EM78 Series Microcontrollers

IDE

INTEGRATED
DEVELOPMENT
ENVIRONMENT

USER'S GUIDE

Doc. Version 1.0

(Applicable to eUIDE Version 1.0 & later)

ELAN MICROELECTRONICS CORP.

October 2009

Trademark Acknowledgments

IBM is a registered trademark and PS/2 is a trademark of IBM.
Windows is a trademark of Microsoft Corporation.

ELAN and ELAN logo are trademarks of ELAN Microelectronics Corporation.

The contents of this User's Guide (publication) are subject to change without further notice. ELAN Microelectronics assumes no responsibility concerning the accuracy, adequacy, or completeness of this publication. ELAN Microelectronics makes no commitment to update, or to keep current the information and material contained in this publication. Such information and material may change to conform to each confirmed order.

In no event shall ELAN Microelectronics be made responsible for any claims attributed to errors, omissions, or other inaccuracies in the information or material contained in this publication. ELAN Microelectronics shall not be liable for direct, indirect, special incidental, or consequential damages arising from the use of such information or material.

The software (eUIDE) described in this publication is furnished under a license or nondisclosure agreement, and may be used or copied only in accordance with the terms of such agreement.

ELAN Microelectronics products are not intended for use in life support appliances, devices, or systems. Use of ELAN Microelectronics product in such applications is not supported and is prohibited.
NO PART OF THIS PUBLICATION MAY BE REPRODUCED OR TRANSMITTED IN ANY FORM OR BY ANY MEANS WITHOUT THE EXPRESSED WRITTEN PERMISSION OF ELAN MICROELECTRONICS.

ELAN MICROELECTRONICS CORPORATION

Headquarters:

No. 12, Innovation 1st Road
Hsinchu Science Park
Hsinchu, TAIWAN 30076
Tel: +886 3 563-9977
Fax: +886 3 563-9966
webmaster@emc.com.tw
http://www.emc.com.tw

Hong Kong:

Elan (HK) Microelectronics Corporation, Ltd.
Flat A, 19F., World Tech Centre 95
How Ming Street, Kwun Tong
Kowloon, HONG KONG
Tel: +852 2723-3376
Fax: +852 2723-7780

USA:

Elan Information Technology Group (U.S.A.)
PO Box 601
Cupertino, CA 95015
U.S.A.
Tel: +1 408 366-8225
Fax: +1 408 366-8225

Shenzhen:

Elan Microelectronics Shenzhen, Ltd.
3F, SSMEC Bldg., Gaoxin S. Ave. I
Shenzhen Hi-tech Industrial Park
(South Area), Shenzhen
CHINA 518057
Tel: +86 755 2601-0565
Fax: +86 755 2601-0500
elan-sz@elanic.com.cn

Shanghai:

Elan Microelectronics Shanghai, Ltd.
#34, First Fl., 2nd Bldg.,
Lane 122, Chunxiao Rd.
Zhangjiang Hi-Tech Park
Shanghai, CHINA 201203
Tel: +86 21 5080-3866
Fax: +86 21 5080-4600
elan-sh@elanic.com.cn

Contents

Appendix

User's Manual Revision History		
Doc. Version	Revision Description	Date
0.1	User's Guide Initial Preliminary Version	2009/07/08
1.0	User's Guide Initial Official Version	2009/10/27

Chapter 1
Introduction

1.1 Overview

The EM78 Series Integrated Development Environment is a project oriented development tool for ELAN's EM78 Series microcontrollers. It comprises of the UICE development in-circuit emulator and the eUIDE software tool.

1.2 Introduction to eUIDE Program

The eUIDE is a Windows 2000 or Windows XP based program for UICE development in-circuit emulator that is used in the development of EM78 Series 8-bit microcontrollers of ELAN. The aims of the eUIDE are to provide a friendly operation environment, powerful functions, a high-speed transmission, and a stable system during development of the microcontrollers.

The eUIDE offers a lot of friendly functions including block comment, auto-updated registers, and real time line disassembler. It is made up of four main modules, namely; the Editor, Project Manager, Assembler, and Debugger, with each module having their respective submenus.

- **Editor:** provides editing functions for creating, viewing, and modifying the source files. It supports Find, Replace, Undo/Redo, and Cut/Copy/Paste.
- **Project Manager:** provides functions for inserting files into a projects deleting files from a projects and compiling of the project.
- **Assembler:** support such functions as Include, Macro, Assembly Arithmetic, Conditional Assembly, List Files, and Map File
- **Source Level Debugger:** provides source-level debugging function on the target which is embedded on the UICE. You can explore and analyze the status register, and the memory contents of the EM78 series target with the eUIDE. With its powerful features, like multiple breakpoints, real-time modification of register contents, and disassembly, the UICE becomes an indispensable partner of eUIDE in offering a perfect development environment for EM78 Series microcontrollers

1.2.1 eUIDE Main User Interface

The eUIDE is a project oriented integrated development environment (IDE) system that is used to edit user application programs and generates emulation/layout files for ELAN's EM78 series (8-bit) microcontrollers.

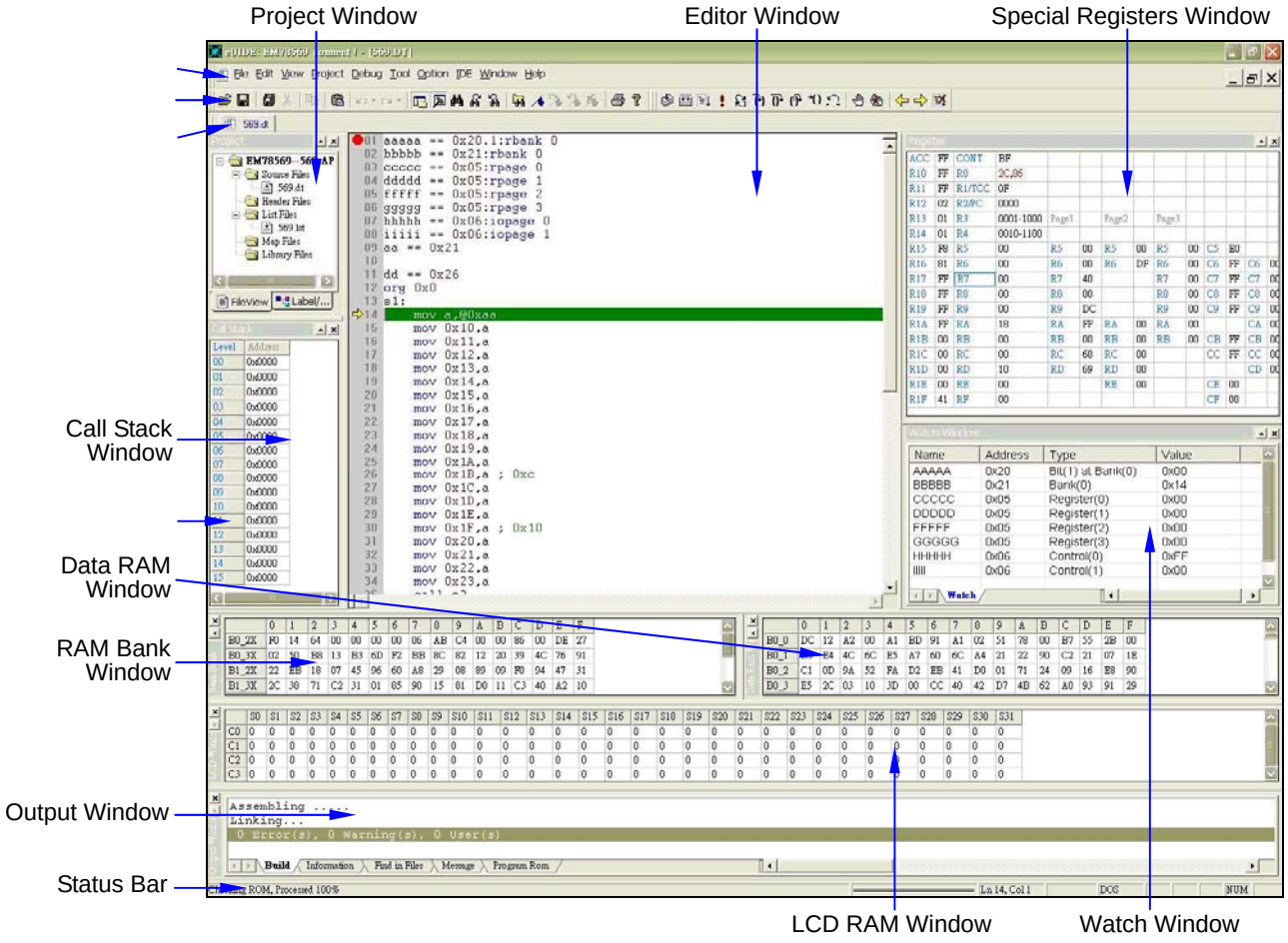


Figure 1-1 eUIDE Main Window Layout

NOTE

Actual number of sub-windows may vary in accordance with the actual target IC in use.

1.2.2 eUIDE Sub-Windows

The sub-window may be displayed or hidden by clicking on the pertinent window commands from the View menu (see Section 2.1.3)

1.2.2.1 Project Window

The Project Window consisted of two view modes, namely; **File View** mode and **Label/Function** View mode.

■ **File View Mode**

The Title Bar of the Project window shows your current microcontroller and project filename.

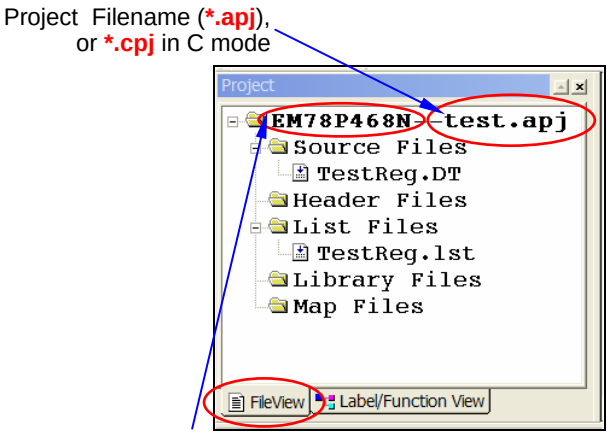


*Figure 1-2a **Project** Window in File View*

The **Project** window holds the *Source*, *Header*, *List*, and *Map* files.

Where**:**

Source Files (**.dt*) –
are the assembly source files that are added into the current project. In C mode, source file will be *.c file.

Header Files *(*.h)* –
are the reference files required by source program.

List Files *(*.lst)* – are the list files.

Map File (**.map*) – are the map file.

Library File (**.bbj*) – are the reference files required by source program.

After you have opened or created a project, click on the selected folder to expand and browse its contents. Then right-click on the selected file to display its shortcut pop-up menu.

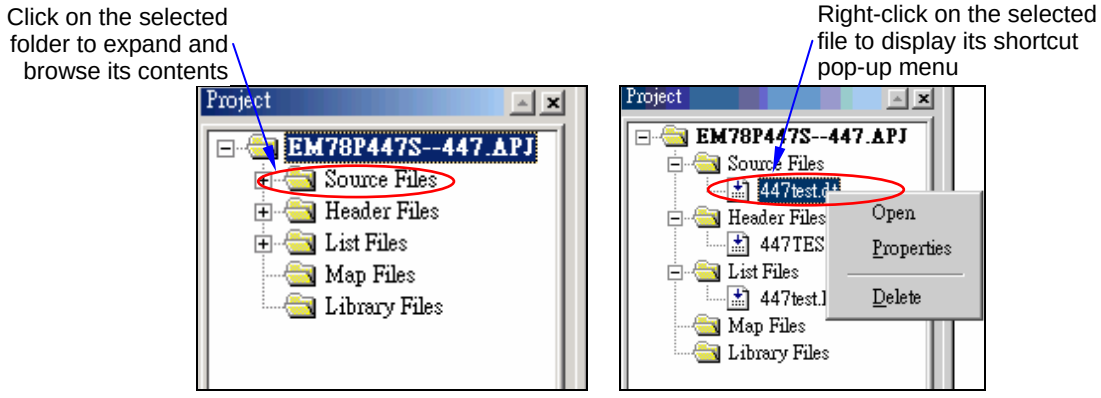


*Figure 1-2b Accessing the Supporting Files from **Project** Window*

The following explains the functions of the 3 commands in the pop-up menu.

Open – Opens the selected file. For example, right clicking on *447test.dt*, the file then opens. If the file is already opened, no action is performed.

Properties – Displays the complete path of the selected file or as illustrated in the following figure which shows where the *447test.dt* file is located.

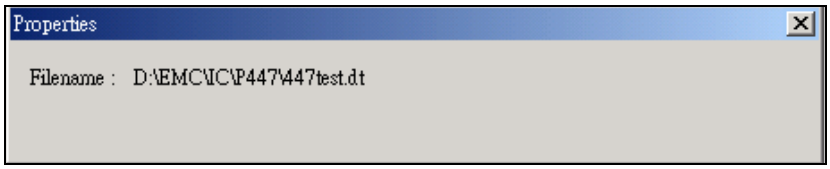


Figure 1-2c An Example of a Complete Path of the "447test.dt" File

Delete – Removes the selected file from the current Project Source Files folder. If the file is still opened, the program closes the file before removing it. For example, if you select to remove the *447test.dt* file, a confirmation dialog will display. Click the **Yes** to remove the file from the project. Otherwise click **No**.

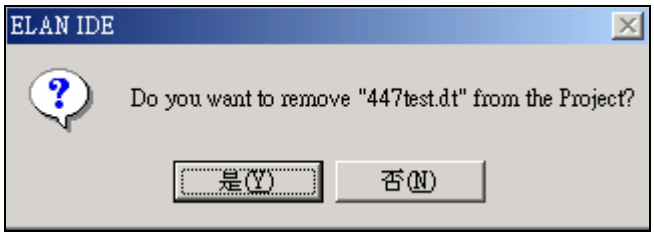


Figure 1-2d Removing a File from Project Source Files Dialog

■ Label/Function View Mode

To access the **Label/Function View** window, click the **Label/Function View** tab at the bottom of the **Project** window

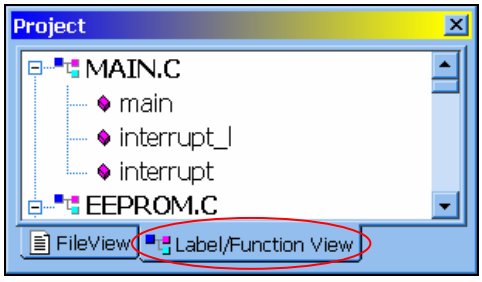


*Figure 1-2e **Project** Window in Label/Function View*

After performing a code dump, eUIDE will automatically determines the functions from "C" code (see figure at left) and labels from ASM code. The result is then displayed in the **Label/Function View** mode of the **Project** window categorized by files.

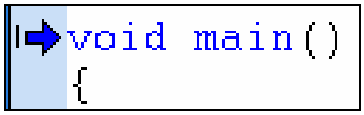


*Figure 1-2f Searched Label /Function Location Pinpointed in the in **Editor** Window*

To find the location of the displayed label or function in the file, double-click on a label or function. The eUIDE will automatically pinpoint to the pertinent location in the **Editor** window (as illustrated at left). At the same time, eUIDE will display the search results in the **Output** window as illustrated in *Figure 1-2*g below.

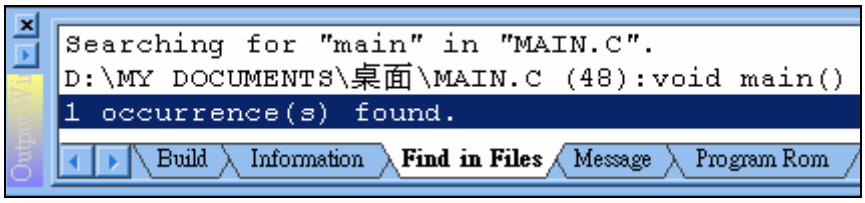


*Figure 1-2g Simultaneous Search Results Display in the **Output** Window*

If after editing the code eUIDE is unable to locate the label/function location in the **Editor** window, the **Output** window will display a message providing a reason for not finding the searched item.

1.2.2.2 Editor Window

*Figure 1-3a **Editor** Window*

The **Editor** window is a multi-windowed editing tool for creating, viewing, and debugging source files.

Its major editing features are**:**

- Unlimited file size
- Multiple files can be opened and displayed at the same time
- Insert (overstrike) mode for editing
- Undo/Redo
- Clipboard support (text can be cut, copied, moved, and pasted onto the clipboard using a keystroke)
- Drag and drop text manipulation (highlighted text can be dragged and dropped between any of the IDE windows)

■ Interacting with Editor Window

The figure below shows a typical **Editor** window displaying contents of an opened source file (*447test.dt*). Assembly keywords are shown in blue, comments & comments symbol are in green, values are in brown, and the rest are shown in black.

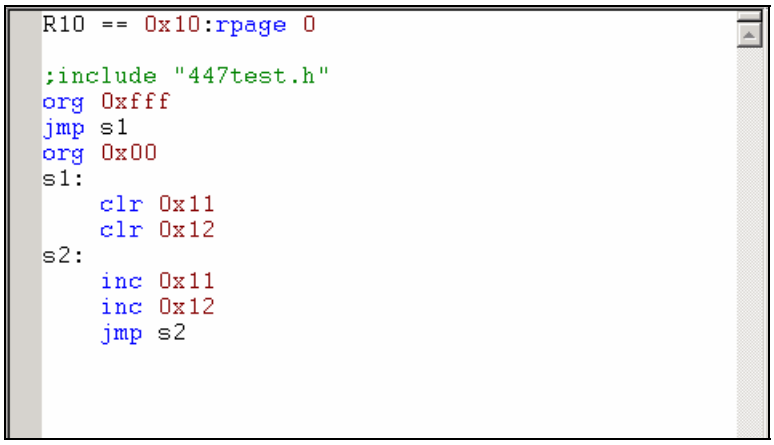


*Figure 1-3b **Editor** Window Displaying Contents of an Opened Source File "447test.Dt"*

Right-click anywhere within the **Editor** window to display an **Edit** shortcut menu (shown at right) exclusive for **Editor** window application.

*Figure 1-3c Edit Shortcut Menu for **Editor** Window*

Note that most of the shortcut menu commands are also available from the **Edit** menu of Menu Bar.

Each command in the shortcut menu functions as follows**:**

Cut – Removes the selected text from current location and move (paste) it into another location.

First, select the desired text range you want to move and then right-click within the selected text. With the shortcut menu on display, click **Cut** command from the menu. The selected text are then removed from the **Editor** window (as demonstrated in the following figures) and temporarily stored in the clipboard. Proceed to paste the text into your target location.

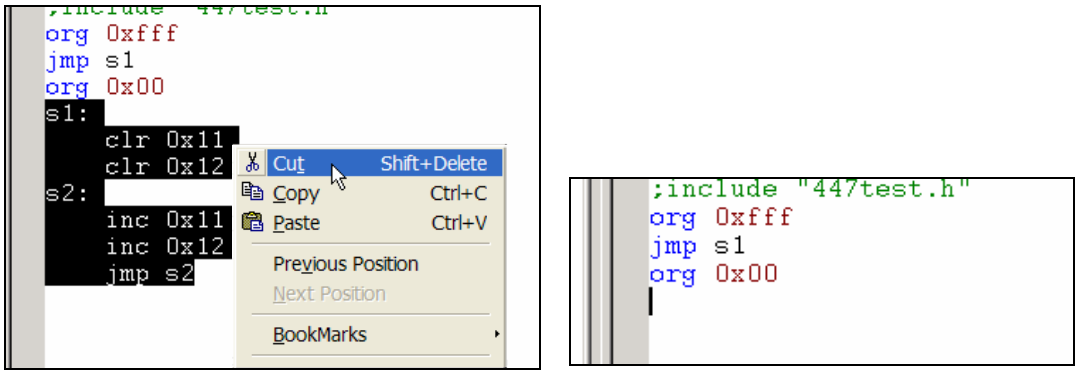


*Figure 1-3d An Example of a **Cut** Command Operation*

Copy – First, select the desired text range you want to copy and then right-click within the selected text. With the shortcut menu on display, click **Copy** command from the menu. A copy of the selected text is then temporarily stored in the clipboard. Proceed to paste the text into your target location.

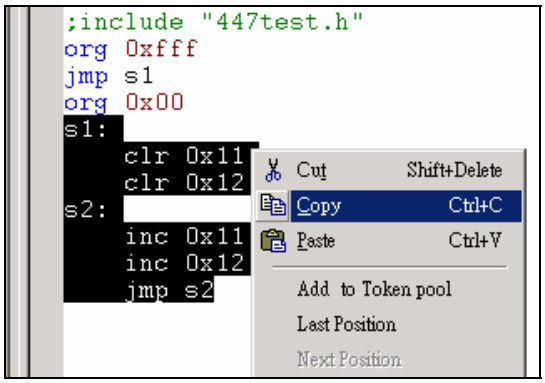


*Figure 1-3e An Example of a **Cut** Command Operation*

Paste – Insert the selected text (that has been recently cut or copied into clipboard) into the target location. The figure at right shows an example of a paste result (framed) at the bottom of the same page in the **Editor** window.

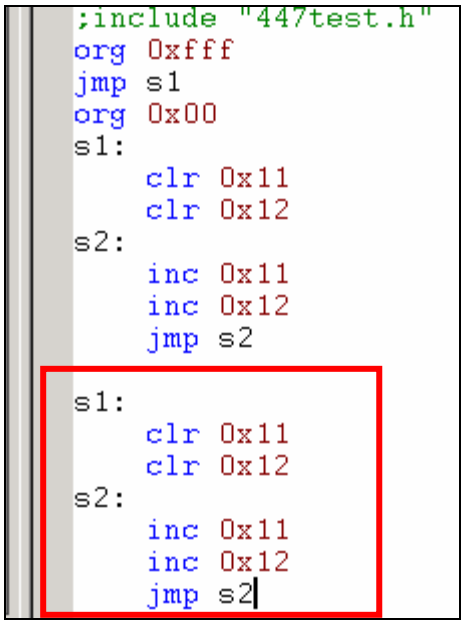


*Figure 1-3f An Example of a **Paste** Command Operation*

BookMarks – Insert markers to specific lines that you may wish to return to at later time.

1) Select a line, then from shortcut menu click **BookMarks → Toggle** from the Menu bar (or directly press the shortcut keys CTRL + F2).

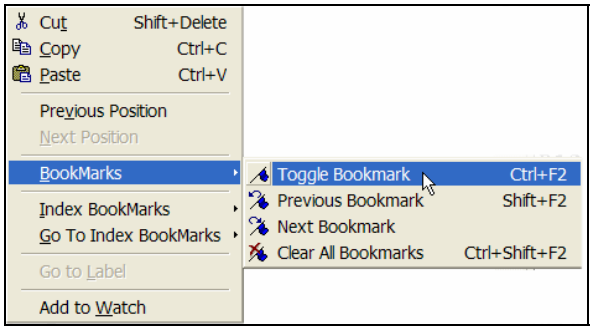


*Figure 1-3g An Example of a **BookMarks** Command Execution*

2) Then go to the bookmarked lines. For example, if you have previously bookmarked Lines 1, 5, and 8 (as shown in the sample figure below) and want to return to the lines, access **Toggle** command again. The **Previous**, **Next**, and **Clear All** commands become active this time.

Click **Previous** to go upward to the bookmarked line previous to the current position (or directly press the shortcut keys SHIFT + F2).

Click **Next** to go downward to the bookmarked line next to the current position (or directly press the shortcut keys CTRL + SHIFT + F2).

Click **Clear All** to remove all existing bookmarks.

