **File** menu.



This button loads a text file into the Built-in Text Editor. This is same as **Load Text File** of **File** menu.



This button saves the memory contents of the trainer in the disk. It is similar to **Save Memory** command of **File** menu.



Makes the computer remember the current windows position and size for future reference. This is equivalent to **Save Setting** of **File** menu.



Enables you to key in your program into the memory of the trainer using built-in single line assembler. It is same as activating **Enter Program** command of **Memory** menu.



By clicking this button, you can make the ClearView window structure comes alive from the Normal view. It is same as activating **ClearView** command from **View** menu.



This button switches the debugger windows to Normal windows mode from the ClearView windows arrangement. It is equivalent to activating **NormalView** from the **View** menu.

Tool Bar Portion



This button opens the Register window or sets the Register window as active if it is already opened in the background. It is same as activating **Register Window** in **View** menu.



This opens the Program window or sets the Program window as active, if it is hidden in the background. It is equivalent to activating **Program Window** from the **View** menu.



This button opens the SFR window or makes the window active if it is in the background. It is same as clicking over **SFR Bit Status Window** from **View** menu.



This button is responsible for making Memory Bit Status Window active. If the window is not available in the background, a fresh window will be opened. It is same as activating the **Memory Bit Status Window** from **View** menu.



This button opens the External Memory window. If the window is hidden at the background, it is brought to front again. It is equivalent to activating **External Memory Window** from **View** menu.



This button is responsible to make the Internal Data Memory as active. You can use **Internal Data Memory** command from **View** menu for doing this.



This button executes your program in full speed taking PC contents or user defined address (already set) as the starting address. You can activate this command by activating **Go** from **Run** menu.



Executes your program in full speed after getting the starting address using a dialog box. It is same as activating **Goto** from **Run** menu.

Tool Bar Portion



It enables you to define the BreakPoint. It is equivalent to **Setting** from **BreakPoint** menu.



Executes your program in BreakPoint mode taking the PC contents as starting address. Equivalent to **Execution** command of **BreakPoint** menu.



Executes your program in SingleStep mode taking PC contents as starting address. It is same as activating **SingleStep** command from **SingleStep** menu.



Executes the program in StepOver mode taking PC contents as starting address. It is also available by clicking **StepOver** command in **SingleStep** menu.



Activates the SingleStep setting command. It is same as **Setting** command of **SingleStep** menu.



This is About command. Same as Help menu's **About** command.



Cuts the selected text in the Editor.



Copies the selected text into the clipboard. Same as activating **Copy** command in **Edit** menu.



Pastes the text kept in the clipboard into the editor. Equivalent to **Paste** in **Edit** menu.



Find command meant for the text editor.

Tool Bar Portion



Runs the external Assembler. It is equivalent to activating **Run Assembler** from the **Command** menu.

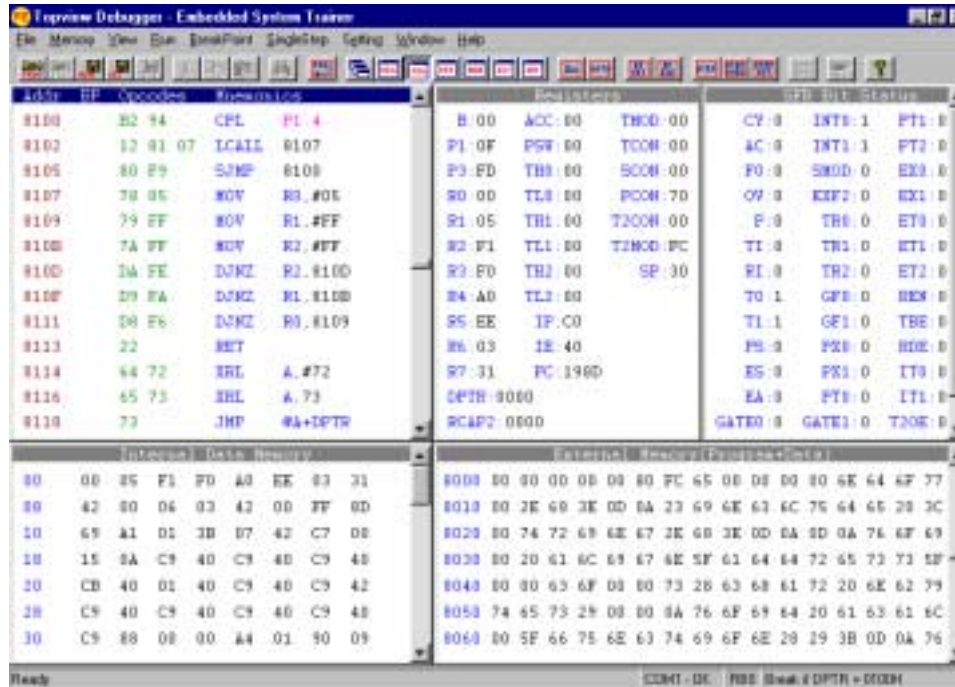


Closes all the opened windows in a single command. Same as **Close All** of **Window** menu.

3.4 - ClearView Window Structure

This is an optimized structure where windows are strategically placed in the display. Total display area of the monitor is divided into 5 windows. Windows meant for Program, Register, Internal Data Memory, External Data Memory and SFR bit status are placed in the ClearView.

Size and position of these windows cannot be changed. But scrolling facility is available wherever it is required.



ClearView Window Structure

This windows structure can be activated or all these windows can also be viewed in Normal view by clicking suitable tool buttons.

This ClearView gives you complete picture on the internal architecture in a single screen when you debug your target program code.

3.5 - Normal Window

In this configuration, you can activate as many windows as required during debugging process. All of these windows are tiled in many ways.

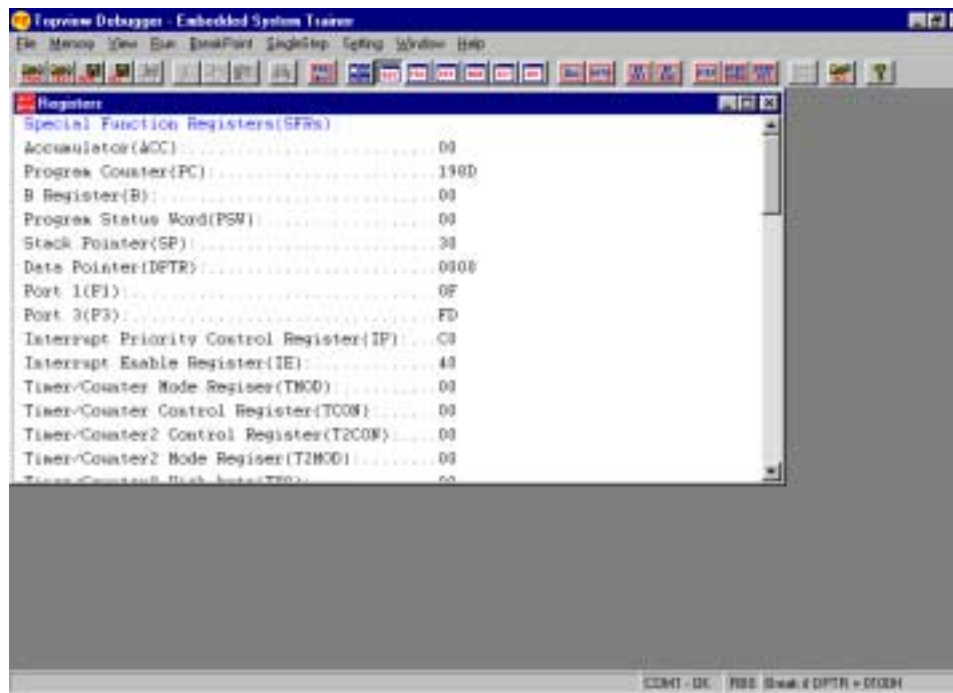
You can activate ClearView window structure at any time at the press of a button.

3.6 - Register Window

Click **Register Window** in the **View** menu or click the relevant button of the tool bar to open or activate this window.

This Register window indicates all the available internal registers and their contents. You can see the complete name of the registers and you can edit the contents by clicking over the register contents.

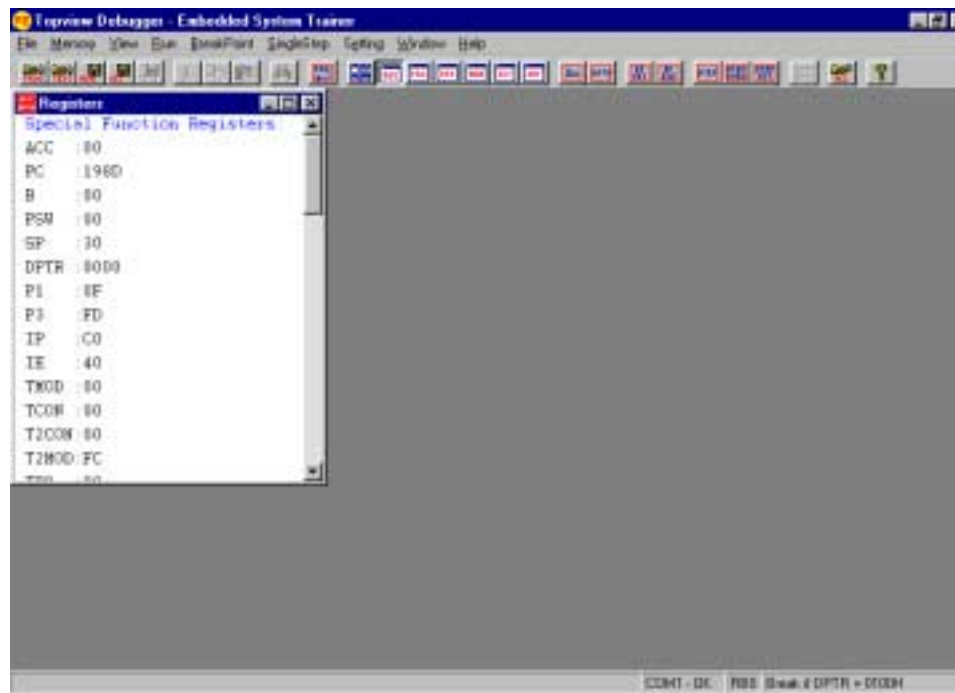
Register Window



The cursor starts blinking wherever you click. You can modify the register contents in that place. When you enter any data, the same will be automatically transferred to the trainer at that specific location.

If you reduce the size of this Register Window, the window will reconfigure itself as shown below.

Register Window

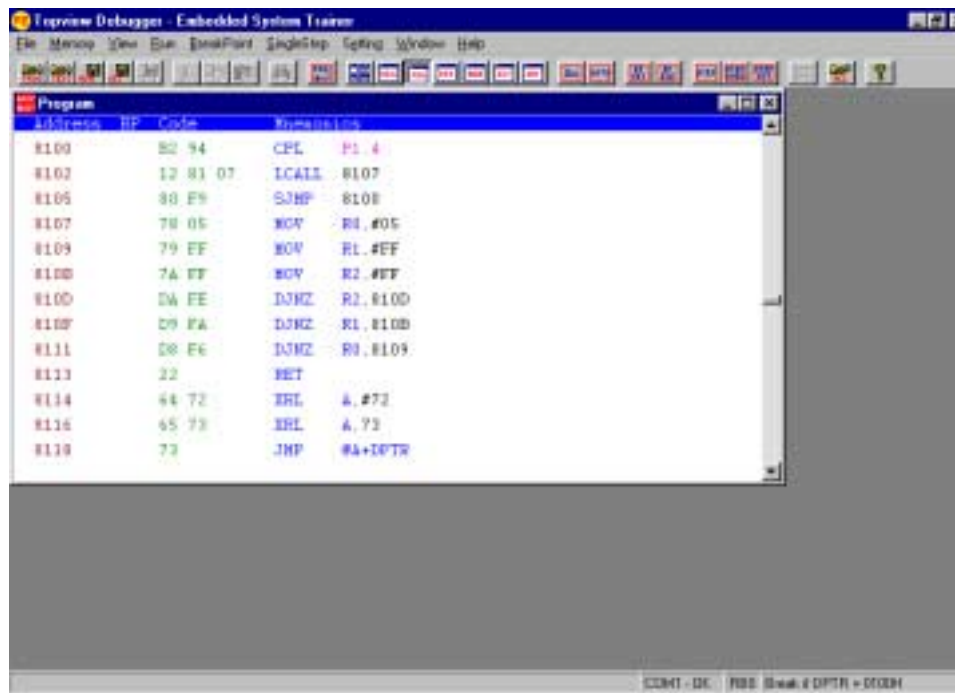


3.7 - Program Window

You can open this window by clicking **Program Window** in the **View** menu. If the window is in the background, it now becomes active and comes to the front.

This window disassembles the program memory contents as shown in the figure.

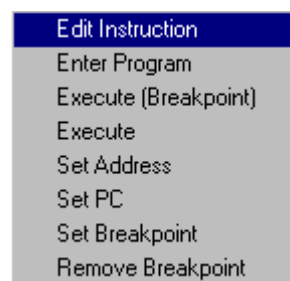
Program Window



Addresses, Opcodes and Mnemonics are displayed in different colours for your visual convenience.

Keeping this window active, you can enter your program line by line using built-in single line assembler. You can know more information on this in the chapter 7.

You can use mouse to do many useful tasks when you are using this program window. Just right click your mouse anywhere in this window and you will get a pop up window that gives a list of operations you can perform as shown:



Program Window

You can proceed with any of these commands by left clicking over the name of the commands.

- Edit Instruction.
- Enter Program.
- Execute (BreakPoint).
- Execute.
- Set Address.
- Set PC.
- Set BreakPoint.
- Remove BreakPoint.

3.7.1 - Edit Instruction

Using this command, you can edit the disassembled program that you are viewing at that time of mouse activation in the program window.

As you know already, this program window displays the raw program in the assembly language. Most of the time, you may use this window to read the target program from the memory of the trainer.

You can use this command to edit the existing code. You can modify/edit any program instruction anywhere in the window.

But if you need to develop or key in your target program right from the scratch you have to use the single line assembler or the external assembler. You can know more of this in the coming chapters.

So this 'Edit Instruction' is suitable for making small changes or editing only few instruction lines in the program window.

You can change a 3 byte instruction into a 2 byte or 1 byte instructions. You can't change a 1 byte instructions to 2/3 byte instructions for obvious reasons.

Edit Instruction

When you change a 3 byte instruction into 1/2 byte instructions the remaining locations are filled with NOP (00H) instruction.

You can also activate this **Edit Instruction** command by double clicking your left mouse button over the exact instruction in the window.

3.7.2 - Enter Program

Activating this command enables you to enter your target program using built in single line assembler.

Detailed information is available in the chapter 7.

3.7.3 - Execute BreakPoint

This command takes you to the BreakPoint execution. Details are available in the chapter 12.

3.7.4 - Execute

This command is for executing your target program in the trainer and you can get complete picture in the chapter 11.

3.7.5 - Set Address

You can set the starting address of the program list in the program window. You can also activate this command by double clicking left button of your mouse over the address portion displayed in the window.

A dialog box comes up to get the starting address.

3.7.6 - Set PC

By this command, you can set the PC with the address where you click the mouse.

The initialization of the PC is indicated by the yellow highlight at that instruction.

3.7.7 - Set BreakPoint

You can set any instruction as the PC BreakPoint by this command.

You can get more information about this BreakPoint operations in the chapter 17.

3.7.8 - Remove BreakPoint

If you activate this command at any specific instruction, which is already set for PC BreakPoint, then that PC BreakPoint condition is removed from that instruction.

3.8 - Internal Data Memory Window

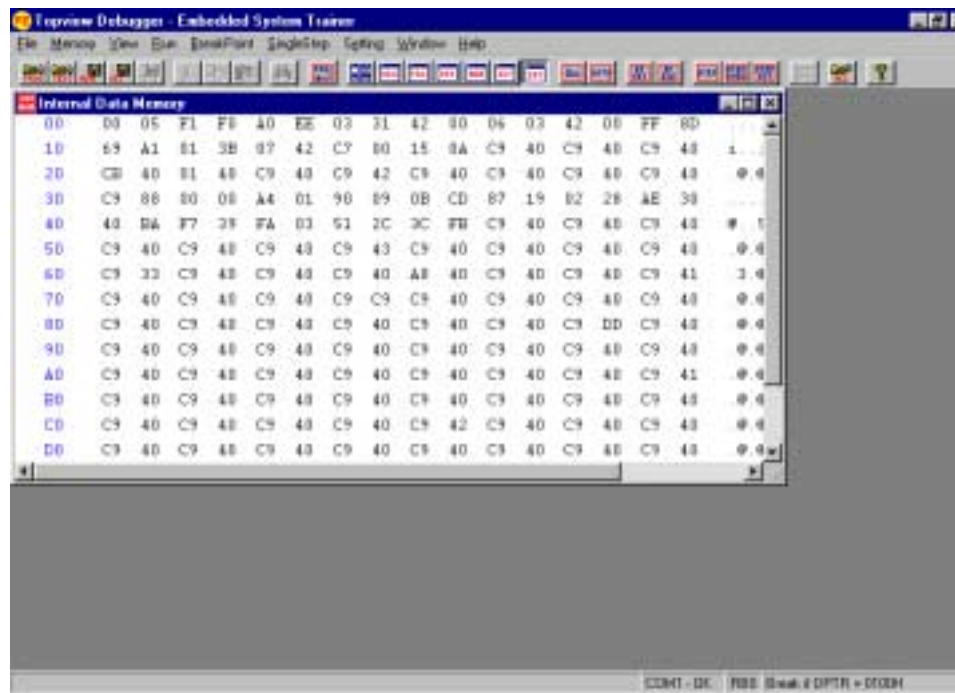
Activate this window by clicking **Internal Data Memory Window** from the **View** menu.

This window displays the memory addresses and the contents. The data are displayed in both Hex and ASCII formats.

When you are using the ClearView, you can see the contents of this window in either one of these formats. To change the contents from one format to another, right click your mouse anywhere in the window and select suitable option from the pop up window.

You can also edit the contents in this window.