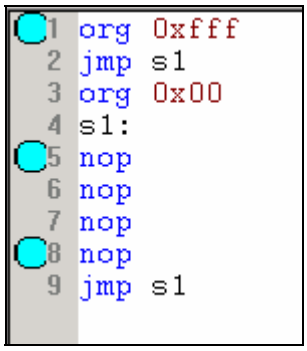


Figure 1-3h An Example of Lines Bookmarked at Lines 1, 5, & 8

Index BookMarks – Embed bookmarks with index numbers. With indexed bookmarks, you can directly access to the bookmarked line you wish to return to.

To embed index number into existing bookmarks, place cursor on the selected bookmarked line, then click **Index BookMarks → Toggle BookMarks *x*** (where "***x***" is the 0 ~ 9).

You can also directly press the shortcut keys CTRL + *x* (where "*x*" = to be assigned index number 0 ~ 9).

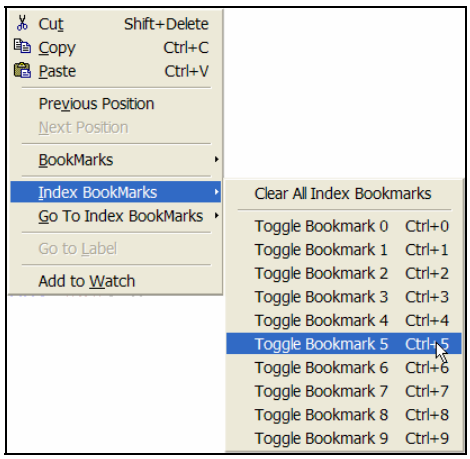


Figure 1-3i An Example of Lines Bookmarked at Lines 1, 5, & 8

The figure at right shows bookmarked Lines 5 & 7 are embedded with Index 1 & 7 respectively.

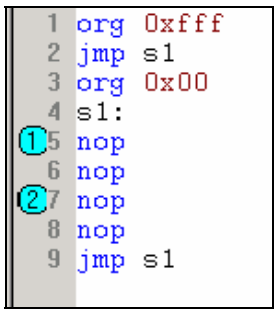


Figure 1-3j An Example of "Indexed Book-Markes" at Lines 1, & 7

Go To Index BookMarks - Go to a particular index bookmarked line.

From shortcut menu, click **Go To Index BookMarks**. Then from the resulting submenu, click on the indexed bookmark number you want access.

You can also directly press the shortcut keys ALT + x (where "x" = the target bookmark index Number 0 ~ 9).

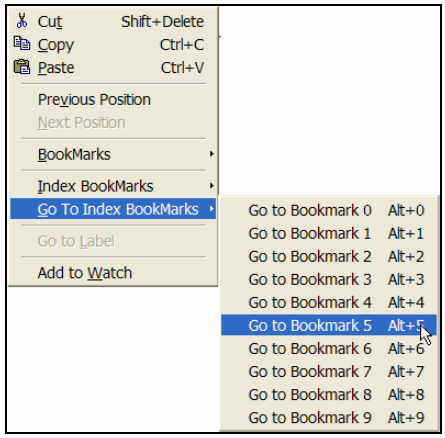


Figure 1-3k An Example of Accessing Indexed Bookmark Line 5

Go to Label of … – Find the location of the displayed label or function in the file. eUIDE will automatically pinpoint at the pertinent location in the **Editor** window (see *Figure 1-2f*, Section 1.2.2.1). At the same time, eUIDE displays the search results in the **Output** window as illustrated in *Figure 1-2g*.

Add to Watch – Watch the data change at the register locations in the **Watch** window during debugging.

After dumping project to ICE, right-click on a temporary register name (followed by "= =" symbol and **register page** or **ram bank** or **control register page**) as indicated in the example figure at right. From the resulting pop-up menu, click **Add to Watch**. Then observe Register **R11** being added into the **Watch** window. Real-time changes of the data during debugging can be observed from this window.

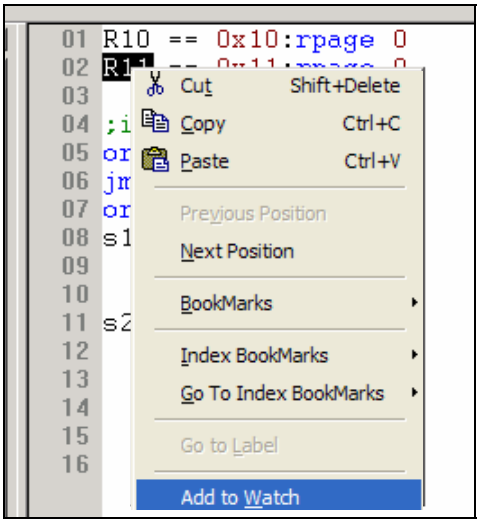


*Figure 1-3l An Example of Adding Register R11 into the **Watch** Window*

When right-clicking on a temporary register name with double–equal characters (= =), but without **register page/ram bank/control register page**, the **Watch Dialog** will display instead. See Section 1.2.2.6, *Watch Window* for further details.

1.2.2.3 Special Register Window

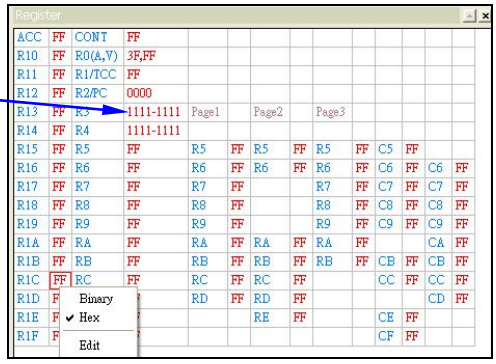


*Figure 1-4a An Example of **Special Register** Window Displaying Updated Registers*

The **Special Register** window shows the updated contents of the registers and also that of the I/O control register depending on the MCU type in use.

When the **Special Register** window is closed, it ceases to interact or read the hardware contents, except for some very special registers which is read internally.

■ **Changing Special Register Value Directly by Editing**

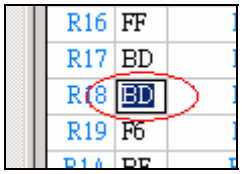


Figure 1-4b Double-Click on Selected Value (BD in this Example)

To change a special register value, double-click on the selected value (**BD** in the example shown at left).

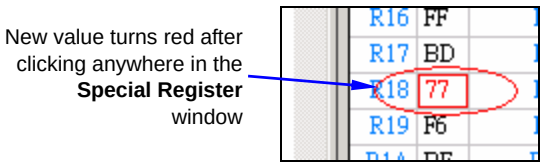


Figure 1-4c Key-in New Value (77 in this Example) to Replace the Selected Value

With the existing value highlighted, key-in the new value (**77** in the example at left figure). Observe the new value changes to red when you click anywhere within the **Special Register** window.

■ Switching Special Register Value into Binary/Hex Value

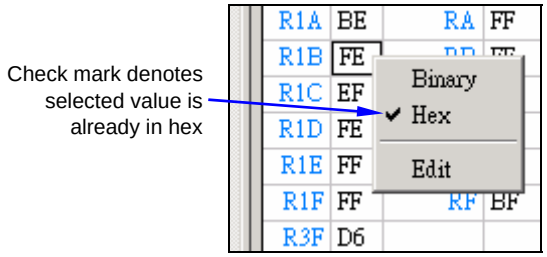


*Figure 1-4d Right-Click on the Selected Value to Display the **Binary /Hex/Edit** Commands*

Right-click on the selected value, a pop-up menu containing commands for editing the selected register value will display.

The following describes each of the menu commands**:**

Binary – Switches the register value from hex to binary format. If the value is already in binary, this command is prefixed with a check mark (✓).

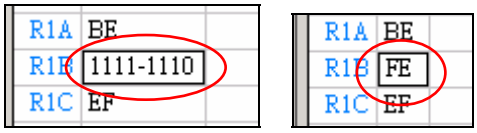


*Figure 1-4e Click **Hex** Command to Switch Binary Register Value into "Hex" and Vise-Versa*

Hex – Switches the register value from binary to hex format. If the value is already in hex, this command is prefixed with a check mark (✓).

Edit – This command auto-selects the clicked value and allows you to change the value by editing. This is the same as double-clicking on the value as explained above. However, using this **Edit** command function is preferable.

1.2.2.4 Call Stack Window

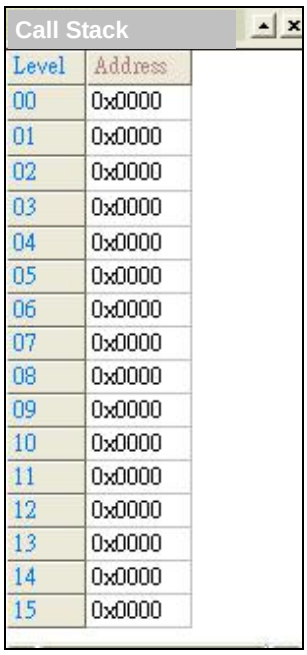


Figure 1-5a ***Call Stack*** *Window*

The **Call Stack** window shows the updated contents of call stack which indicate the actual hardware content.

When the **Call Stack** window is closed, it ceases to interact or read the hardware contents.

■ **Reading Stack Level**

In general, a stack does not have an initial value. When you press F6, eUIDE resets all stack cells to 0x0000. Due to ICE hardware design constraint, if a stack is full during program execution (as shown in the figure below) and returns to the calling sub-routine. Reading the stack level again will display the result as shown in *Figure 1-5c*. This may also affect higher-level sub-routines (as shown in *Figure 1-5d*).

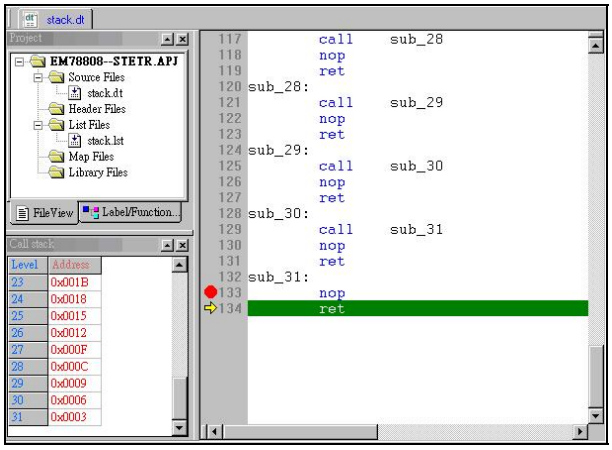


Figure 1-5b All Stack Levels are Full (as Indicated by the Breakpoint Setting)

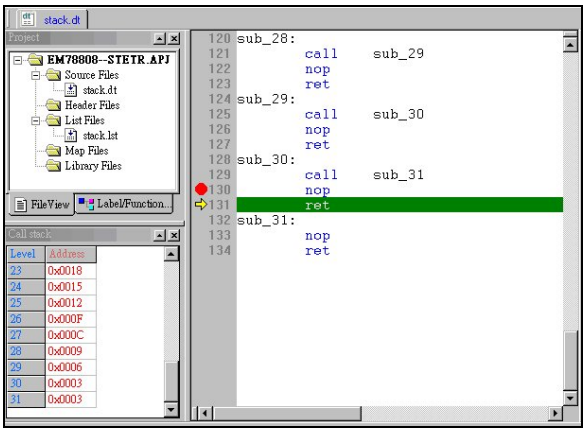


Figure 1-5c Stack Level Returns to the Preceding Stack Level (See New Breakpoint Setting)

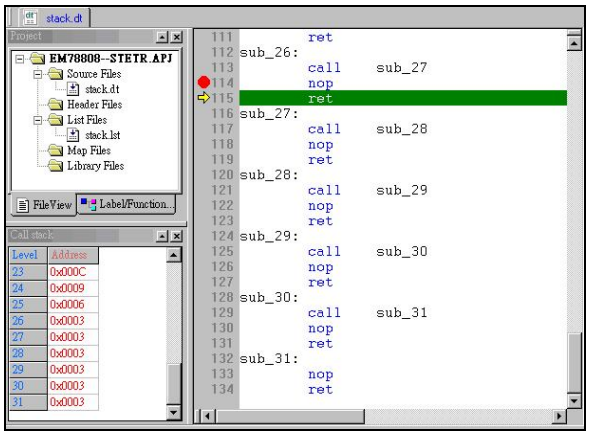


Figure 1-5d Stack Level Returns to an Earlier and Several Levels of Stack

Note that the last (next) stack level still shows 0x0003 (the value of the last stack level when all stack were full)

Note that all the subsequent stack levels still display 0x0003

■ Step-by-Step Execution to Change Stack Values

When you perform a step-by-step execution (F7), eUIDE will compare the last stack value with the stack value after the "ret" action to identify the current level position of the stack. It then changes the value of the last stack level value from 0x0003 to 0x0000 as indicated in the figure at right.

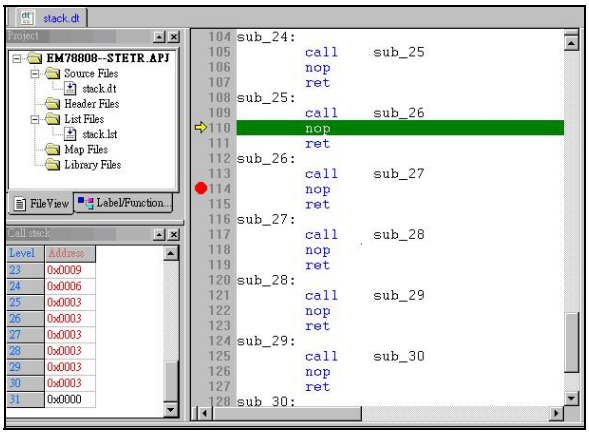


Figure 1-5e Using Step-by-Step (F7) Execution to Change the Last Stack Level Value

To set the values of the higher stacks level to 0x0000, continue to perform Step-by-Step execution. This will not affect the normal stack operation as indicated in the figure below.

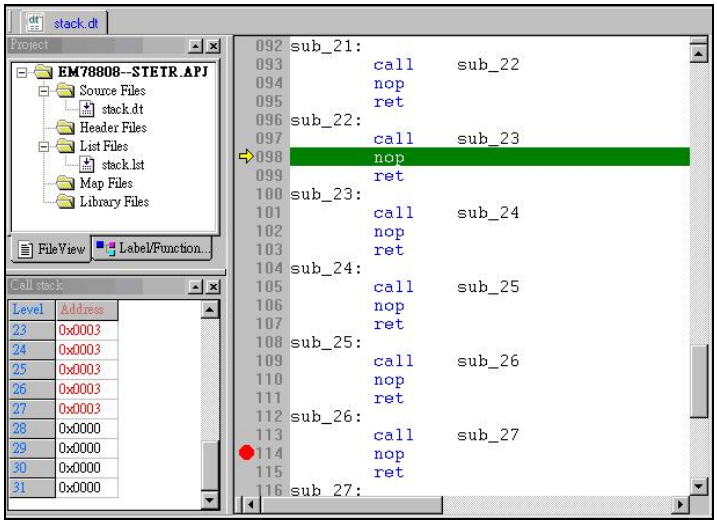


Figure 1-5f Executing Step-by-Step to Change the Values of the Higher-Level Stacks

Currently, eUIDE remains unable to determine the correct stack level position at which the ICE stops. This is due to the ICE hardware limitation which unable the eUIDE Call Stack window to properly display the correct stack level position. However, this does not affect whatsoever, the actual normal stack operation.

NOTE

*After completing the **Go** or **FreeRun** command execution, if all stack level values in **Call Stack** window display the same value, this could be caused by either the program has entered the first level of the call function, or the program has not entered any function at all. Under this condition, it is not recommended to use the **StepOut** command. Otherwise, the program will jump to the first level address in the Call Stack window.*

1.2.2.5 RAM Bank (General Registers) Window

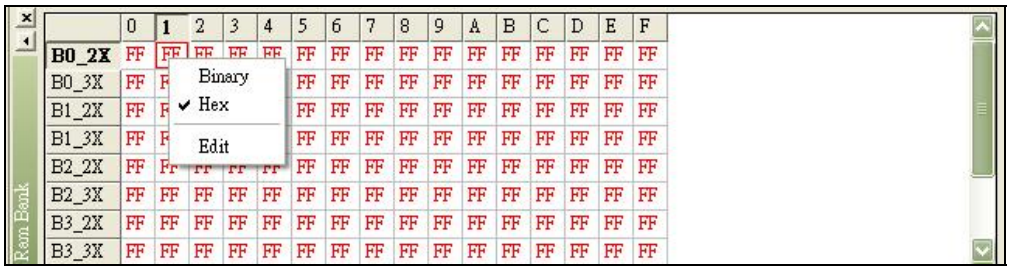


*Figure 1-6a **Ram Bank** (General Registers) Window*

The **Ram Bank (General Registers)** window shows the updated contents of the common RAM bank registers. When the **RAM Bank** window is closed, it ceases to interact or read the hardware contents.

■ Changing General Register Value Directly by Editing

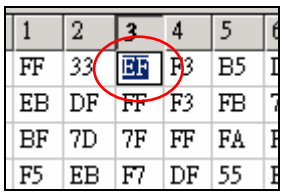


Figure 1-6b Double-Click on Selected Value (EF in this Example)

To change a special register value, double-click on the selected value (**EF** in the example shown at left).

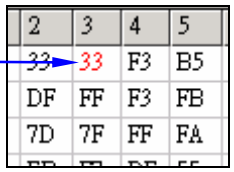


Figure 1-6c Key-in New Value (33 in this Example) to Replace the Selected Value

With the existing value highlighted, key-in the new value (**33** in the example at left figure). Observe the new value changes to red when you click anywhere within the eUIDE window.

■ Switching General Register Value into Binary/Hex Value

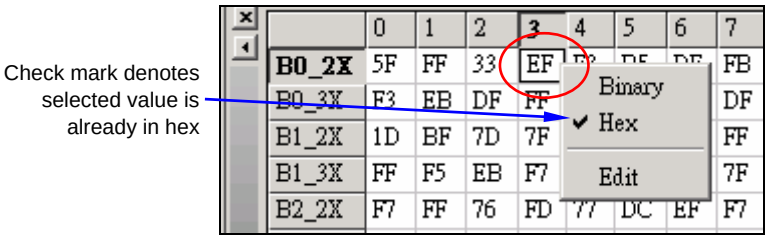


*Figure 1-6d Right-Click on the Selected Value to Display the **Binary/Hex/Edit** Commands*

Right-click on the selected value, a pop-up menu containing commands for editing the selected register value will display.

The following describes each of the menu commands**:**

Binary – Switches the register value from hex to binary format. If the value is already in binary, this command is prefixed with a check mark (✓).

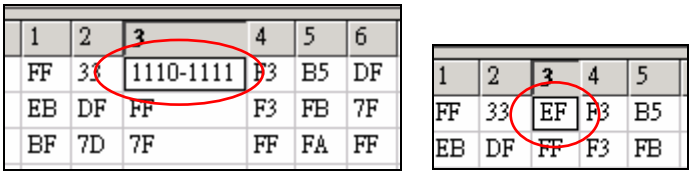


*Figure 1-6e Click **Hex** Command to Switch Binary Register Value into "Hex" and Vise-Versa*

Hex – Switches the register value from binary to hex format. If the value is already in hex, this command is prefixed with a check mark (✓).

Edit – This command auto-selects the clicked value and allows you to change the value by editing. This is the same as double-clicking on the value as explained above. However, using this **Edit** command function is preferable.

1.2.2.6 Watch Window

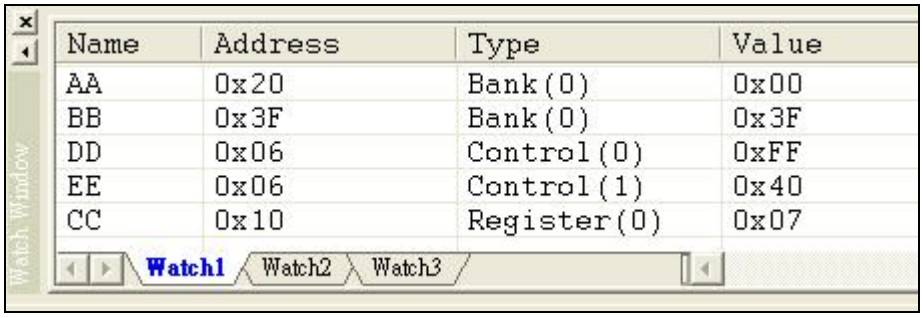


*Figure 1-7a **Watch** Window*

With **Watch** window, you can add variables that are declared in assembly file. The **Watch** window will show the defined variable information, such as name, contents, bank, and address. Refer to Section 4.2, *Statement Syntax*; for more details on assembly codes.

To view real-time changes of selected register values from the **Watch** window during debugging, the register values have to be entered into **Watch** window. To do so, do one of the following (three methods are available)**:**

■ Right-Clicking the Selected Register Value Directly from Editor Window

1) Right-click a register value (or variable) from the **Editor** window. From the resulting pop-up menu, click **Add To Watch** command.

   If the selected value contains **register page** (or **ram bank** or **control register page** as shown in the right figure), the register value is directly displayed in the **Watch** window (see Section 1.2.2.2, *Editor Window → "Add to Watch")*.

   If the selected value does not contain **register page/ram bank/control register page** (as shown in the figure below), the **Watch Dialog** (see *Figure 1-7d* below) will display instead.

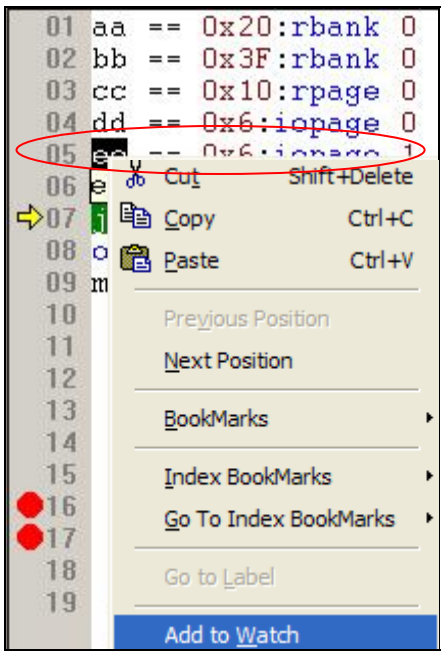


*Figure 1-7b Select & Right-Click Register Value "ee" (with Register Page Register Page) from **Editor** Window, Pop-Up Menu then Displays*

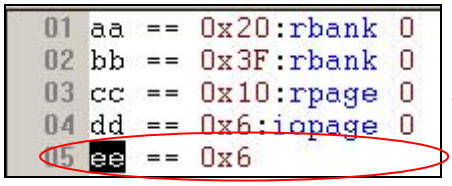


Figure 1-7c Select & Right-Click Register Value "ee" (without Register Page), "Watch Dialog" Displays

2) From the **Label Name** box of **Watch Dialog** window, select and double-click a variable label name you wish to assign. Then from the **Label Types** options, select the variable type to be displayed, i.e., **Special Register**, **Control Register**, **RAM(Bank)**, or **Call ID RAM** (see figure at right).

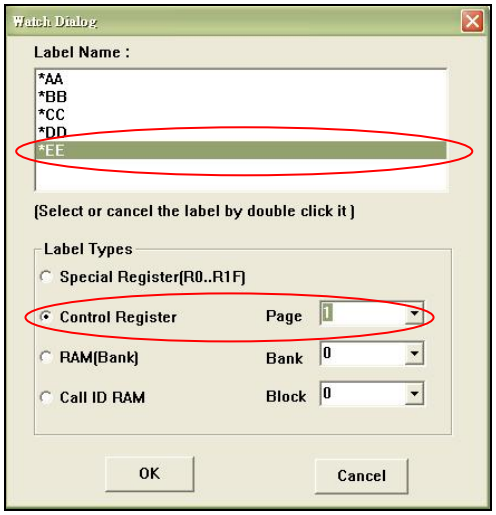


*Figure 1-7d eUIDE **Watch** Dialog*

3) Click the **OK** button to add and display the selections into the **Watch** window as shown below.