Internal Data Memory Window



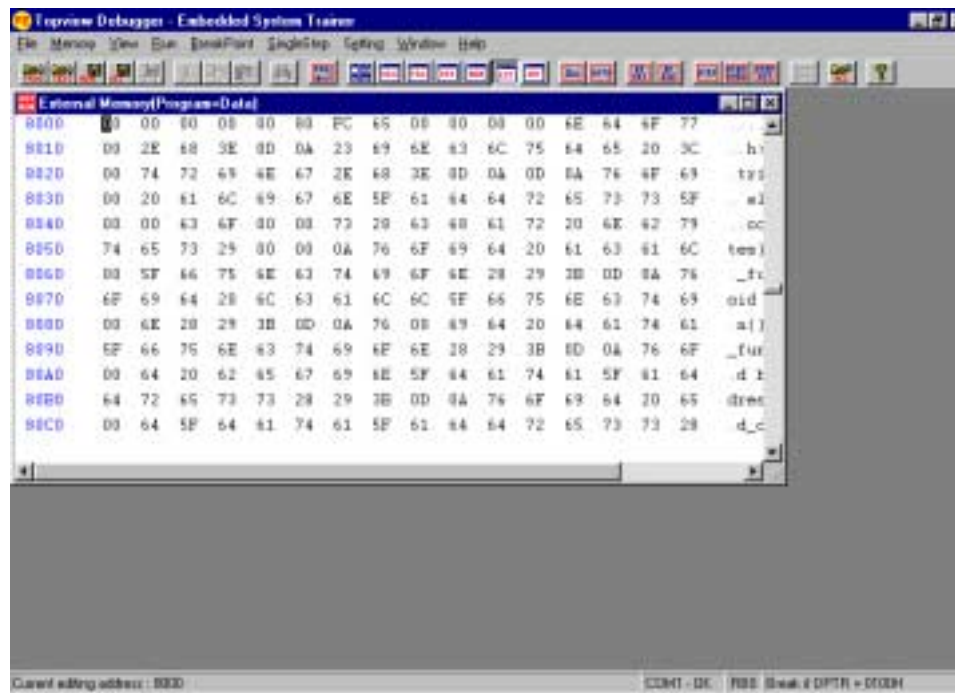
3.9 - External Memory Window

To activate this window, click **External Memory Window** from the **View** menu.

In this external memory window, the contents of the external memory are displayed along with the addresses. The data are displayed in both Hex and ASCII formats.

When you are using ClearView structure you can see the contents in any one of the format because of the space limitation. Just right click your mouse anywhere to select the required format.

External Memory Window



You can edit these memory contents by clicking over the required place and the entered data is automatically transferred to the exact location in the trainer.

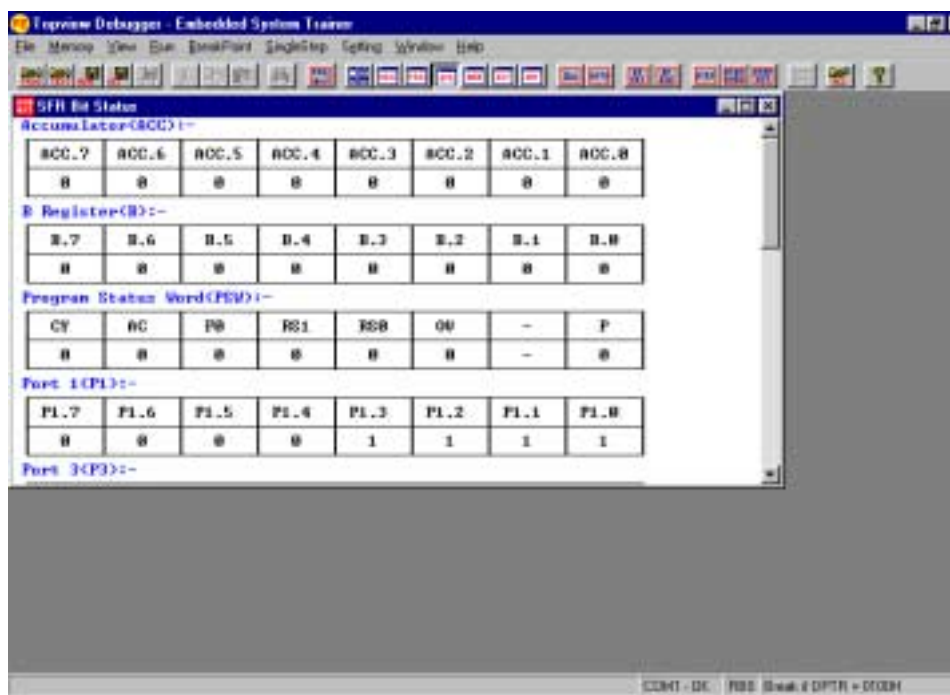
You can also enter ASCII data into this memory using this window. This is very useful handy feature when you use LCD modules.

When you are using this window configuration, you can define the starting address by double clicking left button of your mouse at address part of any instruction.

Remember that this External Memory Window indicates the overlapped Program and Data Memory portions of the trainer.

3.10 - SFR Bit Status Window

Activate this command, by clicking **SFR Bit Status Window** in **View** menu. You can see this window comes into action indicating SFR contents in bit wise along with their names.



You can also edit the contents in this window.

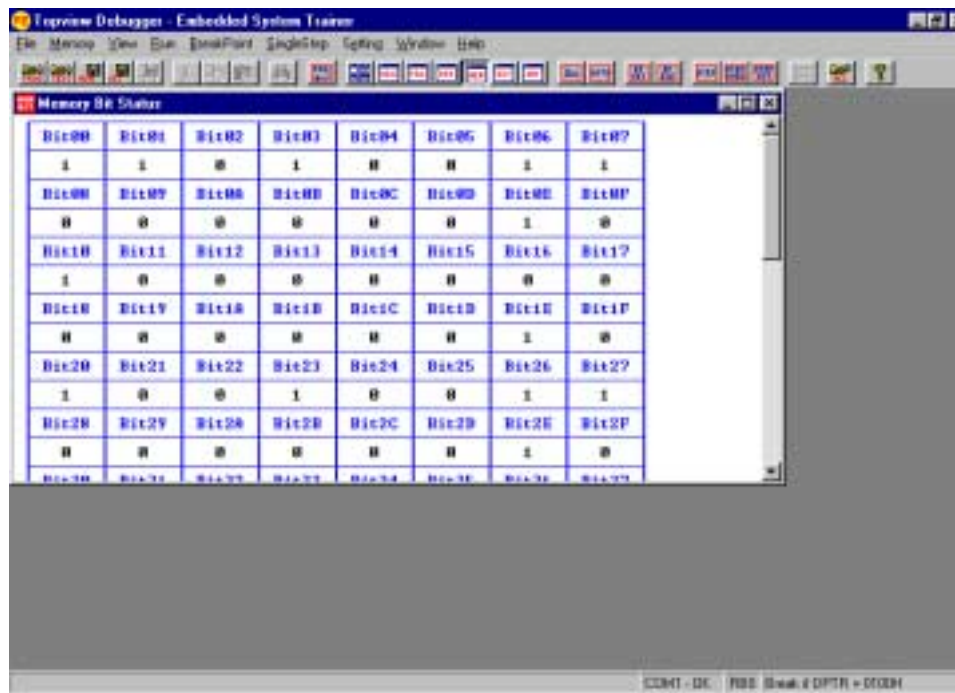
3.11 - Memory Bit Status Window

Click **Memory Bit Status Window** in **View** menu to get this window activated.

In this window, the bit addressable internal data memory (20H to 2FH) contents are displayed in bit wise along with their bit addresses.

You can also change the content of any bit when this window is in active condition.

Memory Bit Status Window



The debugger GUI also sports a status bar at the bottom of the display to indicate the current active Register Bank and the BreakPoint condition if enabled.

4.1 - Introduction

Topview Debugger gives you facility to develop your programs right from the scratch. Debugger's built-in Text Editor takes care of program entry operations. You can simply key in your target code line by line. You can also download any input program from the disk.

4.2 - Developing Target Program Code

You can call a third party Assembler package to assemble the keyed in programs. The debugger captures the output file coming out of the assembler and displays the same in a separate window for your convenience.

You can activate this only when using Normal View.

When you get assembly errors, you can keep both Text Editor Window and the second assembler output window side by side to analyze the output file. It is a convenient feature helping you in debugging process.

We have tested the debugger with the Freeware Cross Assembler supplied by the Atmel, ASM51. This assembler is made available in the accompanied CDROM.

Actually this ASM51 is a DOS program. But when you call this assembler through debugger you need not open DOS window separately. Assembling and Data capturing tasks are handled by the debugger itself. You really don't know that everything is happening in the DOS environment.

Then you can even download the assembled program straight into the trainer.

You can configure the whole process of developing the program and then downloading the same into the trainer into a Single step.

This is an important and time saving feature that helps you save time during repeated debugging operations.

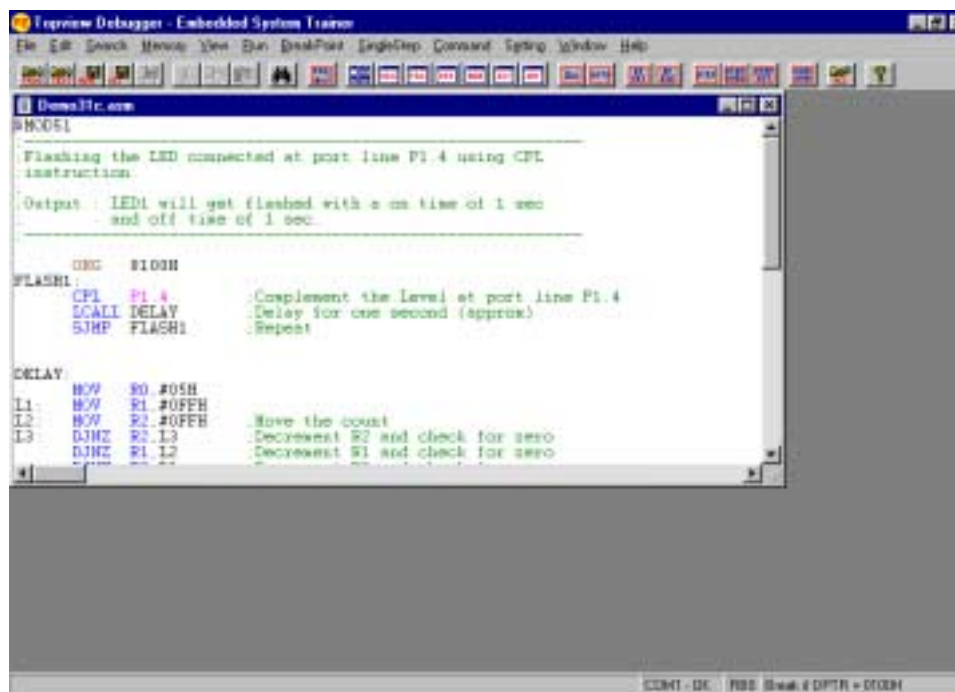
Developing Target Program Code

You should start the program entry by opening the Text Editor. Click **Load Text File** in the **File** menu for this.

A file open dialog box will appear on the screen to prompt you to select an existing file (available in the disk) or enter a new file name and then press **Open** button.



Now the Text Editor comes alive as shown here:



Developing Target Program Code

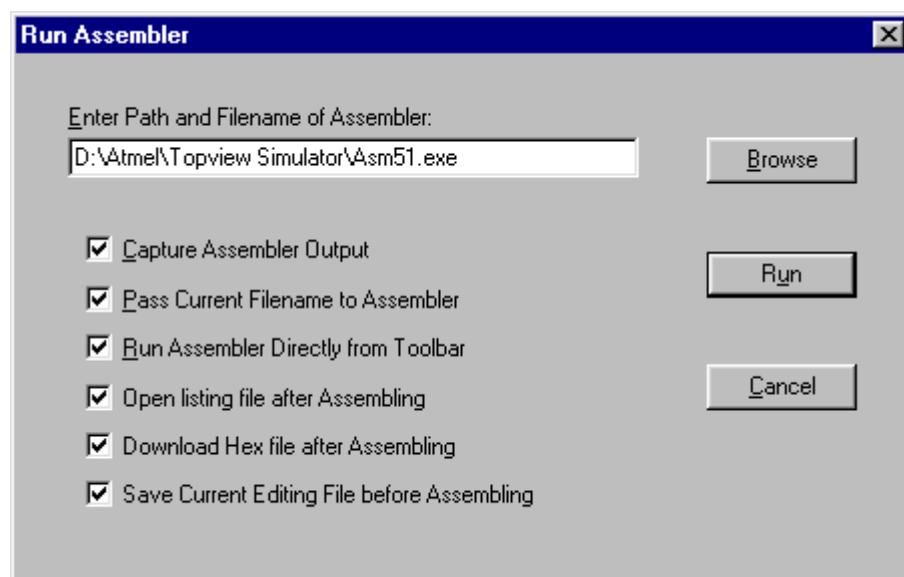
You can store the contents of the current editor into the disk using **Save Text File** command of **File** menu. From keyboard, use **Ctrl+S** to activate this command.

Suppose, if you want to store these contents in a new name, select **Save Text File As** command from **File** menu.

You can enter program code line by line. The editor supports colour syntax facility. Because of this, you can visualize different parts of the program, address, mnemonics, data and comments in different colours.

At this stage, you have to configure the activation of many tasks associated with the external assembler.

Now you select **Run Assembler** command from **Command** menu and a dialog box comes up inviting you to select different options.



If you want to capture the assembler output, check the **Capture Assembler Output** box.

Developing Target Program Code

To pass the current text editor file name to assembler as command line parameter, check the **Pass Current File Name to Assembler** box.

Check the **Run Assembler Directly from Toolbar** box to run this command from the tool bar without opening this dialog box every time.

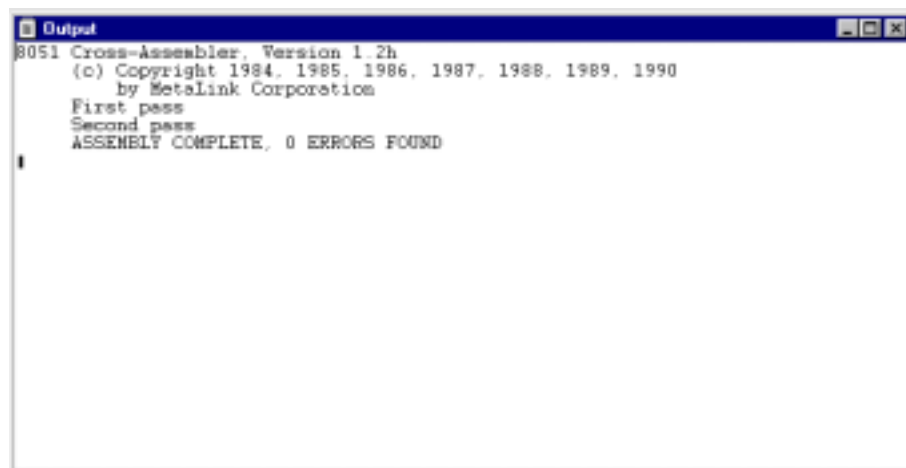
To open the listing file coming out of the assembler, check the **Open list File after Assembling** box.

To download the assembled program into the trainer, check the box, **Download Hex file after Assembling**.

To save the current editor file before passing it to the assembler, check the **Save Current Editor File before Assembling** box.

Apart from selecting all the above mentioned, you have to define the path of the external assembler using the **Browse** button.

Now click **Run** button to run the assembler. When this command is activated, if the capture assembler output is already selected, then the assembler output file is captured and displayed as shown below in a separate window.



```
Output
8051 Cross-Assembler, Version 1.2h
(c) Copyright 1984, 1985, 1986, 1987, 1988, 1989, 1990
by Metalink Corporation
First pass
Second pass
ASSEMBLY COMPLETE, 0 ERRORS FOUND
```

5.1 - Introduction

This chapter gives you an idea about loading a file from the disk into the memory of the trainer. As you know already, the external RAM indicates overlapped portion of both Program and Data Memories.

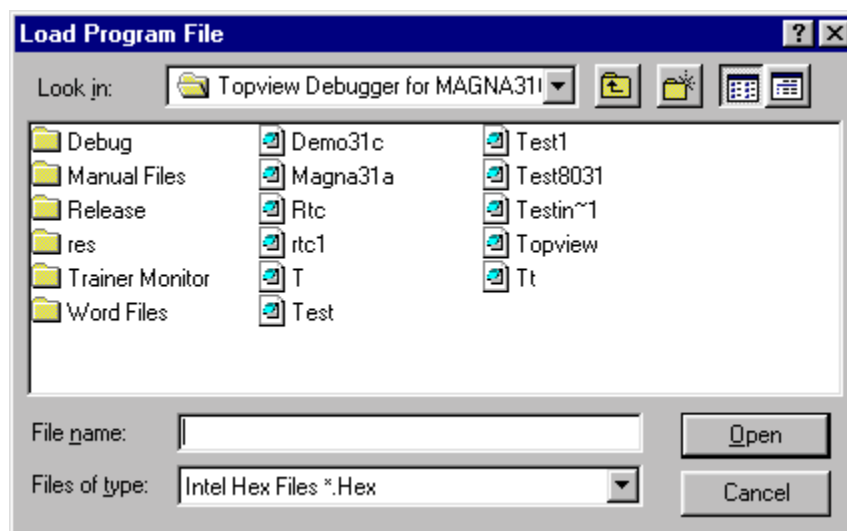
The address access range for this external RAM is from 8000H to FEFFH.

5.2 - Loading a Program from Disk to the Memory of the Trainer

Activate this command by clicking over the **Load Program** in the **File** menu.

Now you should see a dialog box coming up asking you to select the target file or enter a file name.

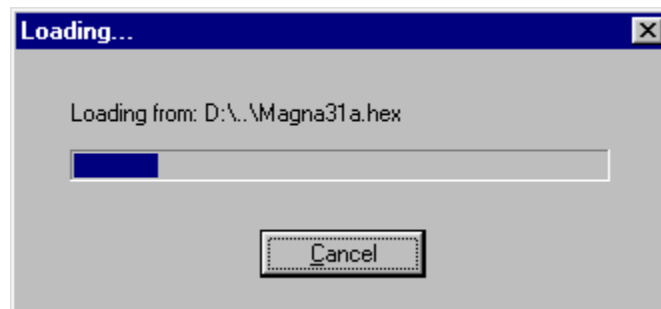
You can import both Hex and Binary files into the trainer.