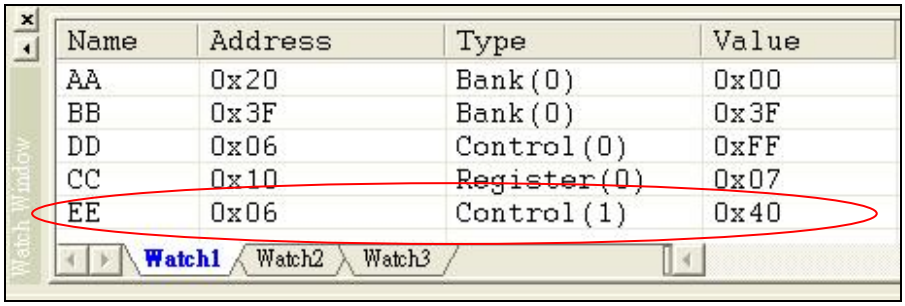


*Figure 1-7e Selected Register Value "ee" Displayed in **Watch** Window*

■ Accessing "Debug" → "Add Label to Watch" Command from Menu Bar

1) Select a register from the **Editor** window and from Menu Bar, click **Debug → Add Label to Watch**.

2) From the resulting **Watch Dialog**, select and double-click a label name (see *Figure 1-7d* above). Then from the **Label Types** options, select the variable type to be displayed, i.e., **Special Register**, **Control Register**, **RAM(Bank)**, or **Call ID RAM**.

*Figure 1-7f **Debug** Pull-Down Menu from eUIDE Menu Bar*

3) Click the **OK** button to add and display the selections into the **Watch** window as shown in *Figure 1-7e* above.

NOTE

*The variables displayed in the **Label Name:** list box of the **Watch Dialog** are the same variables (without **register page/ram bank/control register page**) that you have defined in the program code using the double–equal characters "==". Double-clicking a variable name will add or clear the asterisk "*****". An asterisk prefixed to variable name indicates that the variable is selected. Click **OK** button to add the selected variable to the **Watch** window.*

■ Accessing "Option" → "View" Command from Menu Bar (Mass Selection by Labeling)

1) From Menu Bar, click **Option → View Setting** (see figure at right). From the resulting **View Setting** dialog, select the **Add defined label to watch automatically** option (see *Figure 1-7i* below). There are three sorting options in eUIDE Version 2.6 or later. Labels added into **Watch** window will be sorted automatically according to the selected sorting option (sample figure shows "sort by name" option selected). Then Click **OK** button.

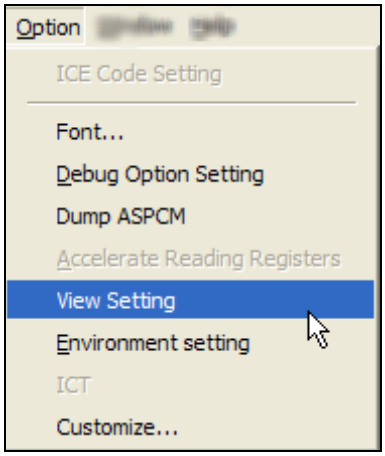
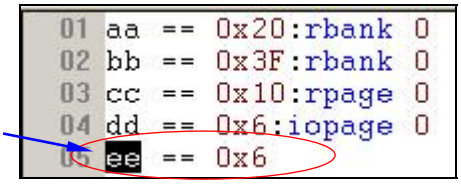


Figure 1-7g ***Option*** *Pull-Down Menu from eUIDE Menu Bar*

2) All variables with **page location data** are automatically formatted into labeled format in the **Editor** window as shown in the example below.



Variable without **page location data**, not labeled

Figure 1-7i Labeled Format of Variables with Page Location Data

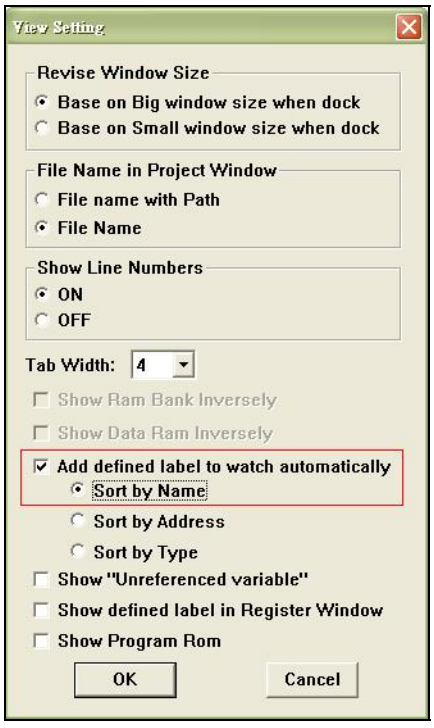


Figure 1-7h eUIDE ***View Setting*** *Dialog*

3) After code dump, the variables are automatically included in the **Watch** window as illustrated in the sample below.

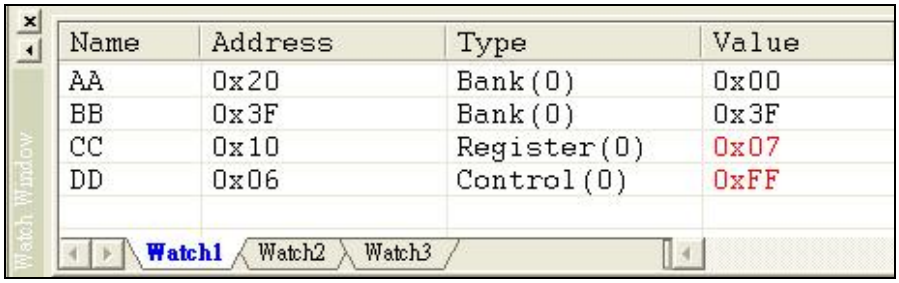


*Figure 1-7j Labeled Register Values Display in **Watch** Window*

NOTES

1. *To remove a variable from the **Watch** window, select the variable and press DELETE from the keyboard.*
2. *If the "Add defined label to watch automatically" check box is selected, labels added into **Watch** window will be sorted according to the selected sorting option (one of the 3 options).*
3. *The labeled variables are placed after the end of the existing **Watch** window variables that were input manually using the two methods described earlier.*
4. *When executing "Project"→"Dump to ICE (F3)" command in "C" Mode, all global labels will display from the **Information** sub-window of **Output** window (illustrated below).*
   - *First field is the type of variable and page number.*
   - *Second field is the register or ROM's address of the variable.*
   - *Third field is the name of the variable.*

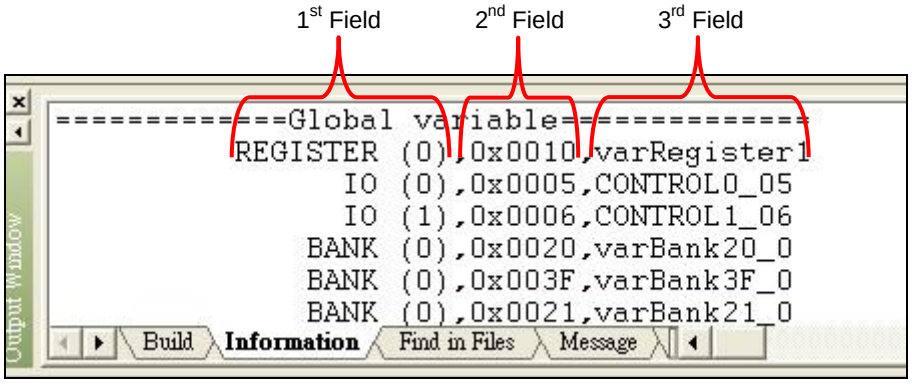


*Figure 1-7k Global Variables Display from **Information** Sub-Window of **Output** Window*

1.2.2.7 Data RAM Window

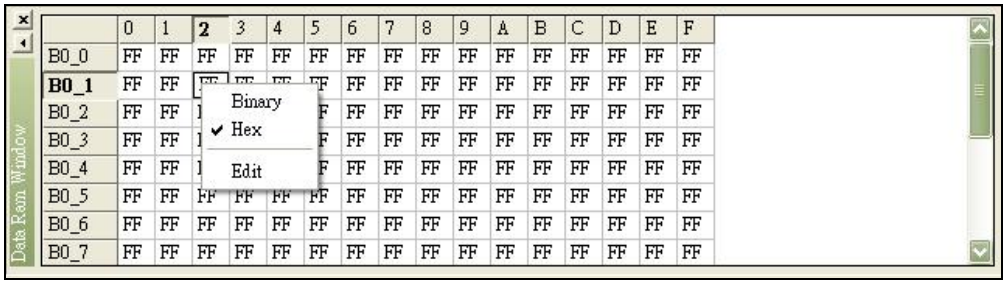


*Figure 1-8 **Data RAM** Window with a Right-Clicked Binary Value*

The **Data RAM** window is accessible only if RAM is available from the target microcontroller currently in use. The **Data RAM** window shows the contents of the data RAM. To change the data RAM values is the same as changing the special register values described in Section 1.2.2.3, *Special Register Window*.

Note that when the **Data RAM** window is closed, it ceases to interact or read the hardware contents.

1.2.2.8 LCD RAM Window

With the EM78 Series IC that supports LCD installed and in used, open the **LCD RAM** window. Drag a corner to increase the size of the **LCD RAM** window until the window looks as shown in the following figure.

Note that when the **LCD RAM** window is closed, it ceases to interact or read the hardware contents.

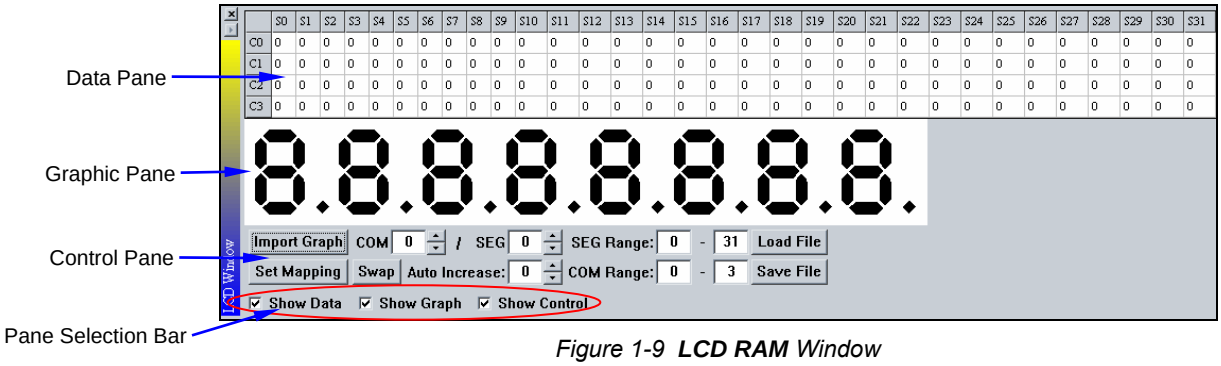


*Figure 1-9 **LCD RAM** Window*

The **LCD RAM** window consisted of four sections as detailed below**:**

1) Data Pane

This pane displays the contents of the **LCD RAM**. "**Cx**" denotes the LCD signal "COM x." "**Sx**" denotes LCD signal "Segment x."

To modify the contents of the LCD RAM elements double click on the selected element (grid block). The color of the selected element will change from pink (1) to white (0) and vise-versa. Any related messages will be shown in the **Output** window.

2) Graphic Pane

This pane displays the status of the loaded BMP graphic. If no BMP graphic is loaded, no display is shown.

3) Control Pane

This pane controls the link between the BMP graphic and the data, as well as setting the COM/SEG values for each graphic segment.

4) Pane Selection Bar

You can select whether to display any or all of the three panes described above.

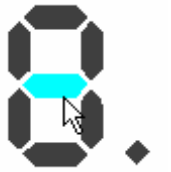
■ Loading Graphic Display and Segment Settings

The following is the step-by-step procedure on how to load the graphic display and define their segment settings:

1) From the Control Pane click **Import Graph** button to load the BMP file. The program will automatically convert the graphic file into black and white colors and determines which black pixels are linked together to establish segments.

2) Click **Set Mapping** button and the black segments fades to gray to indicate that the system is now in Mapping Mode. At the same time, the "**Set Mapping**" button label changes to "**Done.**" Click the **Done** button only after all mapping setting processes are completed.

3) Define COM/SEG values in the Control Pane and point at a segment to apply the value. Observe the segment turns to blue color. This indicates that the pointed segment is now active.

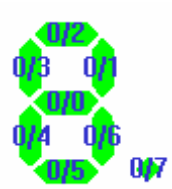


4) Click on the active segment to enter the Control Panel defined COM/SEG values into the segment. The example shows the defined COM & SEG values are "0."

5) Move the pointer away from the segment and the segment turns into green. This indicates that the system has already saved the COM/SEG values for the particular segment.

6) Then move the pointer to other segments and repeat the steps described above to redefine and change the COM/SEG values of the remaining segments.

   An automatic and faster way of re-defining and setting of COM/SEG values is explained in the next topic.



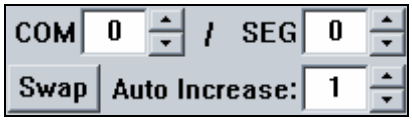
7) After all segment values are set, click **Done** (the original **Set Mapping**) button. Then run the program to test the settings, e.g., set some breakpoints and execute "**GO"** or use the "**Continue step into.**" View the results and check for error.

■ More Efficient Way of Re-Defining COM/SEG Values

The Step 6 above shows how to manually define and change the COM/SEG values for each segment. This method is okay if there are only a few segments involved. However, if a large number of segments are involved, the task becomes complicated and time consuming. The following steps explain a more efficient way of defining a large number of segment values:

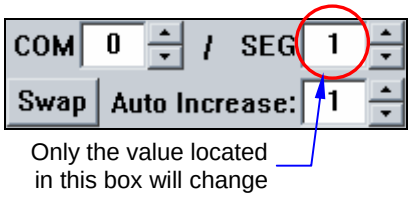
1) From the Control Pane, define the SEG Range as 0~1, and the COM Range as 0~3. Consequently, the segment values will be set according to these defined ranges.

2) Define COM/SEG values to "0" and Auto Increase value to "1."

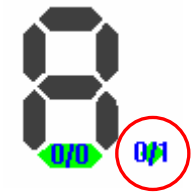


3) Point to the segment where the COM/SEG value (0/0) is to be applied. Click and you should see the segment set as "0/0."

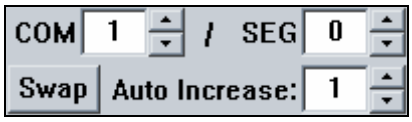
4) Observe that as soon as the "0/0" value is entered on the segment, the Control Pane SEG value is set to Auto-Increase by "1." Note that only the value located at the right box (SEG value in this case) will change.



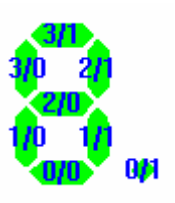
5) Thus, there is no need to manually change the COM/SEG value when setting "0/1" value for the next segment. Just directly click on the segment and the next SEG value is Auto-Increased by "1."



6) Due to the fact that the SEG Range is set at 0~1 (see Item 1 above), the next or 3rd segment you click cannot increase its SEG value to "2." Hence, the system automatically allocates the auto-increase value of "1" to COM and sets the SEG value back to "0."



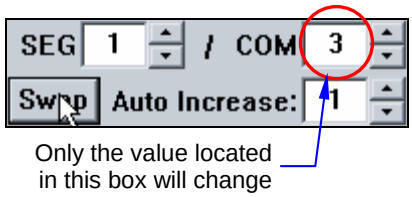
7) Consequently, all you need to do is to continuously click on the remaining segments to set their COM/SEG values within the defined range.



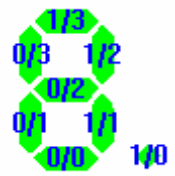
■ Other Convenient Functions of the LCD RAM Window

In addition to the above, other convenient functions of the LCD RAM window are also available and are describe below together with its respective notes**:**

1) If you wish to automatically apply the Auto Increase value to COM (instead of SEG), click **Swap** button and observe SEG & COM swaps positions.



2) After clicking **Swap**, also observe that the system has at the same time, switched the positions of all values (from COM/SEG to SEG/COM) on the previously set segments. Compare the figure at right to the one in Item 7 above and see the difference.

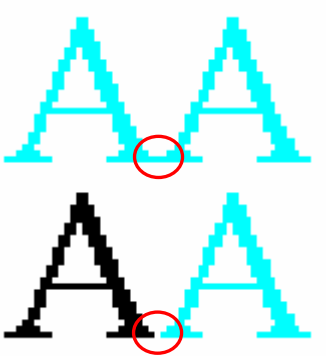


3) After setting all the relative values to affected segments, it is recommended you click **Save File** button to save the setting into LCD Simulator Data File (*.LCD). Otherwise, when you want to use the program the next time, you will need to define and set the values again.

4) Since the LCD Simulator Data File already includes the imported graphic files, you do not have to click the **Import Graphics** button the next time you want to use the saved *.LCD file. Just click **Load File** button to load the data file.

5) The program determines which black pixels are linked together to establish a segment. However, when 2 or more separate segments constitute a non-separable object or character (e.g., *i*, *j*), all integral segments should be assigned with the same COM/SEG value. Otherwise, the program will assume that there are 2 or more separate segments.

6) As stated above, the program determines which black pixels are linked together to establish a segment. However, when 2 or more segments which are ought to be separated and set with different COM/SEG values, are somehow joint together at one point, such graphic file should be modified to break up their link. Otherwise, the program will incorrectly assume that there is only one single segment.



1.2.2.9 EEPROM Window

Click **Refresh** button to restore updating & reading of all EEPROM data

Click row header to disable and exclude row data from updating & reading. Click again to restore

EEPROM

Refresh	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	FF	FF	FF	FF	FF	FF	FF	55	55	55	55	55	55	55	55	55
10	55	55	55	55	55	55	55	55	55	55	55	55	55	55	55	55
20	55	55	55	33	33	33	33	33	33	33	33	33	33	33	33	33
30	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
40	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
50	33	33	33	33	33	33	33	33	33	33	33	33	33	33	33	33
60	33	33	33	33	33	33	33	33	33	33	33	33	33	33	33	33
70	33	33	33	33	33	33	33	33	33	33	33	33	33	33	33	33
80	33	33	33	33	33	33	33	33	33	33	33	33	33	33	33	33

*Figure 1-10 **EEPROM** Window Showing Rows 30 & 40 Disabled*

The **EEPROM** window is accessible only if EEPROM is available from the target microcontroller currently in use. The **EEPROM** window shows the contents of the data **EEPROM**.

Note that when the **EEPROM** window is closed, it ceases to interact or read the hardware contents.

Reading all EEPROM data is time consuming. It needs almost 8 seconds to read 256 bytes. To minimize reading time, you can disable the EEPROM data that you do not need to read by clicking the row header of each row that you do not wish to read. The disabled rows will have their data dimmed and are excluded from updating process. To restore updating to the disabled rows, click the row header. If you want to update all EEPROM data, just click **Refresh** button on top-left of the window. All data are then updated.

1.2.2.10 Output Window

*Figure 1-11a **Output** Window*

The **Output** window displays messages indicating the results (including errors) of the project compiling just performed, such as assembler, linker, trace log history, and debugging. The window consists of four tab sub-windows, namely; **Build**, **Information**, **Find in Files**, **Message**, and **Program Rom,** where**:**

Build – displays assembler/linker related messages and trace logs. Double click on the error message to link to the corresponding program text line where the source of error occurs. The pertinent source file is automatically opened in the **Editor** window if it is not currently active.

Information – displays debugging related ROM and RAM Bank memory usage information.

Find in Files – allows you to find identical string (selected from an active file) from other active or inactive files in your folder. Lines containing the identical string will display on the **Output** window complete with its source filename and directory. See example below.

Message – displays the debugging related changes to the **LCD RAM** window.

Program Rom – displays the contents of program ROM after dump.

■ Executing Find Command from Output Window

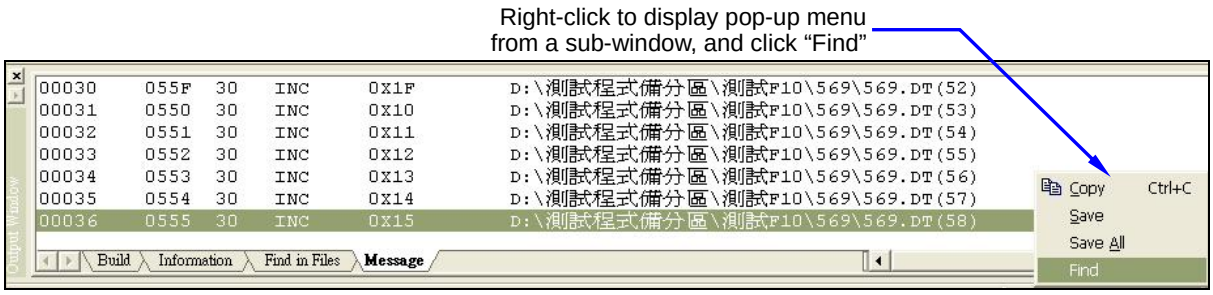


*Figure 1-11b Finding a String from any of the **Output** Sub-Windows*

The steps for initiating the **Find** dialog from **Output** window:

1. From one of the **Output** sub-windows, right-click within the widow. A command popup menu then displays.
2. From the pop-up menu, click **Find** command.
3. The following **Find** dialog then displays.

*Figure 1-11c **Find** Dialog from **Output** Window*

Where**:**

Find: Enter the word string you want to search

Find Next: Click this button to search the next matching string (Hotkey**: Alt + N**)

Cancel: Quit search and exit from the dialog

Direction Up: Search forward (Hotkey**: Alt + U**)

Direction Down: Search backward (Hotkey**: Alt + D**)

Match whole word only: Enable check box to find and match whole word only. (Hotkey**: Alt + W**)

Match case: Select this option to match the string in lowercase or uppercase characters exactly as they appear. (Hotkey**: Alt + C**)

Move with code line: When this check box is enabled, the **Output** window will indicate the search matched line in the trace log. At the same time, the corresponding trace log mark code line in the **Editor** window is highlighted (Hotkey**: Alt** + **M**)

NOTE

*The above hotkeys works only when the **Find** dialog is active.*

*When **Find** dialog is inactive, and the eUIDE is active, use **Ctrl + R** to find forward and **Ctrl + Q** to find backward.*

1.2.3 eUIDE Menu Bar

See Chapter 2 for details.