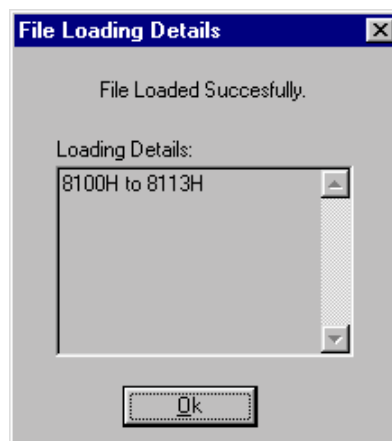
Select the file and click over **Open** button.

Another small window comes up indicating program loading using a progressive display.

Loading a Program from Disk to the Memory of the Trainer

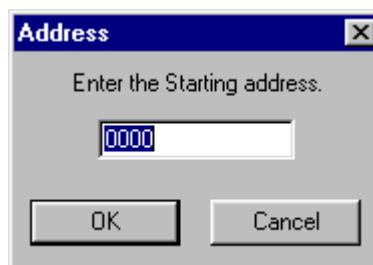


At the end of the command execution, a summary of addresses where the program is loaded is displayed as shown here.



Now click **OK** button to finish this Load Program Command.

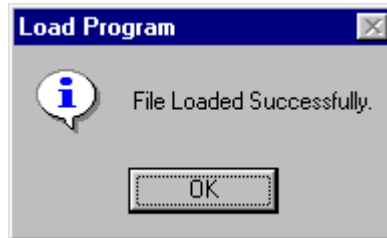
For the Binary file, a small dialog box may appear to get the starting address.



Key in the starting address and then click **OK** button or press **Enter** key to start loading.

Loading a Program from Disk to the Memory of the Trainer

The status of program loading is displayed in a separate message box.



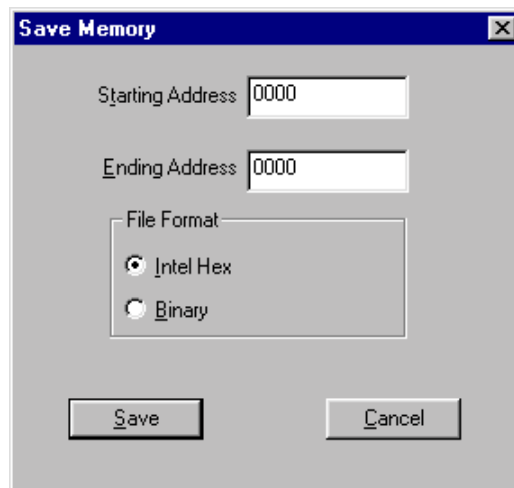
6.1 - Introduction

Here you will know about the way to store the memory contents of the trainer in the disk. You can save this memory block in both Hex and Binary formats.

6.2 - Storing the Memory Contents to Disk

Activate this save command by clicking **Save Memory** in **File** menu. You can see a dialog box comes up in the screen asking following information:

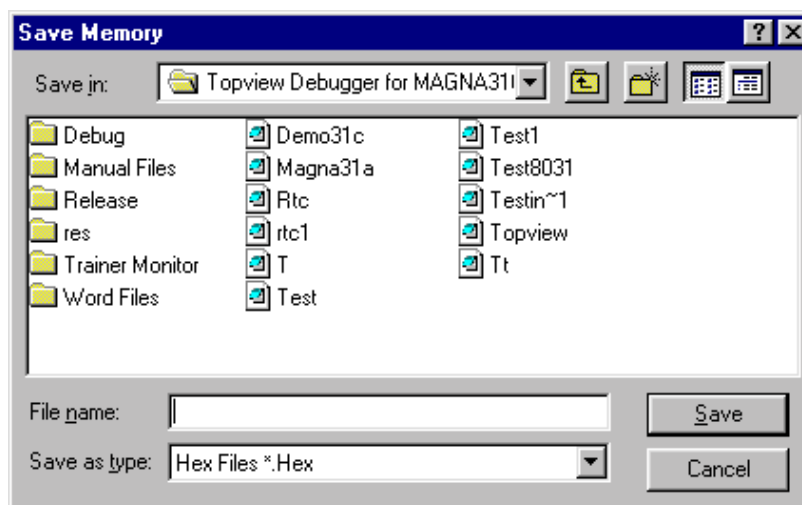
- Starting Address.
- Ending Address.
- File Format (Either IntelHex or Binary).



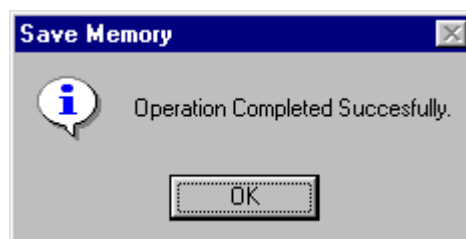
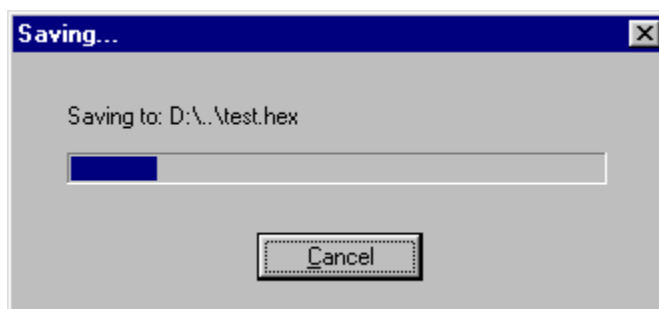
Key in and select suitable format and press **Save** button to proceed further.

Now another window pops up to get your file name. Either create a new file or select an existing file and then press **Save** button.

Storing the Memory Contents to Disk



Then a small window comes alive giving a progressive display of save operation and the completion of the operation is indicated by another window.



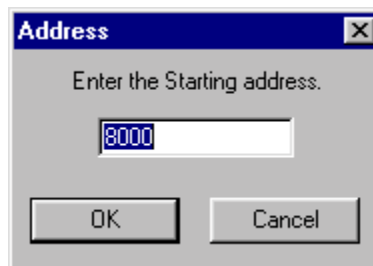
You can cancel the operation of this command at any time by clicking **Cancel** button in any dialog box.

7.1 - Introduction

You can start entering your target assembly language programs in two ways. Either you can call the external assembler once you entered the raw code in the Text Editor. In this case, your complete target code is assembled as a whole by the external assembler. Or you use single line assembler facility of the Program window. This chapter gives more information on how to use this single line assembler when keying in your application code.

7.2 - Using Single Line Assembler

Activate the command, by clicking **Enter Program** in the **Memory** menu. Then you should see a small dialog box coming up in the screen asking for the starting address.

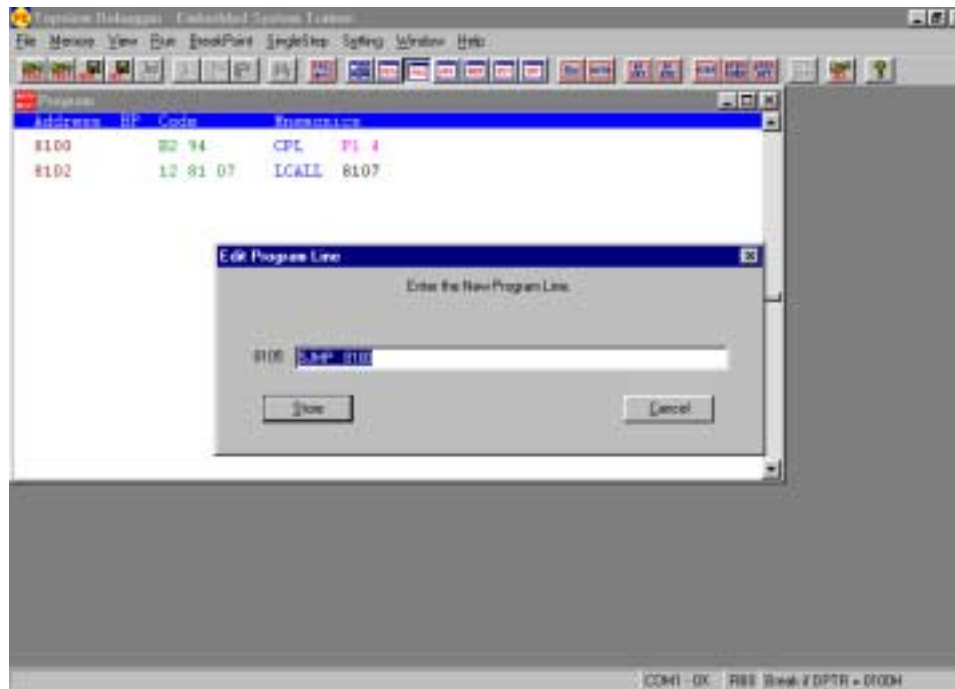


Enter the starting address of the program and press **Enter** key or click **OK** button.

Now the blank Program window gets opened.

Another dialog box will appear at the center of the screen for keying in program instructions as shown below.

Using Single Line Assembler



Start entering your target program line by line and keep on pressing **Enter** key or click over **Store** button to store program in the memory.

At the end of the program entry, press **Esc** key or click over **Cancel** to complete this operation.

Note: Kindly note that this single line assembler does not support any label assignment.

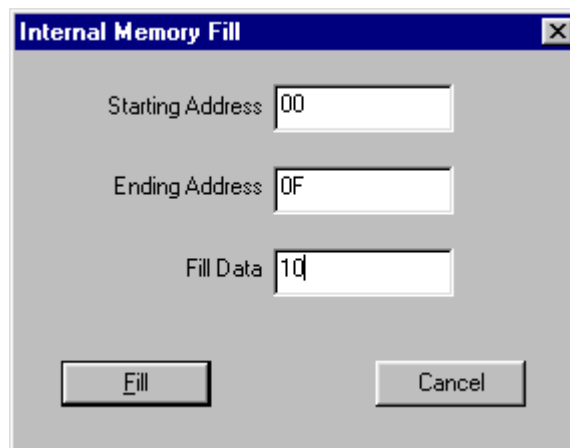
8.1 - Introduction

This chapter gives you more information on filling a fixed data in a block of memory spaces in both internal/external memory areas. Separate commands are available to fill both internal and external memories. Filling operation is same for both the memory areas.

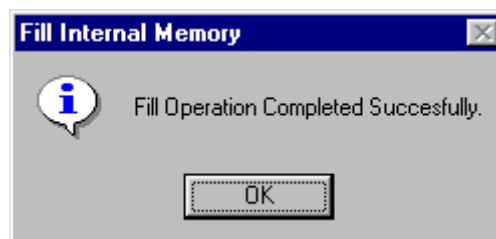
8.2 - Filling a Fixed Data in Internal/External Data Memory

Select either Internal Memory or External Memory command in the menu, **Memory** and then press **Fill** to activate this operation.

A dialog box may appear to get the starting address, ending address and the data meant for the Fill operation.



Key in the required information, press **Fill** button to start the operation. The completion of the operation is indicated by this message.



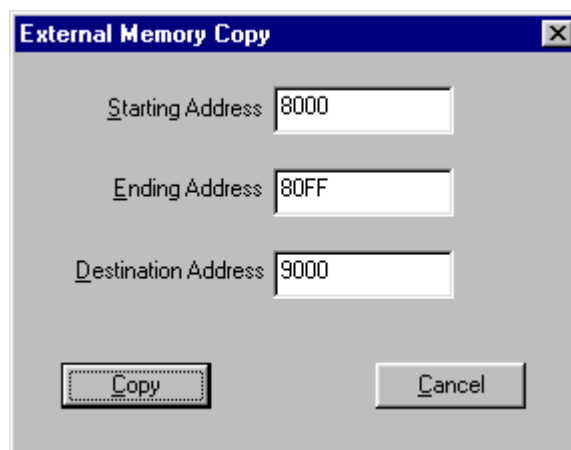
9.1 - Introduction

This chapter gives you an idea on how to copy a block of data from one location to other place in both internal and external memory areas. Separate commands are available to copy data in internal memory and external memory. The details are given here.

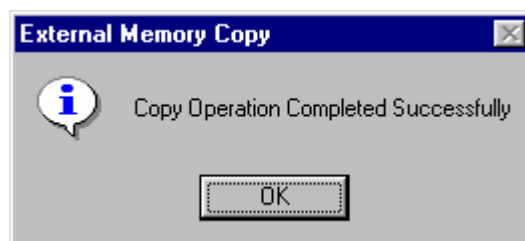
9.2 - Copying Block of Data From One Location to Other Place

You can activate this command by selecting **Memory → External Memory → Copy**.

Now you may see a dialog window pops up at the center of the screen prompting you for the address of the data blocks.



Enter the required information and press **Copy** command to start the operation. The completion of this copy operation is indicated by this window:



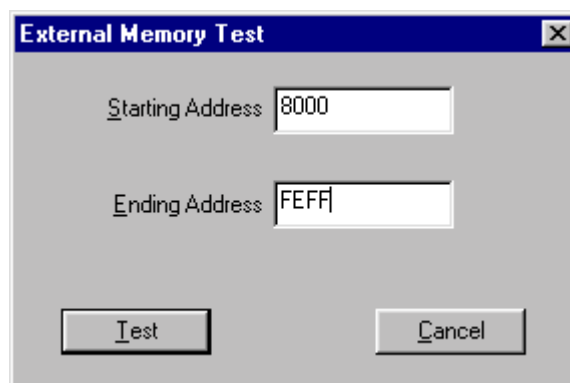
10.1 - Introduction

This command enables you to test RAM portion of the Trainer for any defects. The trainer has 32K byte size RAM meant for application development. Remember that this RAM is the overlapped portion of both Program memory and Data memory. If there is any defective location, the command indicates that for your convenience.

10.2 - Testing the Memory Area of the Trainer

Select this sequence: **Memory → External Memory → Test.**

Now you should enter the memory address information in the dialog window that comes up as shown



The completion of this operation is indicated by a message. If there is any defective location, another window displays that address as given below:



11.1 - Introduction

This command is the one of the most used in the debugging phase. Topview Debugger using right GUI features gives more information during execution of this command enabling you to get the complete picture. This chapter gives more details about this most important command.

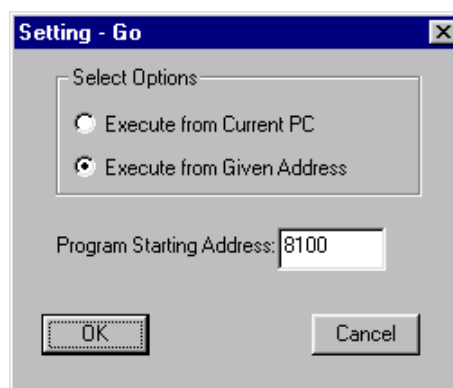
11.2 - Program Execution in Full Speed

Topview Debugger gives you total control on the program execution. You can execute your target program in many ways. You can execute the program in a single shot indicating the starting address of the program. You can make the debugger to execute the program upto a BreakPoint and then analyze the contents of various registers or the desired memory locations. May be you can execute the program line by line using the SingleStep command or execute subroutines in a SingleStep by the StepOver command.

Debugger gives you the required flexibility in controlling the application code execution and now we are going to see how to execute the program in single shot in full speed.

Run menu gives commands meant for full speed program execution: **Go** and **GoTo**.

Run menu also carries Setting option to define address selection required by the Go command. Before using Go command, click **Setting** and define your choice in the address of Program Counter.



Program Execution in Full Speed

If you enable the **Execute From Current PC** choice, the current PC value will be considered as the starting address for the program execution. Otherwise the program starting address will always be taken from the user defined PC value in the **Setting** option.

If the first option, **Execute From Current PC** has been enabled, then each time you have to set the PC value.

Program execution from the current PC option is helpful when using BreakPoints in your development. When the debugger stops the program flow at the BreakPoint value, you can continue program execution without redefining PC contents again. There are two ways to set the PC value.

When right clicking your mouse over the correct address in the Program window you get a pop up window giving few options. You select **Set PC** to initialize PC with that particular address. Now that instruction is highlighted with the yellow colour to indicate this step.

The second way is to enter the right address into PC in the Register window.

If the second option of the **Run** menu setting, **Execute From Given Address** is enabled, the starting address for the PC when using the **Go** command will be always taken from the user provided information of the Setting. This second method is very helpful in the occasions when you are debugging a program module (from the fixed address) in which case you need not define PC for every execution.

During repeated program debugging, just press **Go** button to start the program execution.

Program Execution in Full Speed

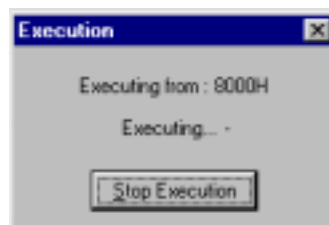
To get an idea on this **Go** command, enter the following program:

```
8000 CPL          P1.4
8002 ACALL        8050
8004 SJMP         8000
8050 MOV          R7,#05
8052 MOV          R6,#7F
8054 MOV          R5,#FF
8056 DJNZ         R5,8056
8058 DJNZ         R6,8054
805A DNJZ         R7,8052
805C RET
```

This program will flash the LED (LED2) connected at port line P1.4.

Now initialize the PC with the starting address of the program. Then click **Go** from **Run** menu. Or click over **Go** button in the toolbar.

A window comes up indicating the program execution operation and also the starting address.