1.2.4 ToolBar

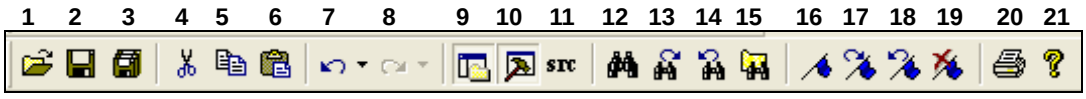


Figure 1-12a eUIDE Main Window (Standard) Toolbar

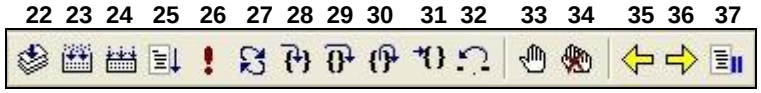


Figure 1-12b eUIDE Main Window (Build) Toolbar

1.2.4.1 Toolbar Icons and its Functions and Hotkeys

Corresponding hot key is enclosed in parenthesis**:**

1 **Open:** open an existing file (Ctrl + O)

2 **Save:** save current active document (Ctrl + S)

3 **Save All:** save all document

4 **Cut:** remove the selected string to clipboard (Shift + Del)

5 **Copy:** copy the selected string to clipboard (Ctrl + C)

6 **Paste:** paste the string from clipboard (Ctrl + V)

7 **Undo:** cancel the last editing action (Alt + Backspace)

8 **Redo:** cancel the last "undo" i.e., restore the "undone" editing action ((Ctrl + Y))

9  **Open/Hide Workspace:** display/hide toggle for **Project** window

10  **Open/Hide Output:** display/hide toggle for **Output** window

11  **Only Source Window:** maximize editor window (Ctrl +Shift +S)

12  **Find:** find string from within the entire active file (Ctrl + F)

13  **Find Next:** find string from cursor position toward the end of file (Ctrl + N)

14  **Find Previous:** find string from cursor position toward the beginning of file (Ctrl + P)

15  **Find in Files:** find string from inactive files

16  **Toggle Bookmark:** apply/remove bookmark on the line where cursor is positioned (Ctrl + F2)

17  **Go to Next Bookmark:** jump to the next bookmark from cursor position toward the end of file

18  **Go to Previous Bookmark:** jump to the next bookmark from cursor position toward the beginning of file (Ctrl + F2)

19  **Clear All Bookmarks:** clear all bookmarks (Ctrl + Shift + F2)

20  **Print:** print the active file

21  **About:** about eUIDE version and other information

22  **Assemble (or Compile in C mode):** assemble (or Compile) the active file in the **Editor** window (Alt + F7)

23  **Build:** assemble (or compile in C mode)the modified files in the project and link object files (Shift+ Alt + F9)

24  **Rebuild All:** assemble (or compile in C mode) all files in the project and link object files (Alt + F9)

25  **Go:** auto dump and execute program with the effect of the breakpoints (F5)

26  **Free Run:** auto dump and execute program with the breakpoints ignored (F10)

27  **Reset:** reset the ICE (F6)

28  **Step Into:** auto dump and execute program step by step including subroutines (F7)

29  **Step Over:** auto dump and execute program step by step excluding the subroutines (F8)

30  **Step Out:** auto dump and execute program until exit from subroutines (Ctrl + F7)

31 **Go to Cursor:** auto dump and execute program, then stop at the cursor position while ignoring the breakpoint (F4)

32 **Run from Selected Line:** Start running command from the line where the cursor is located

33 **Toggle Breakpoint:** insert/remove toggle for breakpoints (F9)

34 **Clear All Breakpoints:** remove all breakpoints

35 **Trace Back:** Trace the executed trace log backward from the last executed address to the address before the current executed address.

36 **Trace Forth:** Retrace the trace log address forward (top to bottom) or at reverse direction of **Trace Back** command

37 **Stop:** Stop Free Run or Go without break point.

■ **Chinese Characters Code Conversion Hotkeys**

Applicable to eUIDE source files written and used in Chinese speaking areas.

1) **Ctrl + Alt + Shift +G:** Convert Traditional Chinese (Big5) to Simplified Chinese (GB)

2) **Ctrl + Alt + Shift +B:** Convert Simplified Chinese (GB) to Traditional Chinese (Big5)

3) **Ctrl + Alt + Shift +Z:** Undo the last conversion

1.2.4.2 Document Bar

Figure 1-13 eUIDE Main Window Document Bar

The **Document** bar displays the file icons representing each of the opened files in the **Editor** window. Click the icon of the pertinent file that you wish to activate and place in front of the **Editor** window to perform editing. Highlighted filename is the active file (function is similar with taskbar buttons under Windows).

Double clicking on opened file icon in the **Document** bar will close the active document.

1.2.4.3 Status Bar

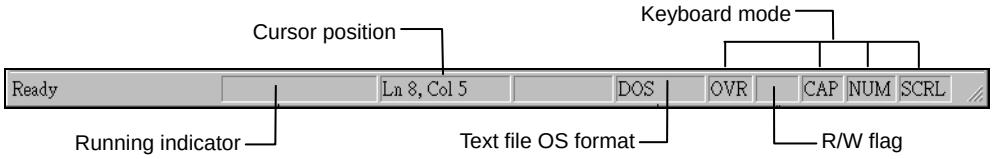


Figure 1-14 eUIDE Main Window Status Bar

A eUIDE running indicator will be shown in the **Status** bar while your project is being compiled.

The Cursor position indicates the cursor location within the text **Editor** window.

R/W flag indicates the active file Read/Write status. If Read only, “Read” will display, otherwise the field is empty.

Keyboard mode displays the status of the following keyboard keys**:**

- **Insert** key – OVR is dimmed when overtype mode is off, highlighted when on.
- **Caps Lock** key – CAP is dimmed when uppercase character mode is off, highlighted when on.
- **Num Lock** key – NUM is dimmed when the numeric keypad calculator mode is off, highlighted when on.
- **Scroll Lock** key – SCRL is dimmed when cursor control mode is off, highlighted when on.

Chapter 2
The eUIDE Commands

2.1 eUIDE Menu Bar and its Menu Commands

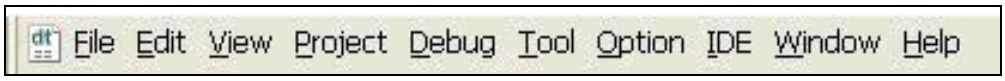


Figure 2-1 eUIDE Menu Bar

2.1.1 File Menu

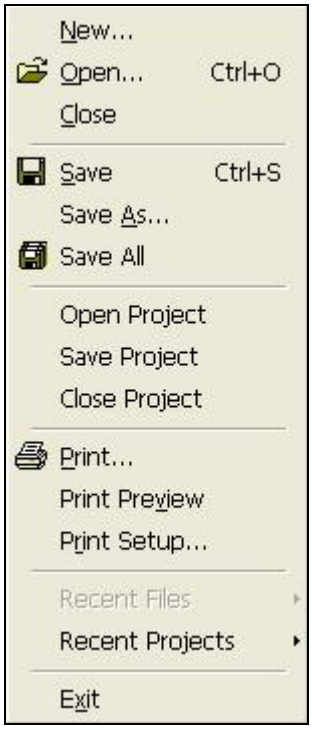


*Figure 2-2 **File** Menu*

New…	Create a new project or source file
Open…	Open an existing document or project
Close	Close the active document or project
Save	Save current active document
Save As	Save current active document with new filename
Save All	Save all opened documents
Open/Save/Close Project	Open/Save/Close the active project
Print	Print active file
Print Preview	Preview printed format of active file
Print Setup…	Define printer settings
Recent Files	View the record of the recently opened file(s)
Recent Projects	View the record of the recently opened project
Exit	Exit from eUIDE Program

2.1.2 Edit Menu

*Figure 2-3a **Edit** Menu*

Undo Cancel the last editing action

Redo Repeat the last editing action

Cut/Copy/Paste Same as standard clipboard function

Select ALL Select all contents of the active window

Go to Line… Move cursor to the defined line number within the active window

Formation Selection Formats the selection using smart indent settings[1]

Find… Find the defined string in the active window

Find Next Find the defined string toward the bottom of the active window

Find Previous Find the defined string toward the top of the active window

Find in Files Find the defined string in the active and non-active files

Replace… Same as standard “find and replace” editing functions

Bookmarks Bookmark the line at cursor position

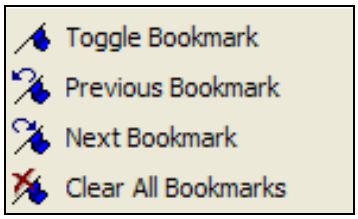


*Figure 2-3b **Bookmarks** Sub-Menu*

Toggle[2] Bookmark the line at cursor position

Previous[2] Jump to next bookmark from cursor position toward the end of file

Next[2] Jump to next bookmark from cursor position toward the beginning of file

Clear All[2] Clear all bookmarks

Index Bookmarks Clear all bookmarks or assign an index value (0~9) to the bookmarks in order to easily access (jump) them using the “**Go to Index Bookmarks**” command below

Go to Index Bookmarks Jump to bookmark with “x” index value

[1] *Supported in C Project only*

[2] *Submenu from **Bookmarks** command*

2.1.2.1 Executing Find Command from Edit Menu

1) From menu bar, click **Edit → Find**. Alternatively, you can press the shortcut keys **CTRL + F**.
2) From the resulting dialog, enter the string you want to find and its parameters.

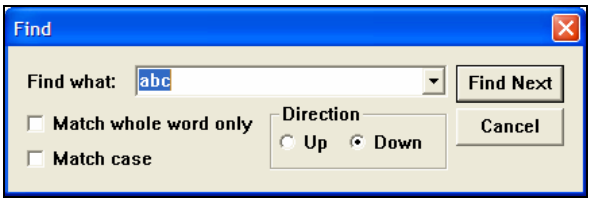


*Figure 2-4b **Find** Dialog from **Edith Menu***

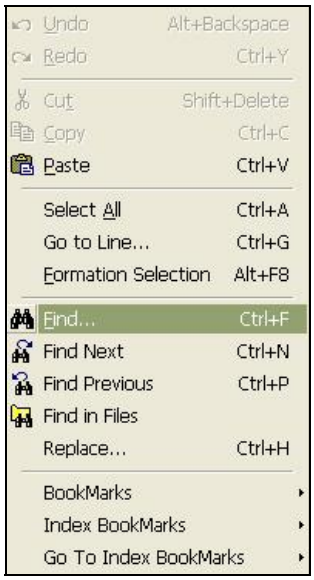


*Figure 2-4a **Find** Command from **Edit** Menu*

Where:

Find what: Enter the word string you want to search

Find Next: Click this button to search the next matching string. Matched string is highlighted

Match whole word only: Select this option to search string that matches the string as a whole word. For example, if you want to search for "*abc*," the function will NOT pick out string that has additional characters (including spaces) before or after the defined string, such as "*123abc*" or "*abced*".

Match case: Select this option to match the string in lowercase or uppercase characters exactly as they appear.

Direction Up: Search toward the top of the document

Direction Down: Search toward the bottom of the document

Cancel: Quit search and exit from the dialog

The figure at right shows an example where the **Find** command found and highlights the string which matches with the search "*mov*" string.

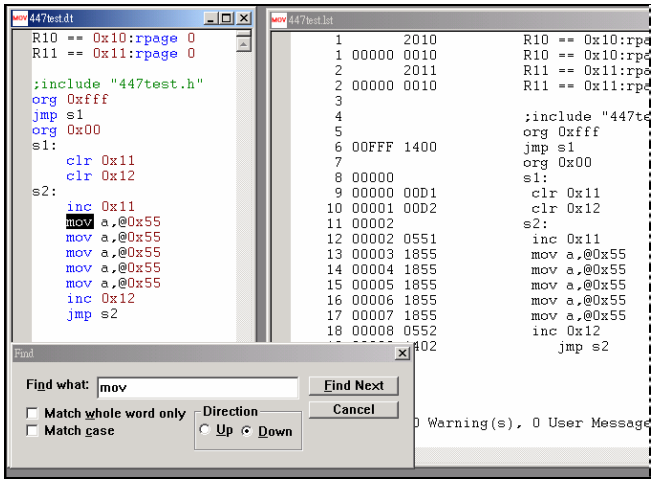


Figure 2-4c An Example of Highlighted Matched String

2.1.2.2 Executing Find Command with Shortcut Hotkeys

1) From the **Find** dialog, enter the string you want to search.
2) To search upward, press the shortcut keys **CTRL + P**.
3) To search downward, press the shortcut keys **CTRL + N**.

2.1.3 View Menu

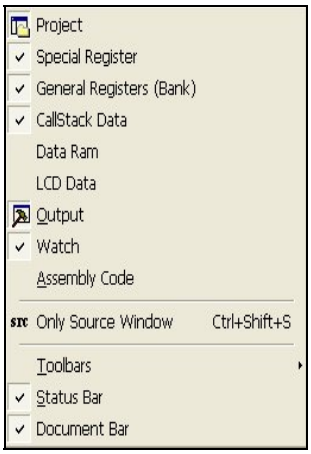


*Figure 2-5 **View** Menu*

Project	Show/hide **Project** window
Special Registers	Show/hide **Special Register** window
General Registers (Bank)	Show/hide **General Register (Bank)** window
Call Stack Data	Show/hide **Call Stack** window
Data RAM	Show/hide **Data RAM** window (if supported by the target chip)
LCD Data	Show/hide **LCD Data** window (if supported by the target chip)
Output	Show/hide **Output** window
Watch	Show/hide **Watch** window
Assembly Code[1]	Show/hide Assembly Code in/from **Editor** window After first dump to ICE (F3), press this command. The assembly code will then display with C source
Only Source Window	Maximize the **Editor** window
Toolbars	Show/hide Standard, Build, or both toolbars
Status Bar	Show/hide Status bar
Document Bar	Show/hide Document bar

[1] *Supported in C Project only*

2.1.4 Project Menu

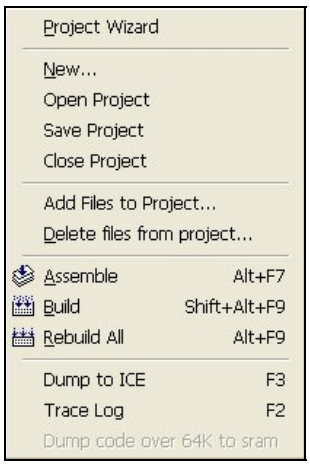


*Figure 2-6 **Project** Menu*

Project Wizard Create a new project with step-by-step dialog (see Section 3.4.1 for details)

New… Create a new project with **New** dialog (see Section 3.4.2 for details)

Open Project Open an existing project

Save Project Save the active project together with all related files

Close Project Close the active **Project** window

Add Files to Project… Add the existing source file into project

Delete Files from Project… Remove source file from project

Assemble (or **Compile** under C Project) Assemble (or compile) the active file window. If errors occur during assembly (or compiling) time, error messages will be shown in the **Output** window. Otherwise, "*0 errors, 0 warnings, 0 users*" will be displayed.

Build Assemble (or compile) the files that have been modified, and link them to the currently opened project.

Rebuild All Assemble (or compile) all files regardless of having been modified or not, and link them to the currently opened project.

Dump to ICE Dump the program code to ICE

Trace Log Refers to available history only when either **Go**, **Free Run**, or **Go To Cursor** command from the **Debug** menu is executed. The maximum length of trace log is 8K words.

> **NOTE**
>
> *With **LPT** connection, the trace log will record the unexecuted instruction next to the last executed one.*
>
> *With **USB** connection, the last executed instruction will be recorded.*

Dump code over 64K to sram[1] Dump the page range program code of over 64K to the SRAM. The SRAM must be plugged into the ICE hardware when executing this command. See example in the following figure.

[1] *This command applies to **EM78815** only.*

2.1.4.1 Executing "Dump code over 64K to sram" Command

When the **Dump code over 64K to sram** command is executed, the following dialog displays. Enter "64" into the **Start Page** box and "127" into the **End Page** box.

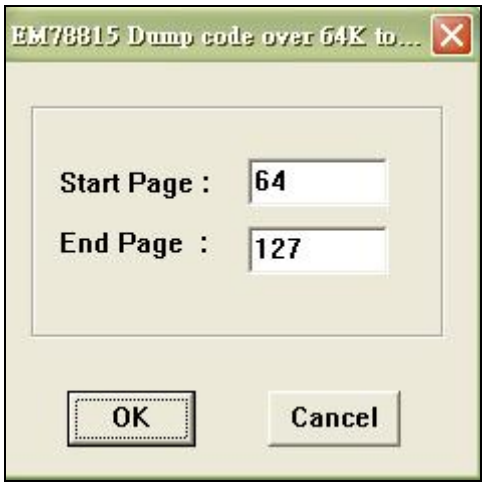


*Figure 2-7 "**Dump code over 64K to sram**" Dialog*

2.1.5 Debug Menu

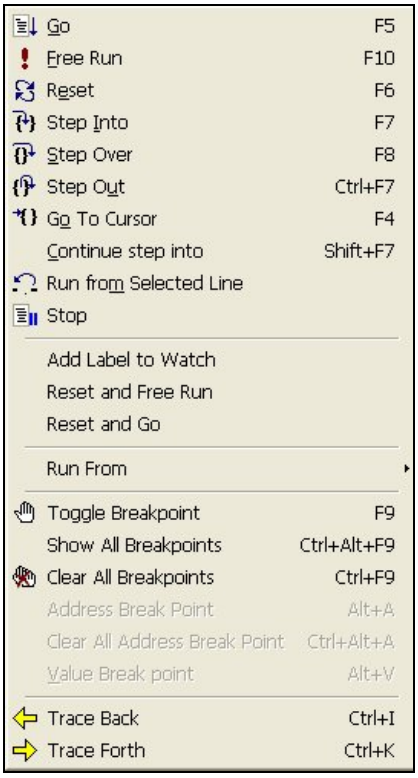


*Figure 2-8 **Debug** Menu*

Go Run program starting from the current program counter until a breakpoint is matched

Free Run Run program starting from the current program counter until the **OK** button of the **Stop Running** dialog is clicked

Reset Perform ICE reset (register contents are displayed with initial values)

Step Into Execute the instructions step-by -step (with register contents updated simultaneously)

Step Over Execute instructions as **Step Into** (see above), but the CALL instruction will execute as "step over"

Step Out	Exit CALL subroutines while executing **Step Into** in CALL subroutines
Go to Cursor	Run program starting from the current program counter up to the location where the cursor is anchored (applies to ICE debug mode only)
Continue step into	Perform **Step into** command function non-stop
Run from Selected Line	Start running command from the line where the cursor is located
Stop	Stop **Free Run** or **Go** command execution without matching breakpoint
Add Label to Watch	Add or delete variables from the **Watch** window. See details in Section 1.2.2.6, *Watch Window*.
Reset and Free Run	Reset hardware (ICE), and then execute the **Free Run** command function
Reset and Go	Reset hardware (ICE), and then execute the **Go** command function
Run From	See Section 2.1.5.2 below for the resulting sub-menu and its functions
Toggle Breakpoint	Set or remove a breakpoint
Show All Breakpoints	Show all breakpoints setup data in the **Output** window
Clear All Breakpoints	Clear all breakpoints
Address Breakpoint	Define addresses for the breakpoints. See Section 2.1.5.3 below for the resulting sub-menu and its functions
Clear All Address Breakpoints	Clear all address breakpoints
Value Breakpoint	TCC, R3 value breakpoint for PC Peripheral IC
Trace Back	Trace the executed trace log backward from the last executed address to the address before the current executed address. The highlighted address moves backward step- by-step each time the command is clicked.
Trace Forth	Retrace the trace log address forward (top to bottom) or at reverse direction of **Trace Back** command. Applicable only after **Trace Back** command is performed.

2.1.5.1 "Run From" Command Sub-Menu Function Description

Where:

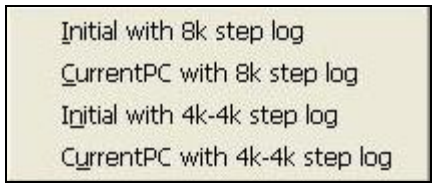


Figure 2-9 ***Run From*** *Command Sub-Menu*

Initial with 8K Step log:
Program is kept running starting from the initial address until a breakpoint is matched. Only the last 8K steps of execution history are stored in the trace buffer.

Current PC with 8K Step log:
Program is kept running starting from the current program counter until a breakpoint is matched. Only the last 8K steps of execution history are stored in the trace buffer.

Initial with 4K-4K log:
Program is kept running starting from the initial address until a breakpoint is matched. Only the last 8K steps of execution history (4K steps before and 4K steps after the matched breakpoint) are stored in the trace buffer.

Current PC with 4K-4K log:
Program is kept running starting from the current program counter until a breakpoint is matched. Only the last 8K steps of execution history (4K steps before and 4K steps after the matched breakpoint) are stored in the trace buffer.

2.1.5.2 "Address Breakpoint" Dialog Function Description

The breakpoint set up method in this dialog is to use an address breakpoint that contradicts with the source level breakpoint; i.e., the source level breakpoint will be ineffective if an address breakpoint is set at the same time. Therefore, if you select the "Address breakpoint not active" option, the source level breakpoint will become effective.

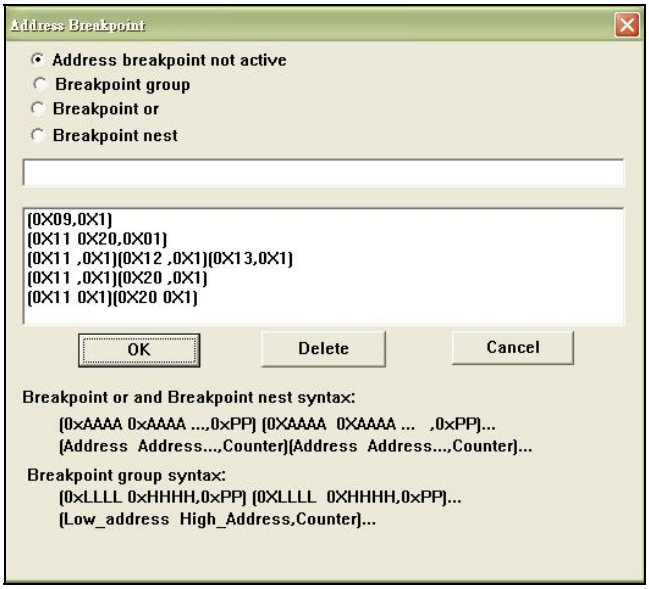


Figure 2-10 ***Address Breakpoint*** *Dialog from* ***Address Break Point*** *Command*

There are three types of address breakpoints: **Group**, **OR**, and **Nest**, which are contradictory to each other. However, the setup syntax for **OR** and **Nest** is the same, but different from **Group**.

Breakpoint Group: 63 groups are available at most and each group has Start Address, End Address, and Pass Count. When any instruction is executed between Start Address and End Address, the Pass Count is deducted by "1." When the Pass Count is equal to "0," the executing program is stopped at once. Each of the 63 groups is independent from each other.

Breakpoint OR: 63 groups are available at most and each group is composed of several addresses and a Pass Count. When an address is executed within a group, Pass Count is deducted by "1." When the Pass Count is equal to "0," a breakpoint occurs. Each of the 63 groups is independent from each other.

Breakpoint Nest: Assign some address location as groups and specify those groups as a Breakpoint Nest. The outer (the later specified) group must be satisfied first before the inner group can take effect.

The Breakpoint Nest and breakpoints of program line are contradictory. In other words, if Breakpoint Nest is active, then the program line breakpoints become inactive, and vise-versa.

- **Breakpoint Nest Setup**

1) With reference to *Figure 2-10* above, enable the **Breakpoint nest** checkbox.
2) Input breakpoint address. If the address breakpoint is for *(0x10 0x20 0x30, 0x55) (0x15 0x100 0x170, 0x10)*, address location *0x10, 0x20, & 0x30* are assigned to the same group (Group 1). A breakpoint is attached to this group, and the associated Pass Count of this breakpoint is 0x55.

   Address locations *0x15, 0x100, & 0x170* are assigned to another group (Group 2). A breakpoint is attached to this group, and the associated Pass Count of this breakpoint is *0x10*.
3) Click **OK** button.
4) Execute **Go** (F5) command from **Debug** menu.

5) If the Group 2 must be satisfied (Pass Count decrements to "0"), then the Group 1 will take effect (decrease its Pass Count on meeting the breakpoint condition). As soon as all the groups are satisfied, the execution is stopped at the breakpoint.

■ Using "Address Break Point" in Defining a More Complex Breakpoints

The **Address Break Point** command allows you to directly set a more complex breakpoints setup which can terminate at an address location.

1) Open the **Address Breakpoint** dialog (**Debug → Address Break Point**) or apply shortcut keys **ALT** + **A**.

2) The **Address Breakpoint** dialog displays as shown in the following figure.