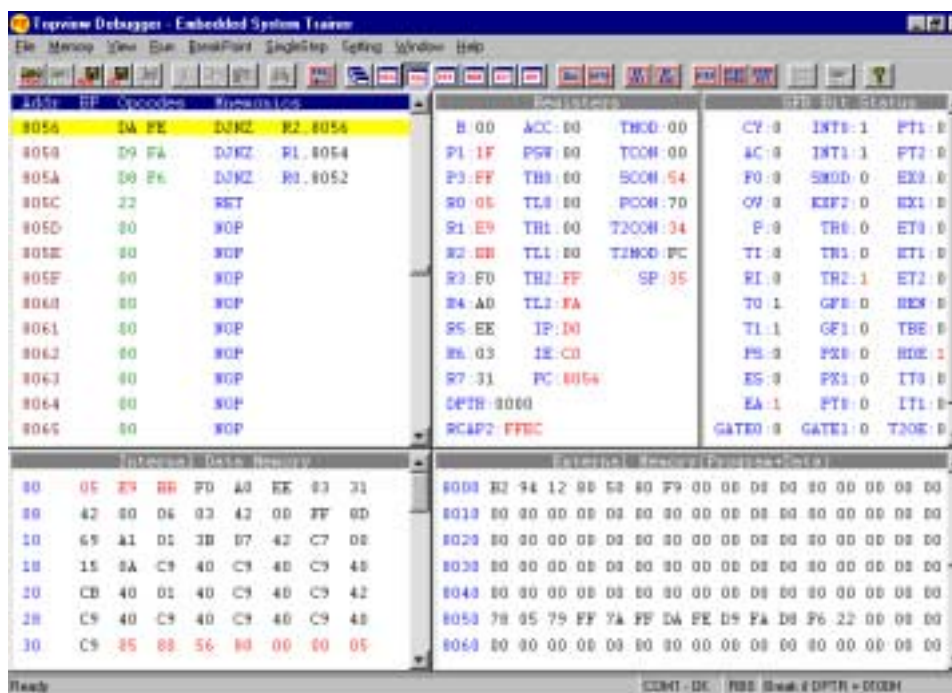


You can notice the LED connected with the port line starts flashing at an approximate interval of one second.

The program execution can be stopped by clicking over **Stop Execution** button or pressing **Esc** key.

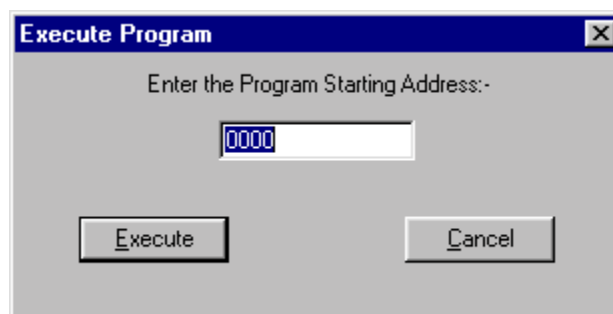
Program Execution in Full Speed

When you stop any execution, all the windows are refreshed and the PC at the moment of execution of stop command is highlighted as shown here:



11.3 - GoTo Command

This is almost similar to **Go** command. But you can use this as a short cut during program executions. Whenever you activate this command, it may ask for a starting address of your program.



GoTo Command

You can use this command as a short cut in following conditions:

Suppose you are debugging a lengthy program and you keep the ClearView configuration as active. You have to either scroll up or down to define the PC with the starting address.

Here just click **GoTo** command in the tool bar and enter your target address to execute the code.

Also assume you have different program modules at different places of your memory and PC has to be loaded with right address every time.

Here also you can initialize the PC with the correct address in just one step.

Other operations of this **GoTo** command is similar to the **Go** command.

Now you can appreciate various full speed execution possibilities when using Topview Debugger that save your valuable development time.

12.1 - Introduction

This type of program control is another way to keep track of your program's behavior during execution. You can make the debugger watch for a specific data in any memory location/register when the target program is executed. When the desired value is reached in the target location/register, debugger stops the program flow and refreshes all the windows with the current information to enable you to check for the desired results.

12.2 - Program Execution Using BreakPoints

You can set a BreakPoint in many places thanks to the Debugger's flexible setting capability.

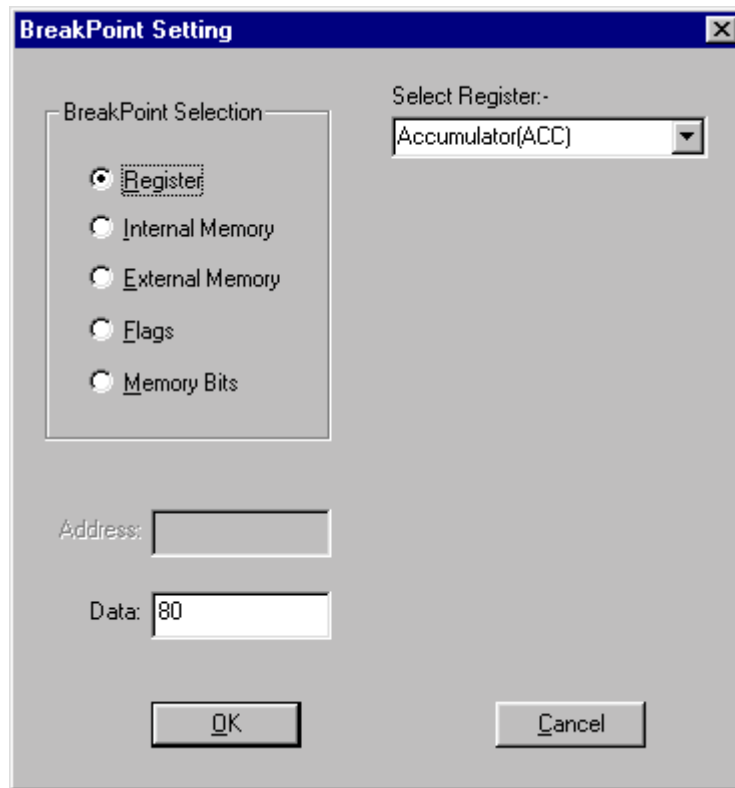
- Program Counter.
- Any one of these Registers:
ACC, B, DPH, DPL, DPTR, R0, R1, R2, R3, R4, R5, R6, R7 and SP.
- Internal Memory Location.
- External Memory Location.
- Any of these flags: Carry (C), Auxiliary Carry (AC), Flag0, Over flow Flag (OV) and the Parity flag (P).
- Any of the memory bit between addresses, 00H and 7FH.

As you can see, this range of options make your BreakPoint selection as very useful one during debugging.

To activate this command, click **Setting of BreakPoint** menu.

A dialog box pops up at the center of the screen prompting you to define the BreakPoint.

Program Execution Using BreakPoints



Make your selection and initialize that location/Register with the BreakPoint data and click over **OK** button or press **Enter** key. Now the BreakPoint is set for your next operation and the same is indicated in the status bar as shown:



Now the debugger starts waiting for the BreakPoint execution command. PC contents are considered as the program starting address when using the **BreakPoint Execution** command.

So before executing the program, initialize the PC with the correct starting address.

You can set the PC BreakPoints in an easy way. As you can see, the program window has a **B** indication that points the PC BreakPoint assignment at that specific address.

Program Execution Using BreakPoints

Just double click at the required address under **B** indication to define that particular address as PC BreakPoint. Debugger accepts this assignment by displaying the indication **B** at that address location

You can also set this PC BreakPoint by right clicking any instruction or it's address in the program window. Select the **Set BreakPoint** from the popup window.

You can remove any breakpoint that has been already defined by the same way. Just right click at that BreakPoint and select **Remove BreakPoint** from the popup window.

To give you more clear picture of this BreakPoint execution, an example program is given here.

```
8000  MOV  A,#00
8002  INC  A
8003  SJMP 8002
```

This small program increments the ACC from 00H to FFH continuously.

Set a BreakPoint to break the program flow when ACC gets the value 80H.

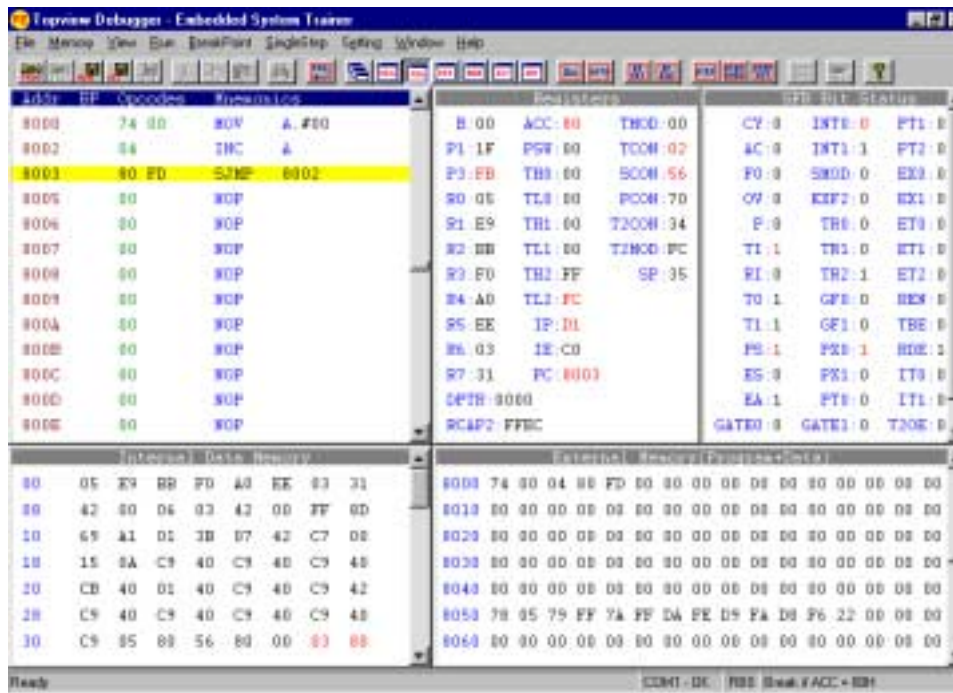
Activate BreakPoint Execution by clicking **Execution** in **BreakPoint** menu.