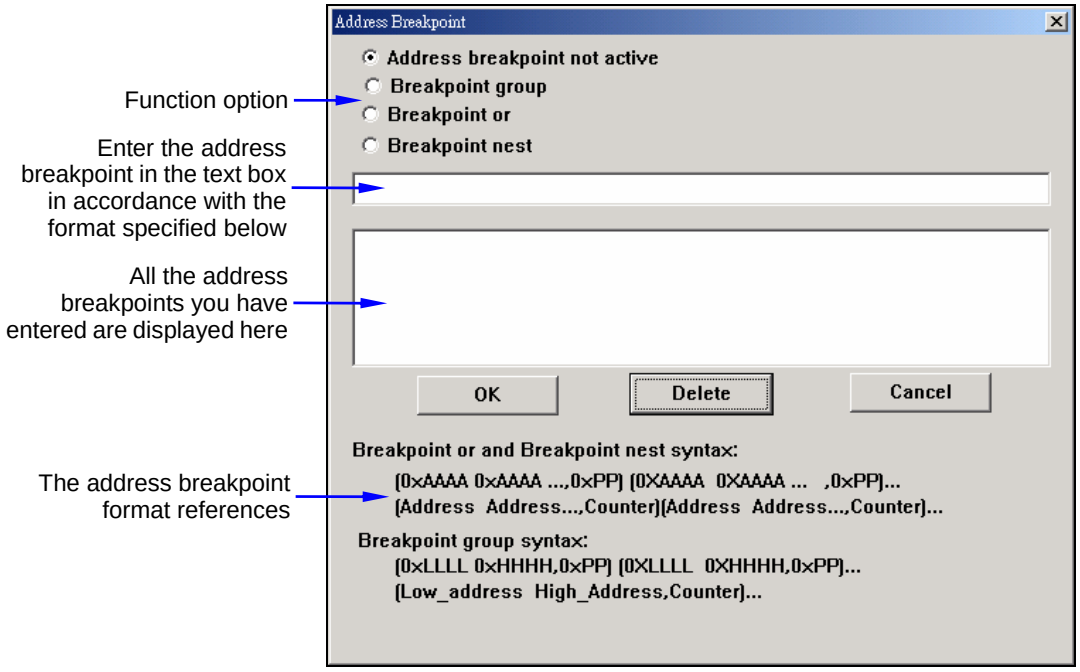


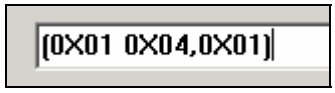
Figure 2-11a ***Address Breakpoint*** *Dialog with Descriptions of their Components*

3) Select an address breakpoint option (**Breakpoint group**, **Breakpoint or**, or **Breakpoint nest**) as follows**:**

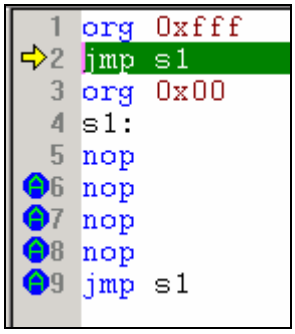
- **Breakpoint group -** Select this option to set a group of breakpoints within a range of address locations with a counter number.

a) Select the **Breakpoint group** option.

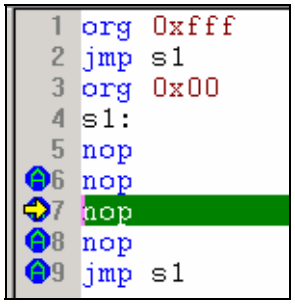
b) Specify a breakpoint group consisting of a start address, end address, and a breakpoint counter in the text box. A breakpoint will be set to all addresses within the specified start and end address range. The figure example below shows "*(0X01 0X04, 0X01),*" which means breakpoints are set to all addresses between "*0X01*" and "*0x04*" and break counter is set at "**1**," thereby allowing a termination break on one passing.



c) Click **OK** button to save the settings. You should see the group breakpoints indicated in the **Editor** window or as shown in the example figure at right.



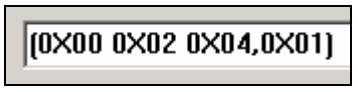
d) Click **Go** command (**Debug → Go**) to see the steps and result of the code execution as shown in the figure at right.



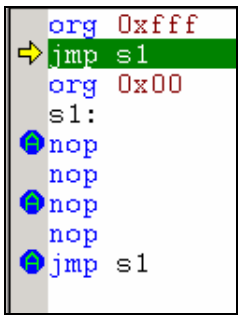
● **Breakpoint or -** Select this option to set breakpoints at one or more address locations with a counter number.

a) Select the **Breakpoint or** option.

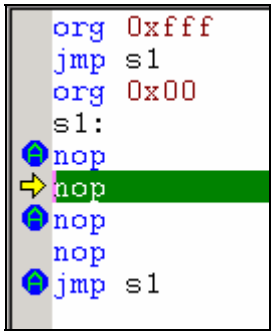
b) Enter the breakpoint address(es) and counter in the format- "***breakpointAddress1 breakpointAddress2 …, counter***". This means that a breakpoint will be set at locations *0x01*, *0x02*, and *0x04* with the counter set to "**1**".



c) Click **OK** button to save the settings. You should see the group breakpoints indicated in the **Editor** window or as shown in the example figure at right.



d) Click **Go** command (**Debug → Go**) to see the steps and result of the code execution as shown in the figure at right.

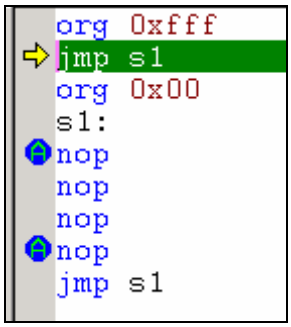


● **Breakpoint nest** Select this option to set nested breakpoints consisting of groups of address locations with each group assigned with a counter number.

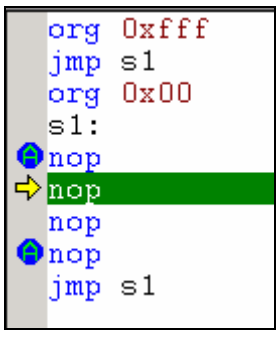
a) Select the **Breakpoint nest** option.

b) Enter the breakpoint address(es) and a counter in the format of – "***breakpointAddress1 breakpointAddress2 ..., counter***". The figure at right shows an example with "*(0X00, 0X01) (0X03, 0X02).*" This means that the program will run address *0x03* twice, run address *0x00* only once and then terminate.

c) Click **OK** button to save the settings. You should see the breakpoints indicated in the **Editor** window or as shown in the figure at right.



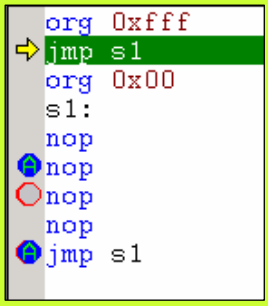
d) Click **Go** command (**Debug → Go**) to see the steps and result of the code execution as shown in the figure at right..



NOTES

1. *You can only set breakpoints after performing dump action. If you try to set breakpoint without dumping, a pop-up message will display to prompt you to execute dump action.*

2. *Breakpoints (solid blue dot) created through the **Address Break Point** command (as explained above) are independent from the breakpoints (solid red dot) created individually by double-clicking on the pointed address (or by pressing F9). Since they do not conflict with each other, the blue and red address breakpoints can co-exist at the same time and at the same address location. However, the blue breakpoints has the priority over the red ones when **Go** command is executed. Hence, any existing solid red dot breakpoints are ignored*



3. *As the **Address Breakpoints** dialog created blue breakpoints has the priority over the individually-created red breakpoints, the solid red dots turns into hollow red circle when both types of breakpoints occupy the same address. Once the blue address breakpoints are cleared, the hollow red circle becomes solid again. The red address breakpoints are applicable only when the blue ones are all cleared.*

4. *To clear all blue breakpoints, do either of the following:*
   - *From the menu, click **Debug → Clear All Address Break Point***
   - *Press the shortcut keys **CTRL + ATL + A***

5. *To clear individual or a range of blue breakpoints, do the following:*
   *a) From the menu, click **Debug → Address Break Point** (or press **Alt+A**)*
   *b) From the resulting **Address Breakpoint** dialog (figure below), which displays all the existing blue breakpoints, select the specific or range of breakpoint(s) to delete from the dialog.*
   *c) Click **Delete** button to remove the selected breakpoint.*

You can also enter the specific or range of address breakpoint(s) you want to delete here

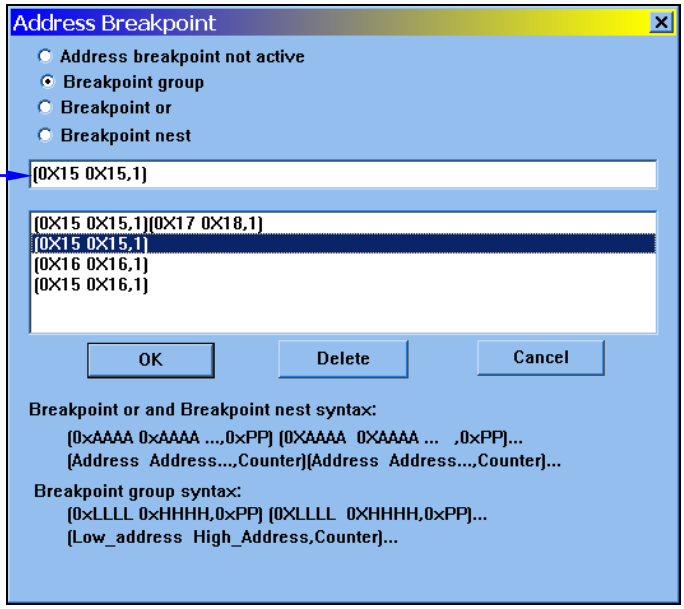


*Figure 2-11b **Address Breakpoint** Dialog with Selected Individual or a Range of Blue Breakpoints to be Cleared*

2.1.6 Tool Menu

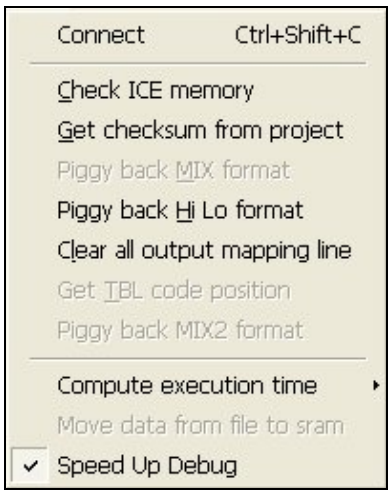


*Figure 2-12 **Tool** Menu*

Connect Set proper connection between target IC & connecting port

Check ICE Memory Check available memory from ICE

Get Checksum from Project Obtain checksum from the compiled program

Piggy back MIX format Create a MIX file to write piggybacked into EEPROM

Piggy back Hi Lo format Create a couple of files with extension names "Hi" and "Lo," which are used to write piggybacked into EEPROM

Clear all output mapping line Clear program mapping lines from output window

Get TBL code position Obtain TBL code position

Piggy back MIX2 format Applicable to EM78813 & EM78815 only.

If an original 8bit is used to piggyback and fit into 16bit, the piggybacked high byte in 8bit will become low byte in piggybacked 16bit.

Compute Execution Time Calculate the execution time between two breakpoints. See next section below.

Move data from file to sram Applicable only to EM78815. See details in the sub-section below (Section 2.1.6.2)

Speed Up Debug For C Compiler use only

2.1.6.1 Computing Execution Time

Follow the steps below in calculating execution instruction time of ICE**:**

1) Click **Compute Execution Time** command, then the following dialog displays. Enter the required ICE frequency and instruction period and click **OK** button.

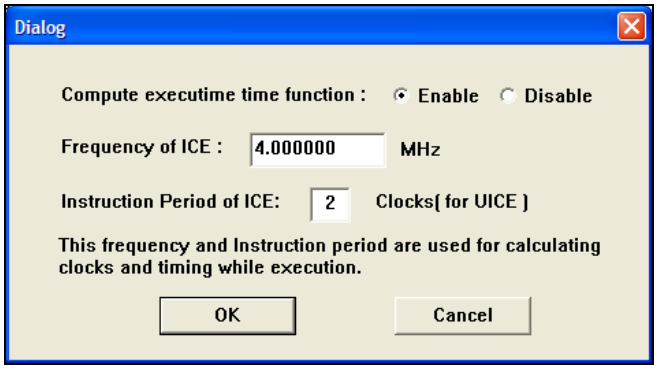


*Figure 2-13a **Compute Execution Time** Setup Dialog*

2) Execute the **Project → Trace Log** command (or F2). The trace buffer info is then displayed in the **Message** tab sub-window of **Output** window.

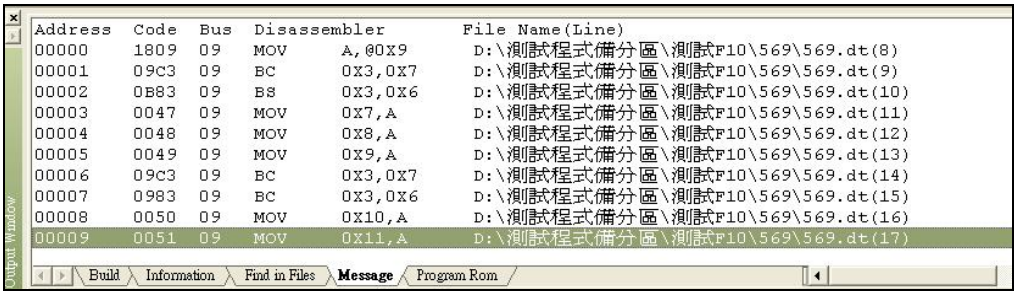


Figure 2-13b ***Compute Execution Time*** *Trace Buffer Display from* ***Message*** *Tab Sub-Window of* ***Output*** *Window*

3) Then look for the execution time result from **Information** tab sub-window of **Output** window.

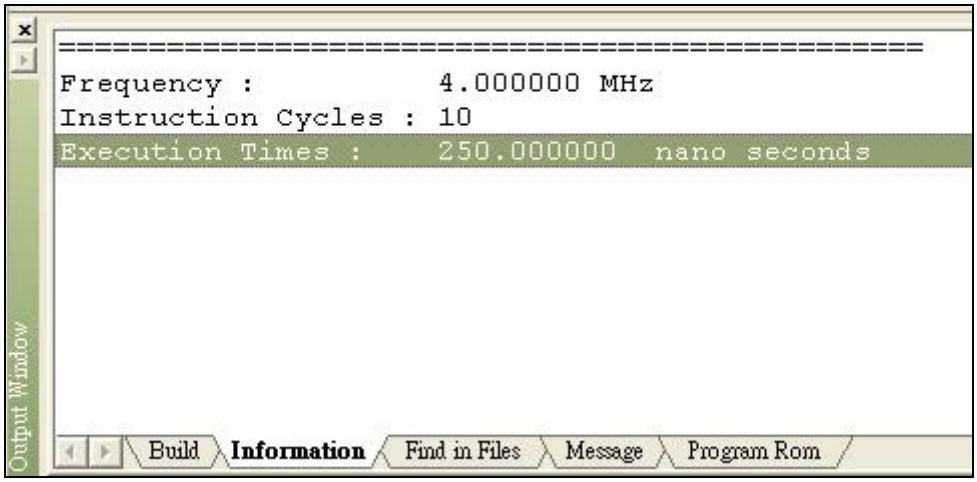


Figure 2-13c Execution Time Result Displayed from ***Information*** *Tab Sub-Window of* ***Output*** *Window*

2.1.6.2 Moving Data from File to SRAM (Applicable to EM78815 only)

The eUIDE supports moving data to external 512K byte SRAM board which allows the use external SRAM as a replacement for the slower flash. With this new feature, you can move data from binary file to external SRAM and use external SRAM to develop program with program size of over 64K words.

With your hardware properly connected and ready to debug your program, access the data from external memory. Now, replace external flash ram with external SRAM. Then **move of data from binary file to external SRAM** and **dump code to external SRAM** as further discussed in the following pages.

■ Move Data from Binary File to External SRAM

1) Execute **Debug → Reset (F6).**

> **IMPORTANT!**
> *You need to initialize the ICE environment and eUIDE by executing reset (F6) before moving binary data into external SRAM. Otherwise, data transfer to SRAM will fail. However, if you have just performed a reset after connecting the ICE or is already in the process of debugging the program, you may skip* ***Reset****.*

2) Click **Tool → Move data from file to SRAM** command.

3) When the resulting dialog (right figure) appears, browse for the **Binary file name** you want to operate and enter the **Data Begin Address** and **Length** into their respective text boxes.

   Be sure to read the notes at the bottom of dialog, and then press **OK** button to move your data. You may enable the **Checking after dumping** checkbox to verify data after finishing the process. When a byte in error is encountered, verification process is aborted.

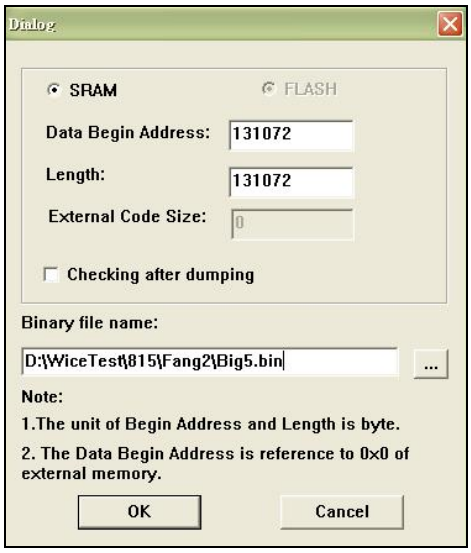


*Figure 2-14 "**Move data from file to SRAM"** Dialog*

The **External-Code-Size** box shows how many bytes your program code is over 64K words. So allow your **Data Begin Address** to be larger than **External-Code-Size**.

The maximum **Data Begin Address** is 524,288 (512K byte) and the **Length** is 262,144(256K byte). So if the data you want to move from binary file to external SRAM is over 262,144 byte (256K byte), you have to partition your binary file into two files and move your data twice.

Example:

Assuming you have a 512K byte external SRAM, a binary file with filename "*Big5.bin*," and its size is 393,216 byte (384K byte), and want to move all the contents of "*Big5.bin*" into external SRAM from the **Data Begin Address** 131,072. The file, being over 256K byte, is partitioned into two files; e.g., "*Big51.bin*" and "*Big52.bin*" with sizes of 262,144 byte (256K byte) byte and 131,072 byte (128K byte) respectively.

First, move "*Big51.bin*" data into external SRAM by setting **Data Begin Address** at 131,072, **Length** at 262,144, and the "*Big51.bin*" folder as **Binary file name**. The moved data will be located from address 131,072 to address 393,215.

When "*Big51.bin*" data transfer is completed, then process to move "*Big52.bin*" into external SRAM by setting **Data Begin Address** as 393,216, **Length** at 131,072, and the "*Big51.bin*" folder as **Binary file name**.. The moved data will be located from address 393,072 to address 524,287.

It is highly recommended to complete moving partitioned data in one successive process.

■ **Dump Code to External SRAM:**

If your program code is over 64K words, click **Project → Dump to ICE** (F3) to dump program code to ICE. eUIDE will process and check code that is under 64K words. If your program code is less than 64K words, eUIDE will not process dumping to external SRAM as discussed in Section 2.1.7.4.

If your code is over 64K words, eUIDE will dump the code into external SRAM. It usually takes about 45 seconds to completely dump the code of 64K words. Actual dumping time is dependent on the code size over and above the 64K words. You can enable the **External Code-checking** checkbox in **Environment setting** Dialog (see Section 2.1.7.4) to check dumping process and result.

It is highly recommended to enable the **External Code-checking** option during the first dumping and disable it when repeating to dump the same file to save another about 40 seconds.

2.1.7 Option Menu

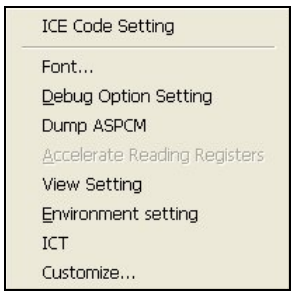


*Figure 2-15 **Option** Menu*

ICE Code Setting Set code option for the selected microcontroller

Font… Define font for **Editor** windows (fonts for other windows are fixed)

Debug Option Setting Set debugger variables options in the dialog shown in *Figure 1-16* below

Dump ASPCM Dump to Data ROM

Accelerate Reading Registers Applies only to USB ICE. Read registers quickly when frequency is over 2MHz as explained in Section 2.1.7.2 below

View Setting Set the eUIDE window view variables as explained in Section 2.1.7.3 below

Environment setting Set eUIDE environment variable, e.g., whether list/map file is to be created or not, and the number of **Editor** windows to display as explained in Section 2.1.7.4 below

ICT Execute ICE test

Customize… Customize toolbars, menus, and accelerators. See Section 2.1.7.5 below for details.

2.1.7.1 Debug Option Setting

Set the debugger variables options with the following dialog. The options are divided into four blocks as illustrated below.

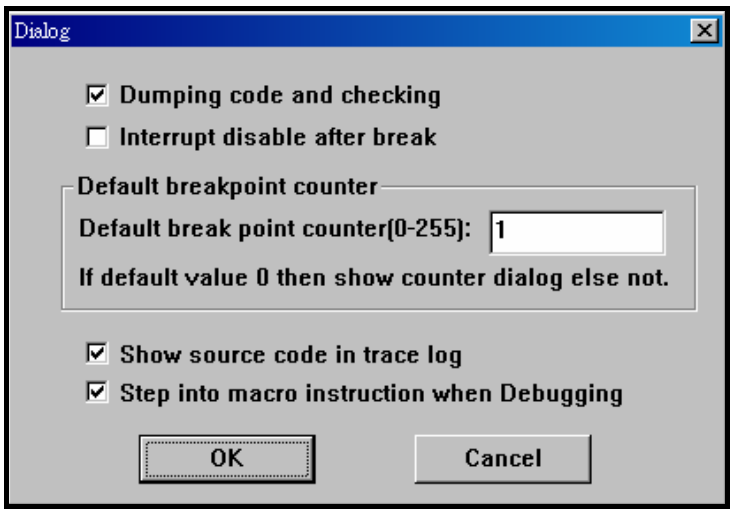


*Figure 2-16 **Debug Option Setting** Command Dialog*

a) **Dumping codes and checking**: eUIDE will check the hardware memory before dumping codes.

b) **Interrupt disabled after break**: Interrupt is disabled when a breakpoint is encountered. It is used to avoid any interrupt from occurring when screen is updating as TCC2, COUNTER1, and COUNTER2 will keep on working after program is stopped. Therefore, the disabled interrupt must remain active; otherwise users cannot debug the program.

c) **Default breakpoint counter**: See Section 3.9 of Chapter 5, *Debugging a Project* for details.

d) **Show source code in trace log**: In the **Output** window, the trace log is set by default to display the disassembler contents only. If this function is enabled, the address source level breakpoint generated by the trace log address will also appear in the **Output** window.

e) **Step into macro instruction when Debugging:** When this option is enabled, click **Debug → StepInto** (F7) to run the program instructions including the macro step-by-step. To run the macro code in the background, click StepOver (F8). As a new feature, you can now disable this checkbox to run the macro in the background, regardless of whether you click StepInto (F7) or StepOver (F8). Please remember that the instruction line that is currently being executed will stop at the macro address location (shaded in green). This location will point to the first line of the macro code. Click StepInto (F7) to execute the first macro instruction and then its stop at the next macro instruction (shaded in green again).

Example:

```
1.  test macro
2.  nop
3.  nop
4.  endm
5.  org 0x0
6.  jmp main
7.  main:
8.  nop
9.  test
10. nop
11. jmp $
```

Per above code example, if the program execution stops at Line 8 (shaded in green on screen), click StepInto (F7) to execute Line 8 and then execution stops at Line 9. Note that in the previous eUIDE version, it stops at Line 2 of the macro.

As Line 9 is actually linked to Line 2 address, clicking StepInto (F7) will execute the code at Line 2 and stops at Line 3. If you click StepOver (F8), the program executes the complete macro instructions (Line 2 & Line 3) and stops at Line 10.