A window comes up as usual to indicate the **BreakPoint Execution** operation.



Program Execution Using BreakPoints

The program execution is stopped when the BreakPoint condition is met and all the windows are refreshed and you can see the changed Registers/memory locations with highlights.



For the example, you can notice the ACC content as BreakPoint value, 80H.

13.1 - Introduction

You can use this SingleStep command to analyze the target program line by line in complicated situations. It enables you to get the total picture of changes happened in Registers, internal/external memory for every program line execution. The GUI of Clearview promptly indicates these changes in highlights to draw your attention.

13.2 - Program Execution Using SingleStep Command

This facility makes you understand the behavior of the target code when you develop complex applications.

Activate this command by this sequence: **SingleStep** menu → **SingleStep**.

The current PC contents are taken as the starting address of the program. So keep the program address in PC before activating SingleStep command.

When you activate this command, first instruction gets executed and the debugger refreshes all windows with the current information and stops the execution and starts waiting for the next command.

For this command, the shortcut key is **F7**. So press this to execute the next instruction. Or click over **SingleStep** in **SingleStep** menu.

The following example helps you to understand fully this SingleStep command.

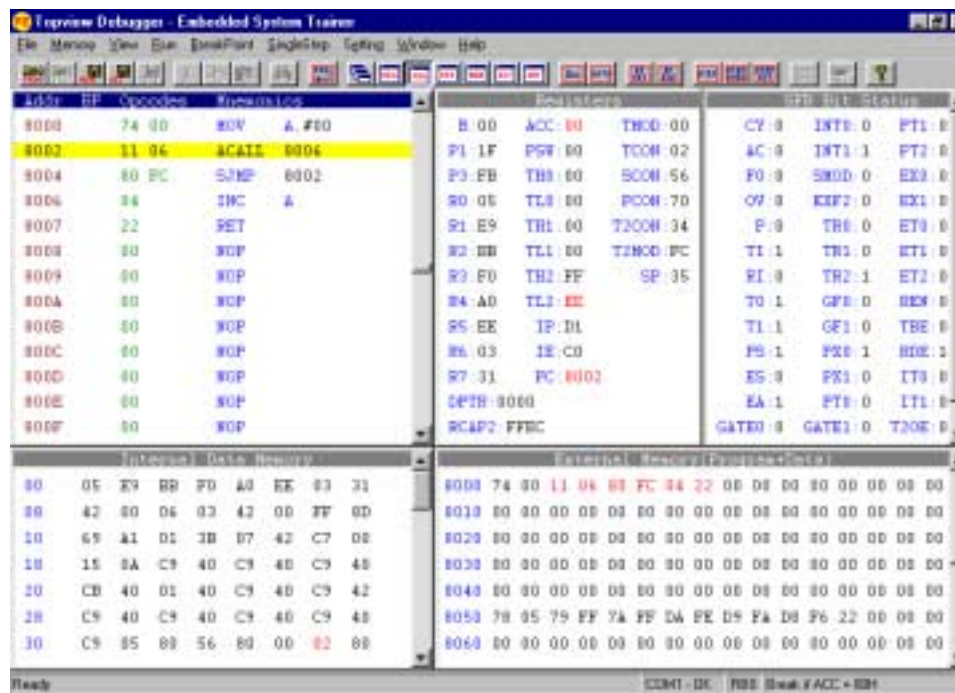
```
8000 MOV    A, #00
8002 ACALL  8006
8004 SJMP   8002
8006 INC    A
8007 RET
```

This simple program will increment the ACC contents from 00H to FFH. It calls one subroutine to increment ACC contents.

After entering the program in memory, initialize the PC with the address, 8000H.

Program Execution Using SingleStep Command

Activate the **SingleStep** command either from menu bar or toolbar. The first instruction, MOV A, #00 moves the data 00H to ACC and the program is made to stop. PC value becomes 8002 as shown in the fig.



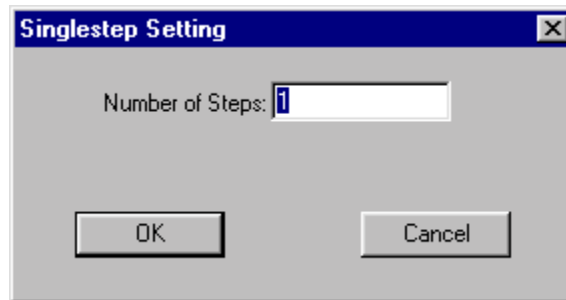
To execute the next instruction, you can activate **SingleStep** command again. You can notice that all the windows are refreshed with the current information in highlights using different colours to grab your attention.

13.3 - SingleStep Setting

The number of steps (ie. number of instructions to be executed) for the SingleStep and StepOver commands can be set using this command.

To activate the command, select **Setting** from **SingleStep** menu.

SingleStep Setting



When the command is activated, a dialog box will appear on screen prompting the number of steps.

Enter the number of steps and press **Enter** key or click **OK** button to set the new number of steps.

14.1 - Introduction

This is another useful command meant for debugging your complicated target code. This is similar to the SingleStep command in operation but it takes routine calls as a single instruction. You need not execute all the instructions of the routine line by line. The complete routine is executed in a single shot. Other instruction lines are executed line by line first like SingleStep execution.

14.2- Program Execution Using StepOver Command

To activate this command, select **StepOver** from the **SingleStep** menu. This command executes your program line by line and call routines by ACALL and LCALL are treated as single line. During the program flow, this command executes these routines in a single shot.

As usual all the windows are promptly refreshed after every program line/routine execution.

Key in and execute the following program using this command:

```
8000  MOV    A,#00
8002  ACALL  8006
8004  SJMP   8002
8006  INC    A
8007  RET
```

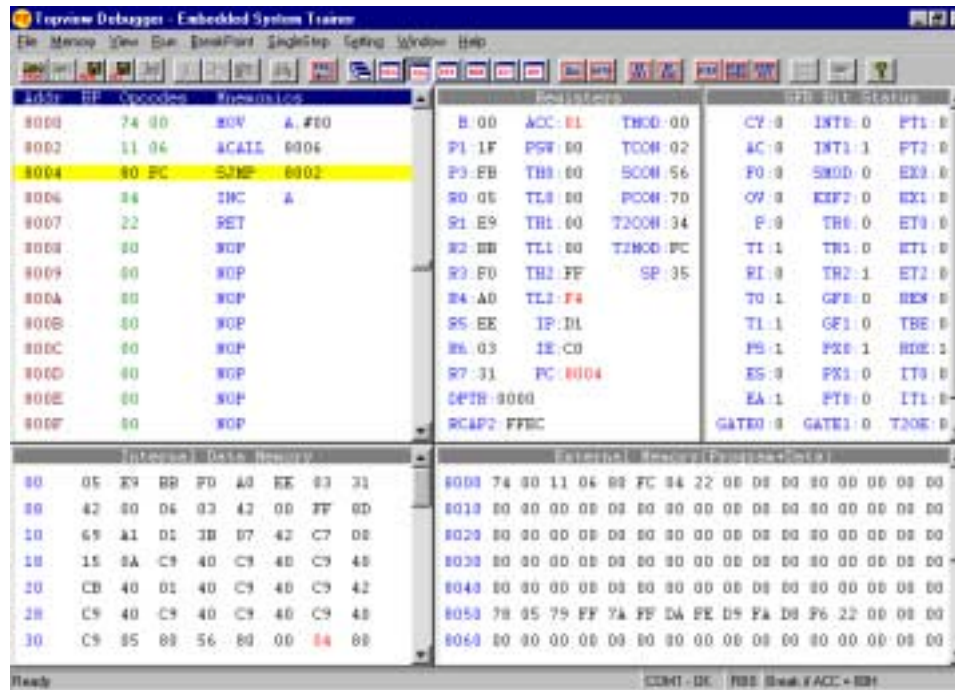
This program increments ACC from 00H to FFH. It calls a subroutine to increment ACC contents.

Initialize the PC with the address, 8000H.

Activate the StepOver command from either menu bar or toolbar. The first instruction moves data 00H to ACC and the program flow gets stopped. Now PC becomes 8002. Press **StepOver** button again to execute current

Program Execution Using StepOver Command

instruction. ACALL is executed in one step and PC points to the next instruction at 8004H.



Trigger **StepOver** command again. The ACC content is incremented by calling the routine at 8006H in one step. Now PC becomes 8004 instead of 8006.

Keep executing **StepOver** button till you finish debugging.

FRONTLINE ELECTRONICS PVT LTD

1/255C - Thatha Gounder St, Kumaran Nagar, Alagapuram,
Salem - 636 016, Tamilnadu. India.

Phone : 0091 427 - 244 9238 / 243 1312. Fax : 0091 427 - 244 9010.

Email : feplslm@frontlinemail.com

www.Frontline-Electronics.com