2.1.7.2 Accelerate Reading Registers

Figure 2-17 ***Accelerate Reading Registers*** *Command Dialog*

When you use USB ICE and the frequency is over 2 MHz, you can enable this capability. But it is not 100% stable. If you see some abnormal appearances, try to disable this option.

NOTE

If you select IRC when you set the code option, eUIDE may enable this option automatically according to the chosen frequency.

2.1.7.3 View Setting

Set the eUIDE window view variables or properties.

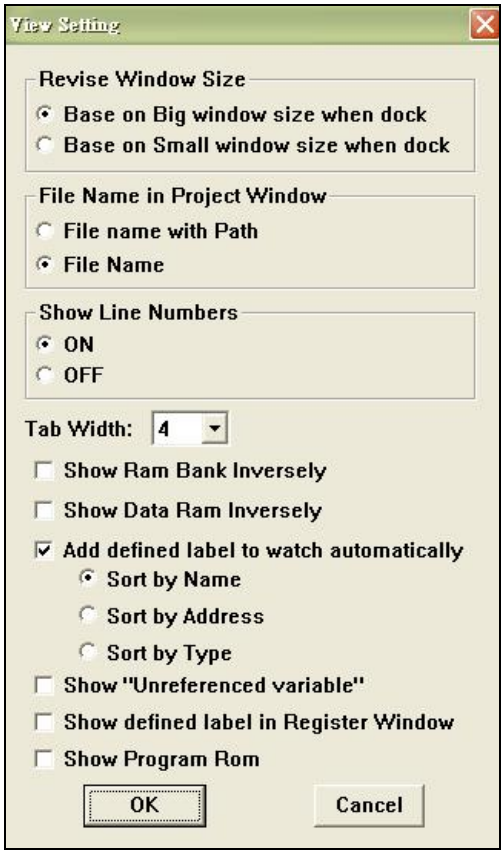


Figure 2-18a ***View Setting*** *Command Dialog*

a) **File Name in Project Window:**

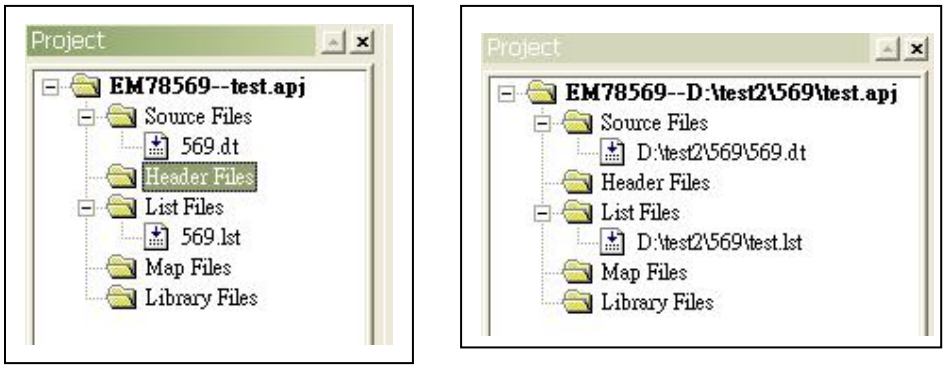


Figure 2-18b ***View Setting*** *Showing Differences between "File Name" (Left) vs. "File name with Path" (Right) Options*

b) **Show Line Numbers:**

Figure 2-18c ***View Setting*** *Showing Differences between "Column On" (Left) vs. "Column Off" (Right) Options*

c) **Tab Width:** customize tab size.

d) **Add defined label to watch automatically:** (Enable after Dump)
When checkbox is enabled, definite variables are automatically appended with *rpage/rbank/iopage* when displayed in **Watch** Window.

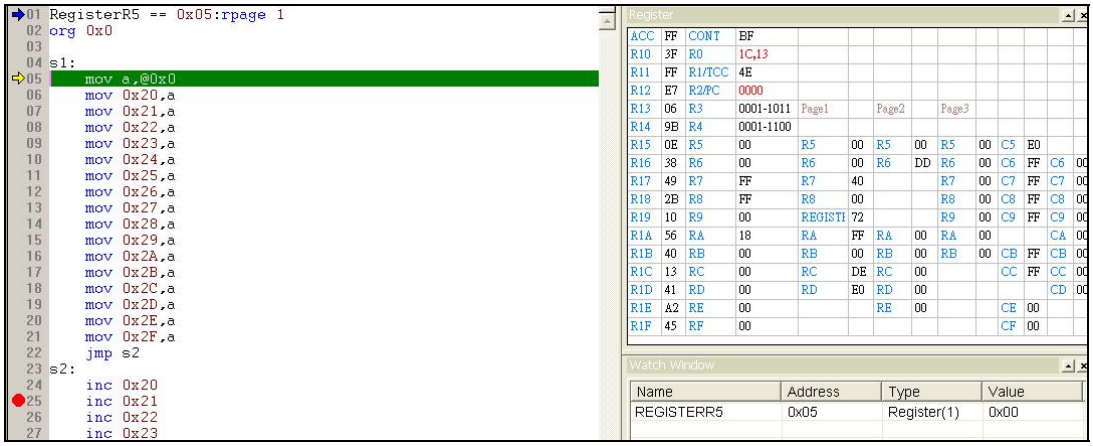


Figure 2-18d ***View Setting*** *Showing the Appended Definite Variable "rpage" Displayed in the* ***Watch*** *Window*

e) **Show "Unreferenced variable":** when enabled, the build assemble code will check and determine whether the variables were utilized or not.

f) **Show defined label in Register Window**: (Enable after Dump)

When enabled, Register name are displayed as label name in the **Register** Window. If the label name length is over 6 characters, Register name will display the first six characters, and tooltip displays complete name.

When disabled, the Register name will appear as initial name in the **Register** Window.

g) **Show program Rom:** show the program code in the **Program Rom** sub-window of **Output** window.

2.1.7.4 Environment Setting

Set the eUIDE environment variables e.g., whether LIST/ MAP file is to be created or not, and the number of **Editor** windows to display.

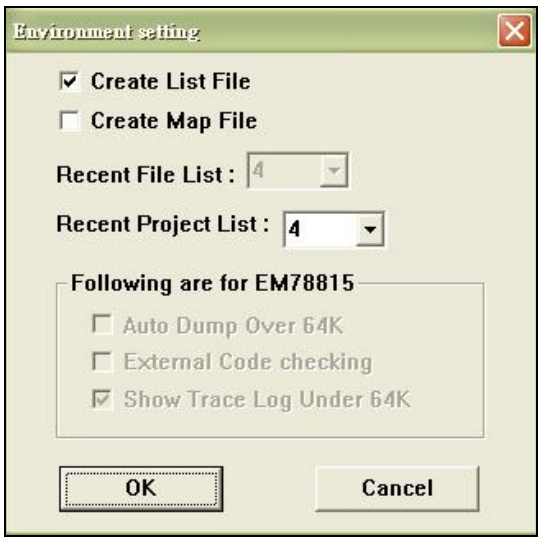


*Figure 2-19 **Environment Setting** Command Dialog*

a) **Create List file:** If selected, the LIST file is created after the related project is assembled. The LIST file will include line number, address, program code, and source file.

b) **Create MAP file:** If enabled, the related LIST file is created after linking to a project. The MAP file will include public symbol name and address.

c) **Recent File List**: The number of recently closed filenames to be saved in the ICE. The maximum number of sub-editor windows that can be accommodated is 10.

d) **Recent Project List**: The number of recently closed project names to be saved in the ICE. The maximum number of sub-editor windows that can be accommodated is 10.

e) **Auto Dump Over 64K**: For EM78815 only. If checkbox is enabled, all program codes are dumped into the hardware.

f) **External Code checking**: For EM78815 only. If selected, the process of dumping program (or data) to external SRAM is monitored and checked.

g) **Show Trace Log Under 64K**: For EM78815 only. If enabled, eUIDE will check the 64K program code when **Go**, **Free Run**, or **Go To Cursor** command from **Debug** menu is executed. The maximum length of trace log is 8K words.

2.1.7.5 Customize…

The **Customize** dialog displays with four tabs as shown at right. The following describes the function of each tab**:**

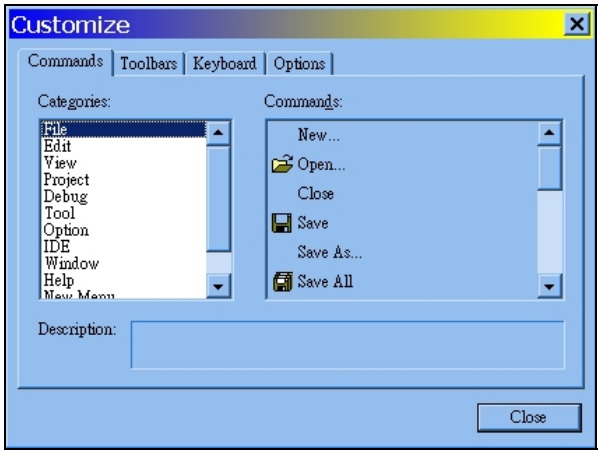


Figure 2-20a ***Customize*** *Command Dialog Showing "**Command**" Tab*

a) **Commands tab:**
Select this tab to display all available commands under a selected category. Then drag and drop a command into toolbars, menu bar, or into a drop-down command menu (from menu bar). To restore default settings, go to **Toolbar** tab and click **Reset All**.

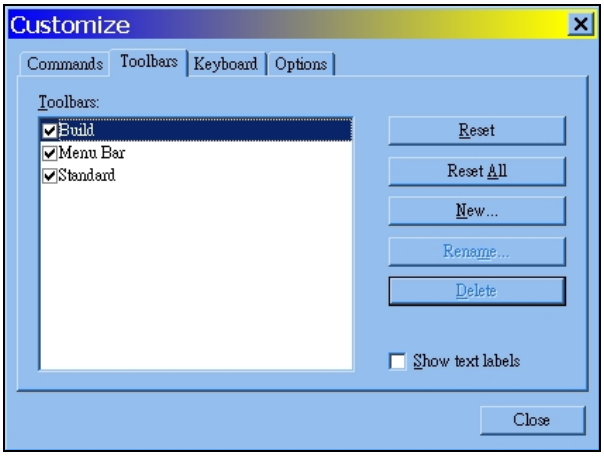


Figure 2-20b ***Customize*** *Command Dialog Showing "**Toolbar**" Tab*

b) **Toolbars tab:**
The **Toolbars** tab allows you to enable/disable the **Build** and **Standard** toolbars but not the **Menu Bar**. You can however, **Reset/Reset All** all toolbars to restore the default settings of a particular or all toolbars. Furthermore, you can create your own new unique toolbar which you can later **Rename** or **Delete** from this tab. You can also add/remove text labels to the toolbar buttons by clicking on the **Show text labels** check box.

c) **Keyboard tab:**
The **Keyboard** tab allows you to create/remove the shortcut keys for the commands of a selected command category. The procedure is explained below:

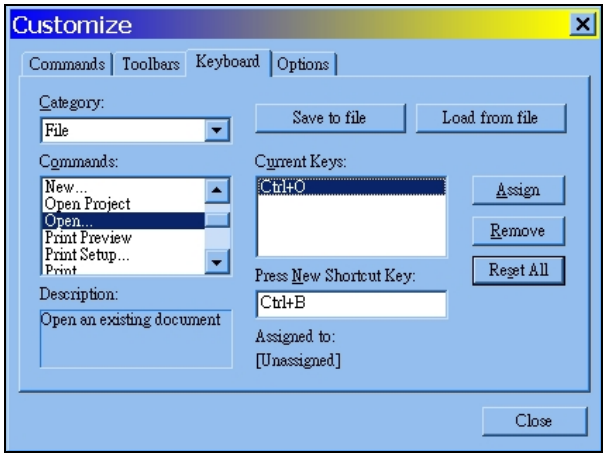


Figure 2-20c ***Customize*** *Command Dialog Showing "****Keyboard****" Tab*

■ Creating a shortcut key:

After selecting a **Category** and **Commands** option from their respective boxes, enter your custom shortcut key into the **Press New Shortcut Key** box. eUIDE will auto-detect and determine whether the new shortcut key has already been assigned or not. If it has already been assigned, the pertinent command name (with which the shortcut key is currently assigned) will display under **Assigned to:** field, and you need to directly enter another shortcut key. Otherwise, *[Unassigned]* will display. Then click **Assign** button to apply.

■ Removing an existing shortcut key:

After selecting a **Category** and **Commands** option from their respective boxes, the corresponding command shortcut key (if available) will appear in the **Current Keys** box. Select the shortcut key you want to remove and click **Remove** button to delete.

■ Restoring all shortcut keys to their default settings:

Click **Reset All** button to reset all command shortcut keys back to the eUIDE default settings.

■ Saving/Loading settings:

To save your custom shortcut keys for future use, e.g., when upgrading eUIDE to a new version, click **Save to File** button to store the settings to a file (with a "*.KEY*" file extension). After installing a new version of eUIDE, you can simply click **Load from file** button to re-apply your custom shortcut keys into the new eUIDE.

■ *Option tab*

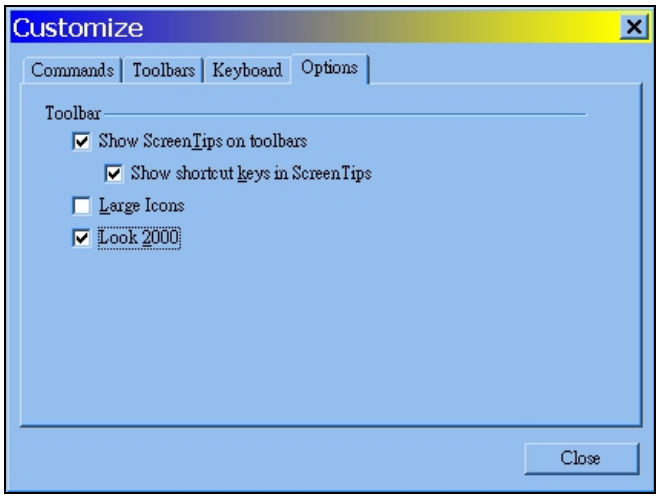


Figure 2-20d ***Customize*** *Command Dialog Showing "****Option****" Tab*

Use the **Option** tab to set the size of the toolbar buttons and specify whether to display screen tips and shortcut keys (where applicable) when pointing at the button. You can also set to display the eUIDE sub-windows to look like that of Windows 2000.

2.1.8 Window Menu

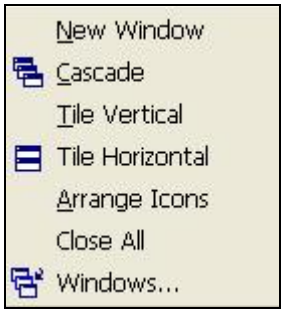


Figure 2-21 ***Window*** *Menu*

New Window Open a new (or split) **Editor** window

Cascade Rearrange all **Editor** window active files so that they will appear overlapping in sequence with their respective title bar fully visible

Tile Vertical Rearrange all opened **Editor** windows vertically

Tile Horizontal Rearrange all opened **Editor** windows horizontally

Arrange Icons Arrange all opened file filenames in a single line formation (minimized into multiple file icons) at the bottom of the **Editor** window.

> **NOTE**
> *This command is effective only after clicking the Minimized button ( ) at the right end of the eUIDE window Menu bar.*

Close All Close all opened files

Windows… Show Select Window Dialog

2.1.9 Help Menu

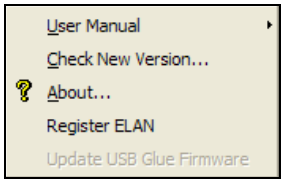


*Figure 2-22a **Help** Menu*

User Manual Open the eUIDE User's Manual

Check New Version… Check new version of eUIDE from ELAN

About… Shows the current version of eUIDE program and other information including a "read me" file on recent changes of the eUIDE

About… Shows the current version of eUIDE program and other information including a "*ReadMe*" file on recent changes of the eUIDE

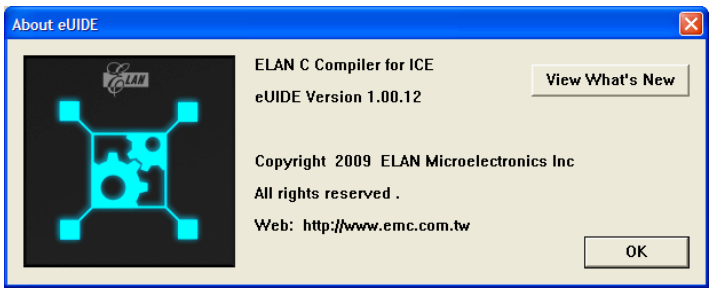


*Figure 2-22b **About** Command Dialog*

Register ELAN On-line registration with ELAN

Update USB Glue Firmware Update USB UICE firmware if necessary

Chapter 3
Getting Started

3.1 Overview

3.1.1 System Requirements

The EM78 Series eUIDE requires a host that meets the following specifications**:**

- IBM PC (Pentium 100 or higher is recommended) or compatible computers
- Win2000, WinME, NT, or WinXP and Vista-32
- At least 40 MB **(**or more) free hard disk space
- At least 256MB of RAM. 512MB or more is recommended
- Mouse and USB connectors are highly recommended

3.1.2 Software Installation

NOTE

- *Please note that eUIDE can only be installed in the predefined directory **C:\EMC\eUIDE**. This restriction is necessary to prevent assigning an installation path that may contain space character that could cause serious error during compilation.*
- ***The file paths (*.apj, *.cpj, *.dt, *.c, *.h, and *.inc) must NOT contain any space.** Any space in the path will cause error during compilation.*

The eUIDE compiler and ICE driver are is included in eUIDE program package. When installing the eUIDE, the compiler is also automatically installed.

If this is your first time to install the eUIDE program, you need to reboot your computer after eUIDE installation is completed because of the printer port driver (DLPortIO). If you use USB ICE under Windows 9X, you also need to reboot.

If it is not the first time to install eUIDE, the setup program will uninstall the previous installation then install the new one.

During installing, users cannot change the default install path.

When the operating system is searching the USB ICE hardware, please make sure the power of ICE is set to ON. You can see the EMC USB ICE through the OS Device Manager if the driver is installed and the ICE is connected correctly.

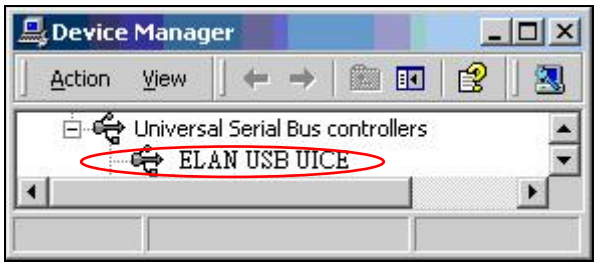


Figure 3-1 OS Device Manager Confirming ICE Installation

3.1.3 ANSI Compatibility

Compliance with the ANSI standard is limited to free-standing C to accommodate the unique design characteristics of the EM78 Series microcontrollers.

3.2 Hardware Power-up

With the ICE properly connected to target board, PC, and power source, switch on ICE power and observe its red power LED lights up. If the target board derives its power from ICE, the yellow LED lights up as well.

Then launch your eUIDE IDE software when ICE and target board power-up is confirmed to function normally.

3.3 Starting the eUIDE Program

To start eUIDE Program, click on the eUIDE icon from desktop or from Windows Start menu. When starting from the Start menu, click Programs, then look for eUIDE group and click on eUIDE icon.

3.3.1 Connect Dialog

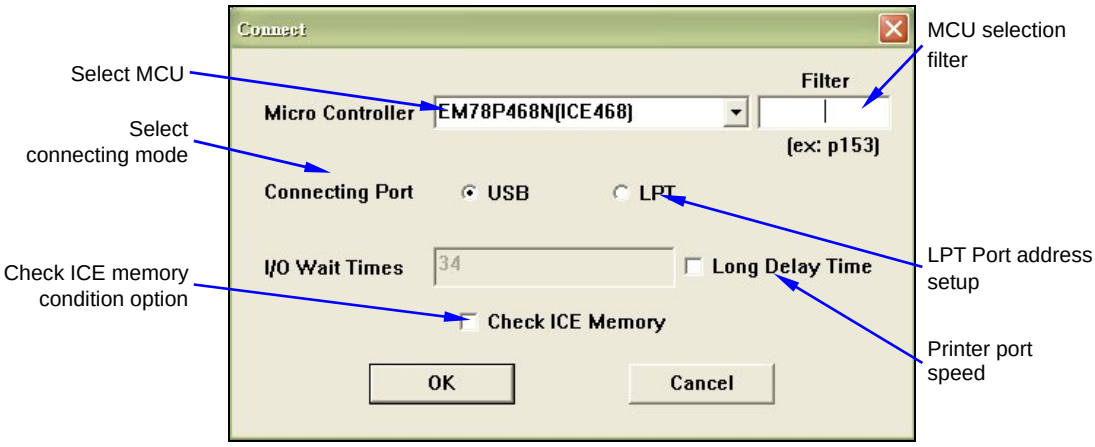


*Figure 3-2 eUIDE Program **Connect** Dialog*

Once the program is started, the main window of the program will initially display the **Connect** dialog (figure above) to prompt you to set the proper connection between your existing target microcontroller and connecting port.

Where:

Filter: Key-in the last 3-digits of the desired target IC, for example; "159." The **Micro Controller** combo box will display all EM78 series ICs having "159" as its last part number digits, thus, speeding up the search for your target IC.

Micro Controller: The IC part number shown on the combo box is one of the EM78 series. Select your target IC by clicking on the combo box arrow.

Connecting Port: Select the proper connecting port, LPT or USB for your ICE. If USB ICE is not installed, you cannot select the USB option as the option will be disabled. However, if the USB option is disabled and you have the USB ICE connected with PC, go to your OS Device Manager and try the following steps**:**

1) Turn off the ICE

2) Disconnect the USB cable

3) Turn on the ICE

4) Reconnect the USB cable

5) Then check if the USB option is enabled

LPT Port Address: The system will automatically detect the printer port address (default is 378H) which is already connected with the hardware. After the connection is successful, eUIDE will also diagnose the hardware right away.

I/O Wait Times: It depicts the I/O response speed. Increase the value for slower speed and decrease for faster speed. Usually, the bigger the value, the better is the stability.

Check ICE Memory: You may enable this check box to check the ICE memory condition.

Long Delay Time: When you are unable to connect your computer to ICE, enable this check box. This will allow a longer handshaking time between your computer and ICE. Click **OK** button when done.

3.3.1.1 Reconnection

If a new ICE replaces the current one, it is necessary to reconnect it with PC. Click **Tool → Connect** to reconnect. The **Connect** dialog (*Figure 3-2* above) will pop-out again. It will take a longer waiting time to establish communication between the PC and hardware during reconnection under the same hardware environment.

3.3.2 Code Option Dialog

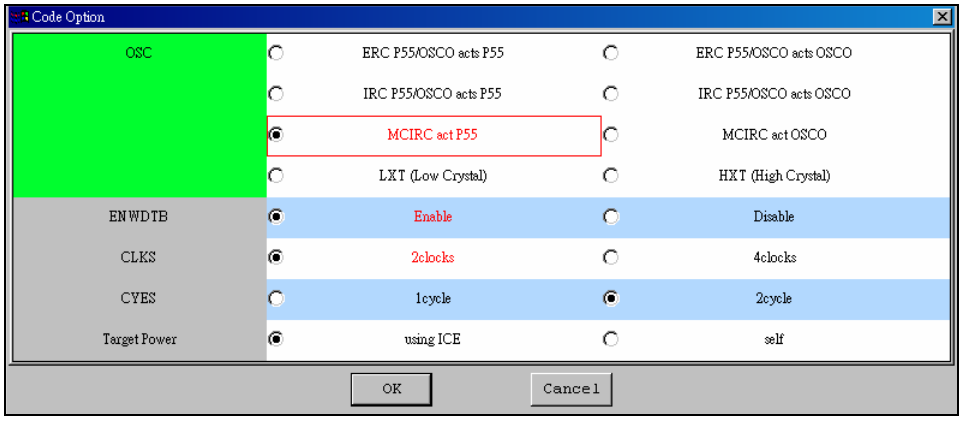


*Figure 3-3 eUIDE **Code Option** Dialog*

The **Code Option** dialog is displayed next. Check all items to confirm the actual status of the ICE and make appropriate changes as required. Then click **OK** button. You need to get acquainted with MCU or ICE specifications to able to select code option correctly. Otherwise, proper connection cannot be achieve with ICE. You can enlarge or reduce the dialog's size by dragging its edges. This dialog applies to both USB and printer port modes.

NOTE

*Not all EM78 Series MCU requires the **Code Option** setting.*

3.3.3 Accelerate Reading Registers Dialog

If you are using USB ICE, the **Accelerate Reading Registers Dialog** dialog will pop up after setting code option. Refer to Section 2.1.7.2, *Accelerate Reading Registers* for details.

3.4 Create a New Project

3.4.1 Using the Project Wizard (Project → Project Wizard)

The project wizard consists of several dialogs which will walk you through a step-by-step setting up of your project.

Step1 – Select a controller

Step2 – Select controller Code Option

Step3 – Create a new project: Set Project Name and Type

Step4 – Add a new file or an existing file to your project

Step5 – Summary

Step 1 - Select a controller

Select a controller for your project from the list, the click **Next** button.

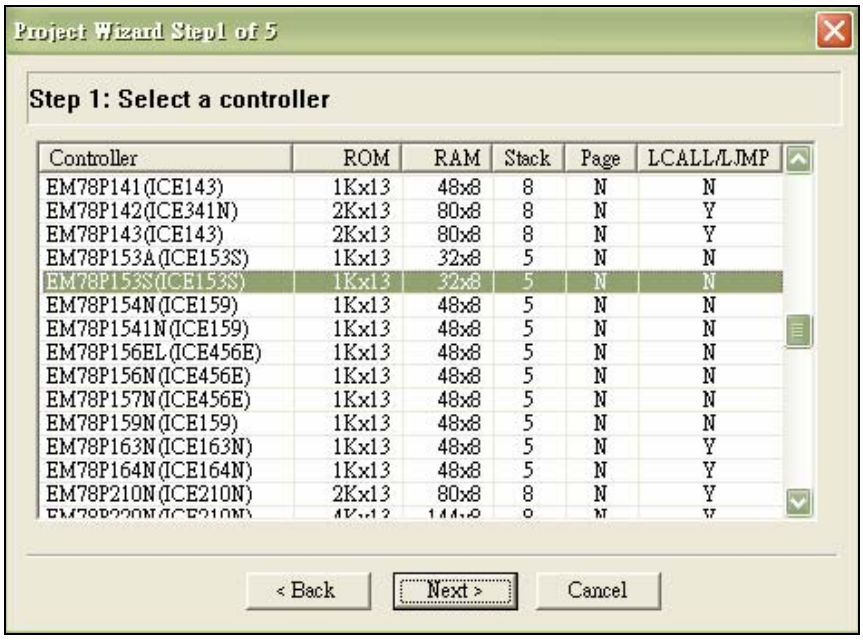


Figure 3-4a ***Project Wizard Step 1 of 5*** *Dialog*

Step 2 - Set controller Code Option

If the controller does not require Code Option setup, this dialog is skipped.

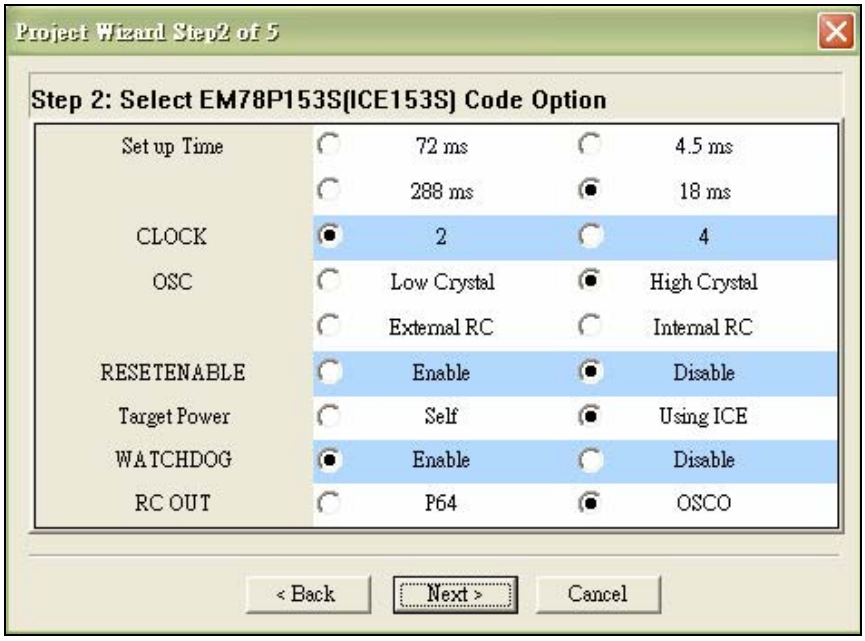


Figure 3-4b ***Project Wizard Step 2 of 5*** *Dialog*

Step 3 - Create a new project: Set Project Name and Type

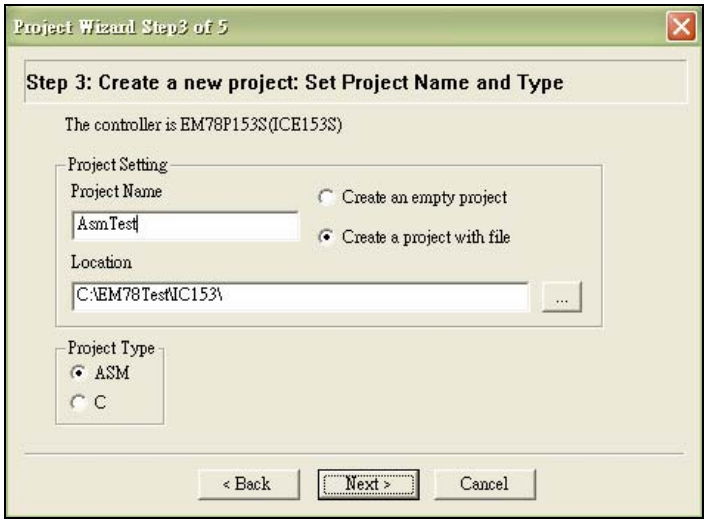


Figure 3-4c ***Project Wizard Step 3 of 5*** *Dialog*

Selecting **Create an empty project** option and clicking on **Next** button, will directly jump to and display the final go **Summary** dialog (Step 5).

Selecting **Create a project with file** option and clicking on **Next** button will go to the next step or Step 4, **Add a new file or an existing file to your project** (see below).

Enter a project name, browse or create folder to save the new project, and the project type (ASM or C) for your new project.

To set Project Type:

- Click **ASM** option button to create an Assembly language project
- Click **C** option button to create a C language project

Step 4 - Add a New File or an Existing File to Your Project

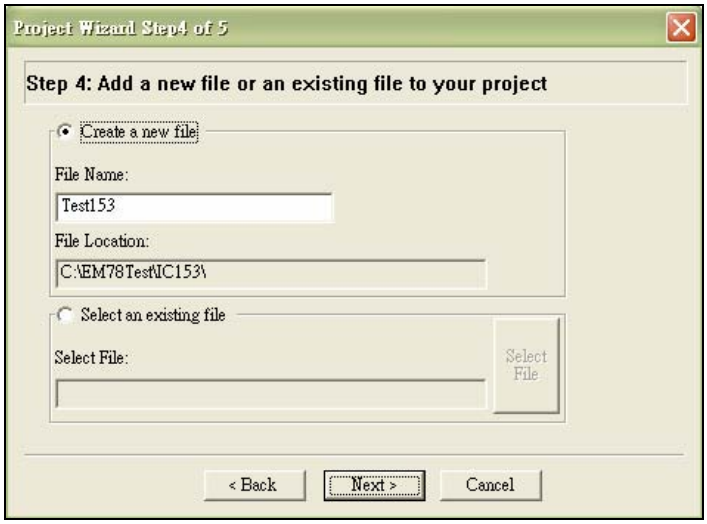


Figure 3-4d ***Project Wizard Step 4 of 5*** *Dialog*

To “Create” a new empty file**:**

Type a filename in the **File Name** text box and the created file will be saved in the folder defined in the **File Location** text box.

To Select an existing file**:**

If you already have an existing file that you would like to add to the new project, click **Select File** button. The **Insert Files into Project** dialog will be opened. The default folder is as shown in the **File Location** text box shown in the *Figure 3-4d* above.

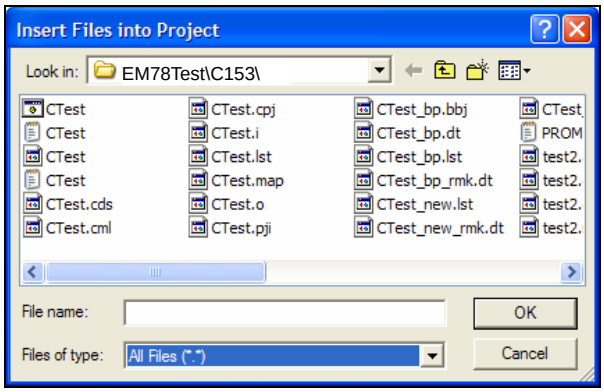


Figure 3-4e Insert Files into Project Dialog

Step 5 - Summary

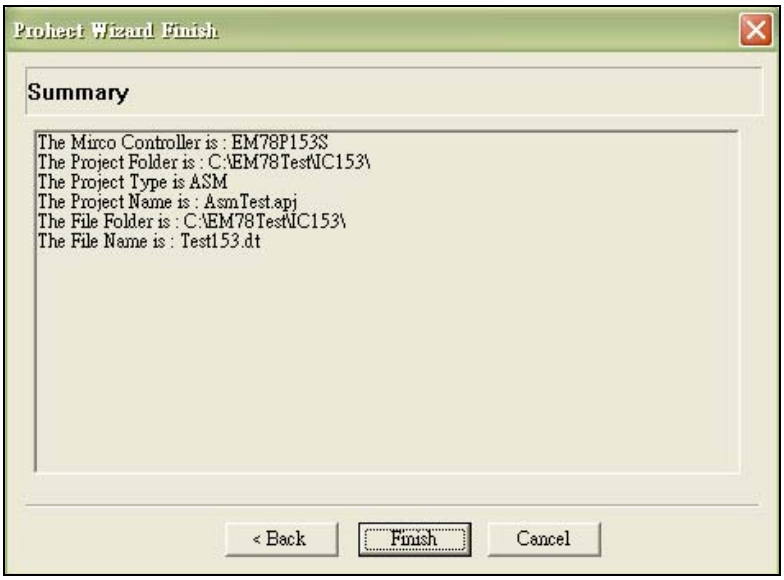


*Figure 3-4f Project Wizard **Summary** Dialog*

Double-check the summarized information on the **Summary** dialog. If further correction is needed, click **Back** button to return to the particular dialog you need to change the setting. Otherwise, click **Finish** button to complete the new project creation.

3.4.2 Using the New Command (File/Project → New…)

To create a new project with the **New…** command from the **Project** menu, follow the following steps**:**

1. Click on **File** or **Project** menu and select **New…** command from the resulting pull-down menu.
2. From the **New** dialog (shown below) that displays, click on **Projects** tab

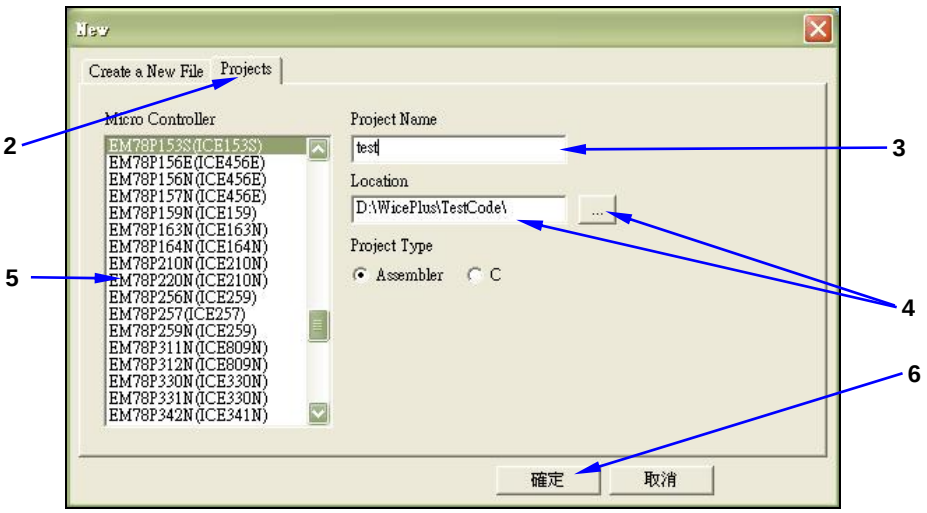


Figure 3-5a ***New*** *Dialog Showing* ***Project*** *Tab for Creating New Project*

3. Assign a name for the new project in the **Project Name** text box (suffix "***.prj***" will auto-append to the filename).
4. Locate the folder where you want to store the new project. You may use the **Browse** icon to find the appropriate folder.
5. Select the target microcontroller for your project from the **Micro Controller** list box.
6. Click **OK** button after confirming all your choices and inputs.

The new project is then created with the defined project name and micro-controller you have selected is displayed on top of the **Project** window.

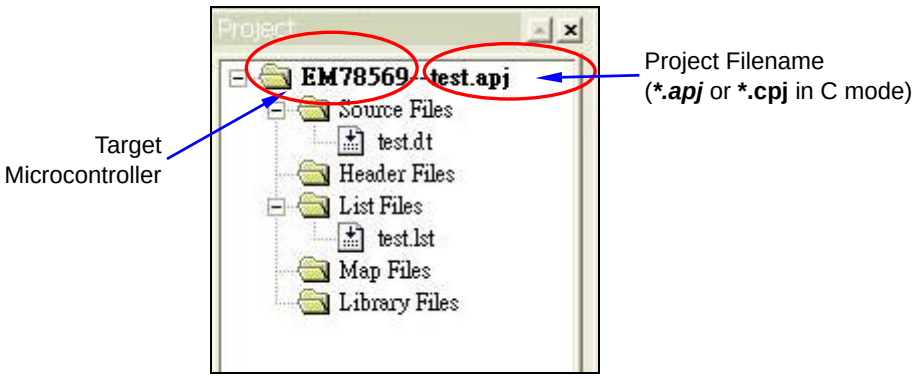


Figure 3-5b ***Project*** *Window Showing Target IC & Project Filename*

3.5 Add and Remove Source Files from/to Project

You can either insert existing source files into the new or existing project, or create new ones with eUIDE text Editor and insert them into the project.

3.5.1 Create and Add a New Source File for the Project

If your source file is yet to be created, you can take advantage of the **New** dialog (by clicking **NEW…**command from the **File** or **Project** menu) to create your new source file and use the eUIDE text editor to compose its content.

1. From the **New** dialog, click the **Create a New File** tab and select the type of source file you want to create from the **EMC Source File** list box, i.e., ****.dt*** (default or *.c for C mode) for assembly file; ****.h*** for header file.

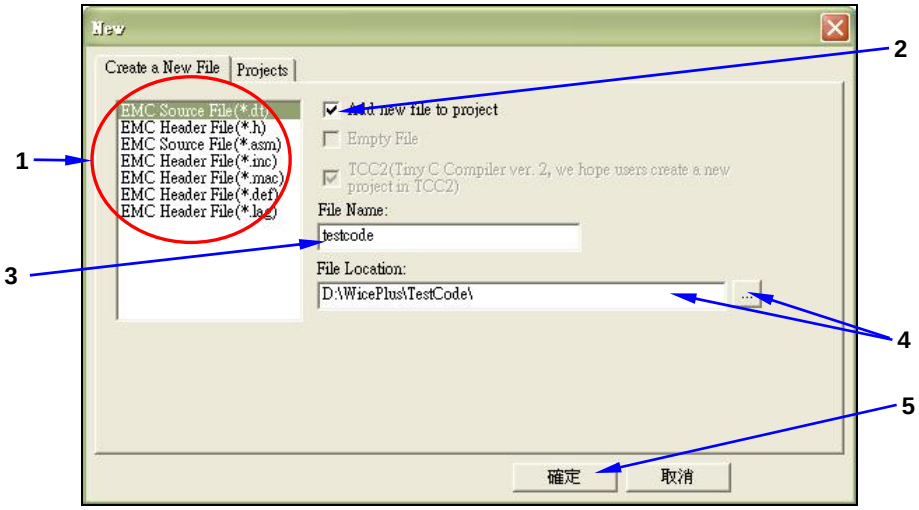


*Figure 3-6a **New** Dialog Showing Project Tab for Creating a New Source File*

2. Select **Add to Project** check box (default) if you want to automatically add the new file into your project. Otherwise clear the check box.
3. Assign a filename for the new source file in the **File Name** text box.
4. Locate the folder where you want to store the new source file in your disk. You may use the **Browse** icon to find the appropriate folder.
5. Click **OK** button after confirming your inputs. You will be prompted to start writing the newly defined source file in the **Editor** window.

NOTE

1. *Do not write code over 512 lines, especially in C mode, or serious error could occur.*
2. *In C mode, we recommend you enable the **Add new file to project** check box" to add the first C file in a new project. eUIDE supplies "main()" function, interrupt save procedure, interrupt service routine frame, restore code in the file to develop project, and write code on interrupt easily(see figure below). Interrupt is discussed in detail in Section 6.10, "Interrupt Routine." Interrupt program is very easy to develop under TCC2.*
3. *Remember that C compiler can only accept one "main()" function in a project. If you want to add another new file after the first main file was added, you can enable the **Empty File** check box to add an empty file*

```
01 void main()
02 {
03   _asm{MOV A,@0x10
04        CONTW
05    }
06 }
07
08 void _intcall interrupt(void) @ int
09 {
10
11  // Write your code (inline assembly or C) here
12
13 }
14
15 void _intcall interrupt_1(void) @ 0x08:low_int 0
16 {
17   _asm{PAGE @0x0}
18 }
```

Figure 3-6b Typical Interrupt Function

3.5.2 Add Existing Source Files to the New Project

If your source file is ready, you can immediately insert it into your new project.

1. From the Menu bar, click on **Project → Add Files to Project**. The **Insert Files into Project** dialog is then displayed.

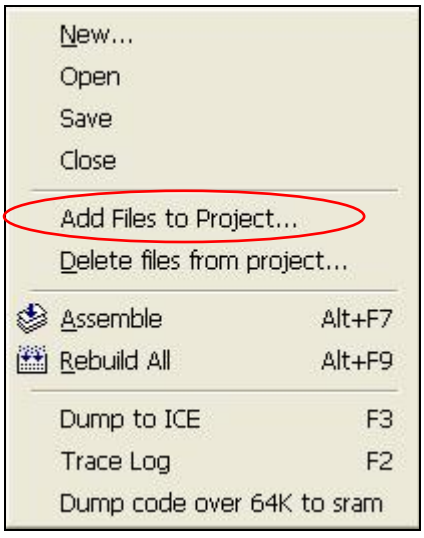


*Figure 3-7a **Add Files to Project** Command of **Project** Menu*

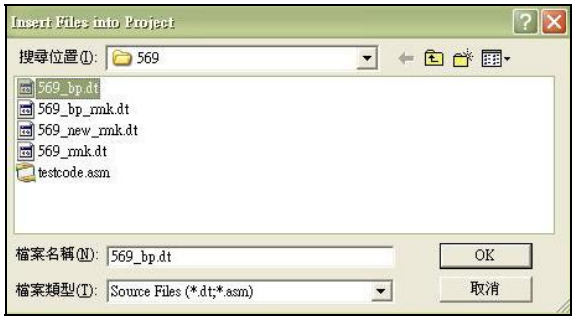


*Figure 3-7b **Insert Files into Project** Dialog*

2. Browse and select the source file (or multiple files) you intend to insert into the new project.
3. Click **OK** button to insert file into your new project after confirming your selection.

3.5.3 Deleting Source Files from Project

From the **Project** window, select the file(s) you wish to delete. Then press the **Delete** key from your keyboard or click **Project → Delete Files from Project…**.

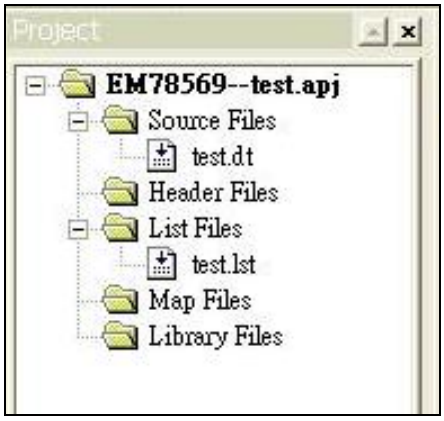


*Figure 3-8a Deleting Source Files Directly from **Project** Window*

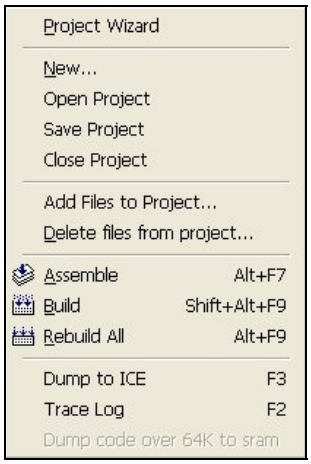


*Figure 3-8b Deleting Project Files from **Project** Menu.*

3.6 Editing Source Files from Folder/Project

3.6.1 Open Source File from Folder for Editing

You can also open an existing source file in the **Editor** window for a last minute editing before adding it into the new project. To do this–

1. Click **File → Open** command.
2. From the resulting **Insert Files into Project** dialog (*Figure 3.7b* above), select and click on the source file you want to edit. The file is automatically opened in the **Editor** window.

To edit source files that are already added into the Project, see next Section.

3.6.2 Open Source File from Project for Editing

You can edit source files that are already inserted in the project. To do so, double click the source file you wish to edit from the **Project** window and the file will open in the **Editor** window.

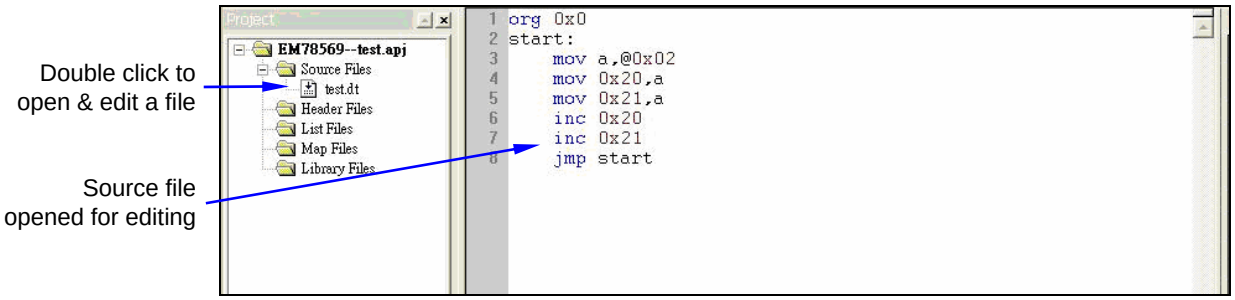


*Figure 3-9 Editing Source File Directly from **Project** Window.*

3.7 Compile the Project

With your source file(s) embedded into the project, you are now ready to compile your project using the following commands from **Project** menu.

- Click **Assemble (or Compile)** command to compile the active file only (generates **.lst*).
- Click **Build** command to compile the files modified in the project they were modified.
- Click **Rebuild All** command to compile all files in the project regardless of whether they were modified or not.

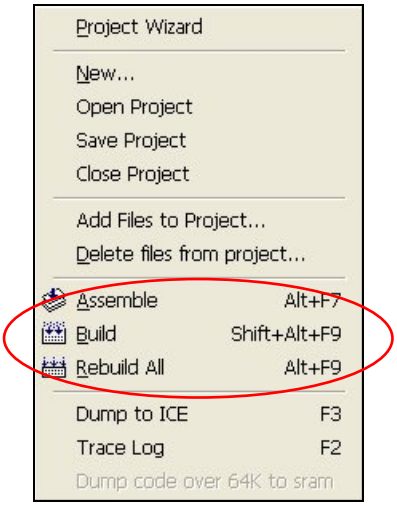


*Figure 3-10a Compilation Commands from **Project** Menu*

Note that in Asssembly mode, **Build** and **Rebuild All** commands will generate objective (**.bbj*) file, list (**.lst*) file, and binary (**.cds*) file. In C mode, both commands will generate objective (**.o*) file, assembly (**.s*) file, and binary (**.cds*) file.

The compiled files are automatically saved in the same folder where your other source files are located. Status of the assembly operation can be monitored from the **Output** window as shown below.

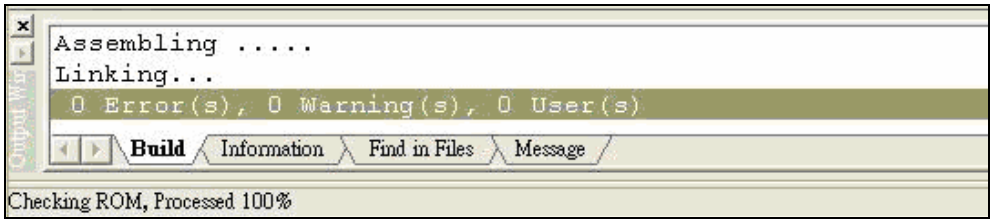


*Figure 3-10b **Output** Window Showing Successful Compilation*

If error is detected during compilation, pertinent error message will also display in the **Output** window with **Build** tag. Double click on the error message to link to the source of error (text line) in the corresponding source file displayed in the **Editor** window. If the corresponding source file is not currently opened, it will be opened automatically.

Double click to link to the source of error

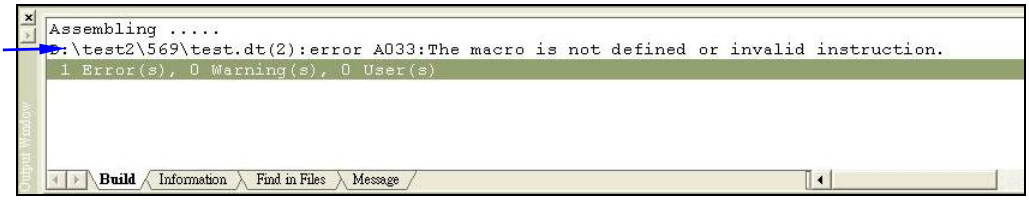


*Figure 3-10c **Output** Window Showing Compilation Errors*

Modify source files to correct the errors and repeat assembling and linking operations.

In C mode, there are many useful messages reported in the **Output** window's **Information** tab if compiling succeeds. For example, it tells you the used ROM size, available ROM size, used RAM data in figure, used data register in figure, IO control data in figure, call depth, and max call depth. The most important message is the characters "C" in register data location *0x10~0x1F*. These "C" characters tell programmer which and how many common registers to save and restore in interrupt service routine. Refer to Section 6.1, *Register Page (rpage)* for further details.

Figure below shows these messages which advice you to save common register 0x10 and 0x11 when MCU has just ran into interrupt service for C system and restore these two common registers before leaving interrupt service.

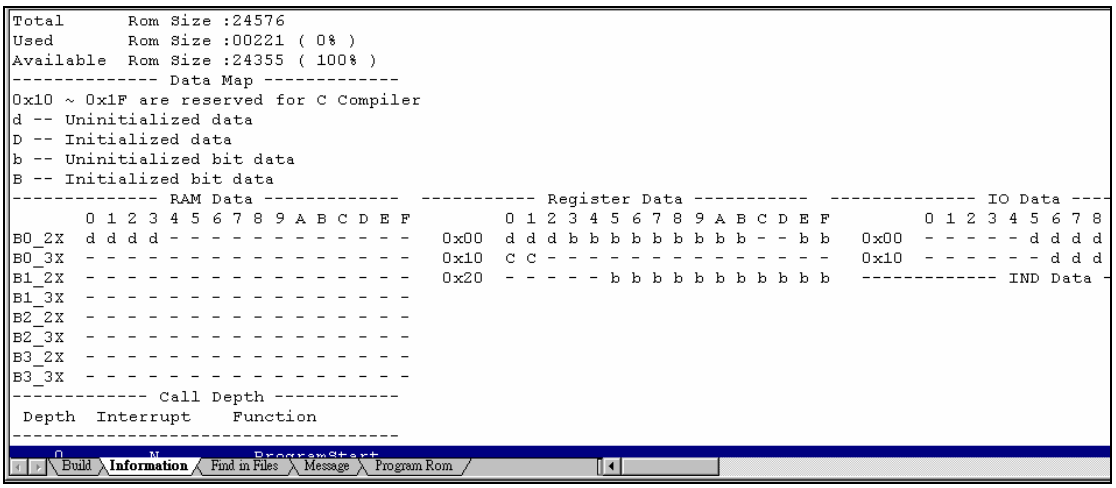


*Figure 3-10d **Output** Window Messages after Compiling Successfully*

3.8 Dumping the Compiled Program to ICE

With the source files deprived of its errors and successfully compiled, download your compiled program to ICE by clicking **Project → Dump to ICE** or its corresponding shortcut key (F3).

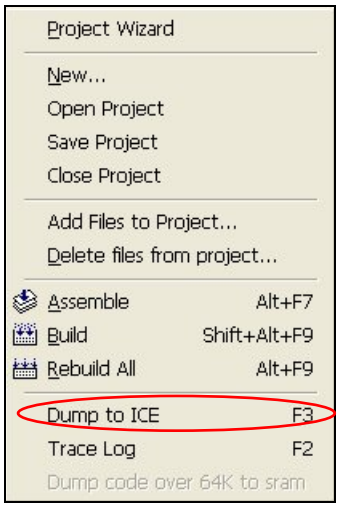


*Figure 3-11 **Dump to ICE** Command from **Project** Menu*

3.9 Debugging a Project

With the compiled program successfully downloaded to ICE, you are now ready to debug the files. Be sure the ICE is properly connected to your computer.

Full debugging commands are available from the **Debug** Menu (shown with its corresponding shortcut keys in the drop-down menu at right). A number of the frequently used debugging icons are also available from the eUIDE Program Toolbar.

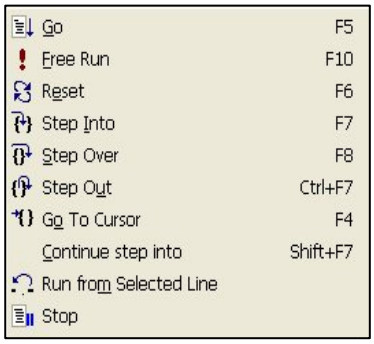


Fig. 3-12a Debugging Commands Drop-Down Menu

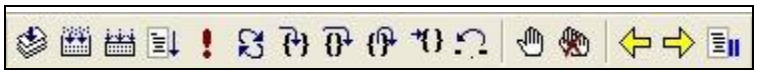


Figure 3-12b Toolbar for Debugging Commands

Where:


F5

Go – Auto dump and run program starting from the current program counter until breakpoint is matched and breakpoint address is executed.


F10

Free Run – Auto dump and run program starting from the current program counter until the **Stop** button of the **Stop Running** dialog is clicked. All defined breakpoints are ignored while the program is running.


F6

Reset – Perform hardware reset (register contents are displayed with initial values). ICE will return to its initial condition.


F7

Step Into – Auto dump and **e**xecute instructions step-by-step including subroutines (with register contents updated at the same time).

If the RAM of your computer is full, do not click command continuously.


F8

Step Over – Same as **Step Into** command (see above), but excluding subroutines and the CALL instruction is executed as **Go** command.

If the RAM of your computer is full, do not click command continuously.


Ctrl+F7

Step Out – Auto dump and run program starting from the current program counter until the RET / RETI / RETL instruction address is executed.


F4

Go to Cursor– Auto dump and execute from current program counter to the location where the cursor is positioned with breakpoints ignored.

Run from Selected Line – Start running command from the line where the cursor is positioned.


F9

Toggle Breakpoint – Click command with cursor positioned on the line. If line is set with breakpoint, it will be removed. Otherwise, breakpoint is set.

Clear All Breakpoints – Remove all existing breakpoints.

Trace Back After trace log is executed; trace the log backward step-by-step from the last executed address to the address located before the current executed address (bottom to top).

Trace Forth After trace log is executed, retrace the log forward step-by-step from the last executed address to the address where **Trace Back** was started (top to bottom).

Stop: Stop **Free Run** or **Go** with breakpoint ignored.

During debugging, the contents of Program Counter, Registers, and RAMs are read and displayed each time the program is stopped to provide important interim information during program debugging.

3.9.1 Breakpoints Setting

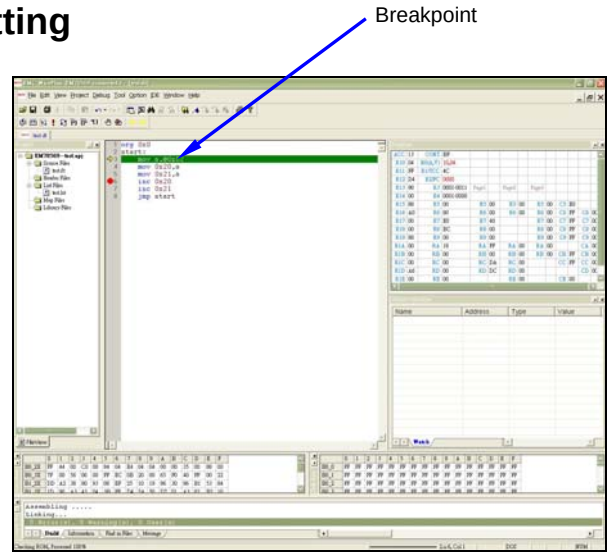


Figure 3-13 Active Source File with a Defined Breakpoint

To assign a breakpoint, position cursor on the line where a breakpoint is going to be set, then double click. Observe the line highlighted in green.

With cursor positioned on the line, you can also click on the **Insert/ Remove Breakpoint** icon (hand shape) on the toolbar to set a breakpoint, or press F9.

Likewise, the defined breakpoint is cleared if you double click on it again, or the hand icon is clicked the second time while the cursor is positioned on the defined breakpoint. To clear all existing breakpoints, click **Clear All Breakpoints** command from **Debug** menu.

Chapter 4
Assembler and Linker

4.1 Assembler and Linker Process Flow

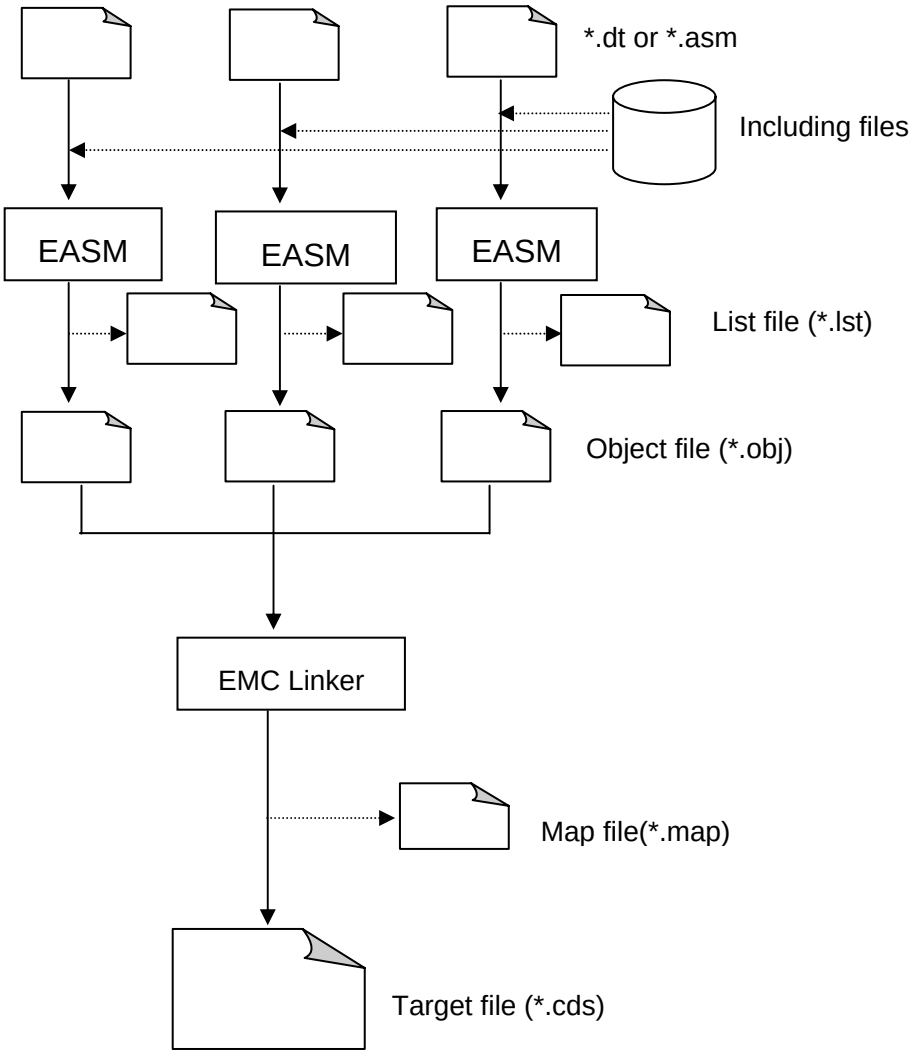


Figure 4-1 Assembler and Linker Process Flow Chart

4.2 Statement Syntax

[label [:]] operation [operand] [,operand][; comment]

All fields are not case-sensitive, and separated by space or tab.

Label—The [:](colon) is optional and is followed by one or more spaces or tabs.

A label consists of the following characters-

A~Z a~z 0~9 _

but with some restrictions**:**

- 0~9 must not be the first character of a name
- Only the first 31 characters are recognized
- To reserve colon (:) is recommend to programmers, because it could be more readable.

4.2.1 How to Define Label

a) zeroflag equ RXX(.YY)

Ex1: zeroflag == R3.2

Ex2: status == R3

b) zeroflag_1 equ 0xXX(.YY)

Ex1: zeroflag_1 == 0x3.2

Ex2: status_1 == 0x3

c) zeroflag_2 equ zeroflag(.ZZ) (zeroflag_1 equ RXX(.YY))

Ex: status_2 == R3

zeroflag_2 == Status_2.2

d) zeroflag_3 equ zeroflag1(.ZZ) (zeroflag equ 0xXX(.YY))

Ex: status_3 == 0x3

zeroflag_3 == Status_3.2

e) Add label with rpage / rbank / iopage

“range” and select “Special Register(R0..R1F)” from **Watch** dialog are the same.

“rbank” and select “RAM(bank)” from **Watch** dialog are the same.

“iopage” and select “Control Register” from **Watch** dialog are the same.

Ex: status_3 == 0x3:rpage 0
zeroflag_3 == status_3.2 (status_3 is already defined in rpage 0)
temp == 0x20:rbank 1
output == 0x6:iopage 1
outputbit_2 == 0x6.2 (output is already defined in iopage1)

IMPORTANT!!
0x3:rpage 0 → *Do not insert space before and after colon (:) character.*

f) Derive Register or Bit information from label__R(Duplicate _) or label__B:

Ex: zeroflag == 0x3.2
mov a, zeroflag__R ;equal mov a, 0x3
mov a, zeroflag__B ;equal mov a, 0x2

IMPORTANT!!
*This method is only displayed automatically in **Watch** window. When using this variable, you must use register that is defined in the ICE specification for register or bank page change.*

Operation—An assembler instruction or directive
Directives give the direction to the assembler
Instruction examples**:**

Example 1: MOV A,@0X20
Example 2: ADD A,@0X20
Example 3: zeroflag == R3.2

```
status == R3
carryflag == status.0

org 0x0
jmp start
start:
BC zeroflag
BS status,2
BC carryflag
BS status,0
BC 0x3,2
BS R3,2
```

Examples of directives**:**

Example1: ORG 0X20
Example2: END

Operands—One or more operands separated by commas

Comment—Comments include line comment and block comment

Line comment syntax: preceded by a "**;**" (semi-colon)

Example**:** MOV A,@0X20 **;** move constant value 32 to accumulator

Block comment: /* comment statements */

4.3 Number Type

Type	Expression 1	Expression 2	Expression 3
Decimal	0D<digits>	<digits>D	<digits>
Hexadecimal	0X<digits>	<digits>H	
Octal	0Q<digits>	<digits>Q	
Binary	0B<digits>	<digits>B	

NOTE

If the first digit is "A~F" or "a~f" on hexadecimal expression 2, then you must add "0" to precede the digits.

4.4 Assembler Arithmetic Operation

The arithmetic result must be calculated after assembler. If the arithmetic expression cannot be calculated on assembler time, then error message will display. At the same time, the arithmetic expression cannot support floating point, so floating point number is self-transferred to integer.

TRUE and FALSE**:** TRUE is 0xFF, FALSE is 0x00.

The following operators give a summary of the operators, in order of priority.

a) Parentheses (and)

b) Unary operators

! Logical NOT

~ Complement

– Unary minus

c) Multiplication, division, modulo, shift

*	Multiplication
/	Division
%	Modulo
<<	Logical shift left
>>	Logical shift right

d) Addition arithmetic operators

+	Addition
–	Subtraction

e) Bit AND operator

&	Bit AND

f) Bit OR and XOR operators

|	Bit OR
^	Bit XOR

g) Logical AND

&&	Logical AND

h) Logical OR

||	Logical OR

i) Comparison

==	equal
!=	not equal
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to

4.5 Program Directives

a) ORG: set value of program counter

Syntax**:** ORG <expression>

Example: org 0x200

b) EQU or == (Duplicate =): definition constant value

Syntax**:** <label> EQU<expression>

Example 1: R20 equ 0x20

Example 2: R20 == 0x20

c) Comment

Line comment

Syntax**: ;** < string >

Example: **;** this is the comment string

Block comment

Syntax**:** /* <strings> */

Example: /* this is block comment example including multi lines */

d) EOP: end of the program ROM page with which the EOP instruction is belong to

Syntax**:** EOP

```
Example:    org     0x10
            mov     0x20,A
            inc     0x20
            eop
            inc     0x20
```

Result after assembling (the first column is address)**:**

```
            org     0x10
0010        mov     0x20,A
0011        inc     0x20
            eop
0400        inc     0x20
```

e) END: the end of program. The rest of the program code after END instruction will not be assembled.

Syntax**:** **END**

```
Example:    org     0x10
            mov     0x20,a
            inc     0x20
            end
            mov     0x20,a
```

Result after assembling (the first column is address):

```
            org     0x10
0010        mov     0x20,a
0011        inc     0x20
            end
            mov     0x20,a
```

f) PROC, ENDP: definition of subroutine. The directives make the program more comprehensible.

Syntax: **<label> PROC**
<statements>
ENDP

Example:
```
BANK0:  PROC
        BC      0X04,6
        BC      0X04,7
        RET
        ENDP
```

> **NOTE**
> *PROC and ENDP directives only make the program more comprehensible. So RET instruction must exist in the subroutine.*

g) INCLUDE: include other source files. The instruction makes the program more refined and clear.

INCLUDE function can include system default files and user defined files.

1) Include system default files; e.g., EMC456.INC, EMC32.INC**:**

Syntax**: INCLUDE <filename>**

Example: INCLUDE <EMC456.INC>

2) Include user defined files**:**

Syntax**: INCLUDE "file path + file name"**

Example: INCLUDE "C:\EMC\TEST\TEST456.INC"

The user defined file must include full folder path and filename.

> **NOTE**
> *The source files usually include variable definition, macro definition, and subroutine definition.*

h) PUBLIC and EXTERN: The defined scope of the global label is public or external. Although the eUIDE software is project oriented, a project can contain two or more files. If the global label is referenced by another file, the global label must be defined to PUBLIC in the defined file, and must be defined to EXTERN in the referenced file.

PUBLIC:

Syntax**: PUBLIC <label>[,<label>]**

EXTERN:

Syntax**: EXTERN <label>[,<label>]**

PUBLIC and EXTERN instructions can be defined at any location of a file that contains one or more PUBLIC or EXTERN instructions.

Example: A project contains two files, one is TEST1.DT; the other is TEST2.DT.

```
TEST1.DT:
        org  0x00
        Public start
        Extern  loop1
Start:
        mov   a,@0x02
        mov   0x20,a
        jmp    loop1
TEST2.DT:
        org  0x100
        Public  loop1
        Extern  start
Loop1:
        inc  0x20
        jmp  start
```

The label of “Start”, which is defined in the “TEST1.DT” file and is referenced by the “TEST2.DT”file; must be announced as “PUBLIC” in the “TEST1.DT” file and “EXTERN” in the “TEST2.DT” file.

The label of “loop1”, which is defined in the “TEST2.DT” file and is referenced by the “TEST1.DT” file; has to be addressed as “EXTERN” in the ‘TEST1.DT” file and “PUBLIC” in the “TEST2.DT” file.

i) VAR: The instruction defines variable name during assembly time, so the value of variable is only changed during the assembly time.

Syntax**: Label VAR <expression>**

```
Example:  test    var  1
          mov     a,@test
          test    var  test+1
          mov     a,@test
```

j) MACRO, ENDM: The instruction defines macro

Syntax**:** **<label> MACRO <parameters>**
statements
ENDM

Example: bank0 macro
bc 0x04,6
bc 0x04,7
endm

IMPORTANT!!

- *The maximum number of macro is 32*
- *The parameters must be a definite address as shown in the examples below.*

```
aa_label macro num
    if num >= 0x400
        bc 0x3,6
        bs 0x3,5
    else
        bc 0x3,6
        bc 0x3,5
    endif
endm

org 0xfff
jmp start
org 0x0
start:
    aa_label bb_label
    call bb_label
    aa_label $
    jmp $

org 0x400
bb_label:
     inc 0x20
     ret
```

Example1:

If “org 0x400” is omitted, the following conditions will result**:**

a) The required “**aa_label**” macro parameter (num) becomes unknown. Hence, macro cannot expand.

b) As macro fails, all its instructions cannot be implemented.

c) The error message “*error A037: The operand value cannot be calculated*” will then display

To prevent the above error, address “org 0x400” should be added before “**bb_label.**”

```
FCALL  MACRO  addr
    IF  (addr/0x400) != ($/0x400)
        LCALL  addr
    ELSE
        CALL  addr
    ENDIF
ENDM

FJMP  MACRO  addr
    IF  (addr/0x400) != ($/0x400)
        LJMP  addr
    ELSE
        JMP  addr
    ENDIF
ENDM
```

Example2: (EM78P510N)

FCALL and **FJMP** are reserved words. These two words will determine whether the program code exceeds page or not.

```
FPAGE MACRO NUMB
    IF  NUMB==0
        BC  0x3.5
        BC  0x3.6
    ELSEIF  NUMB==1
        BS  0x3.5
        BC  0x3.6
    ELSEIF  NUMB==2
        BC  0x3.5
        BS  0x3.6
    ELSEIF  NUMB==3
        BS  0x3.5
        BS  0x3.6
    ELSE
        ERROR
    ENDIF
ENDM

FJMP MACRO ADDRESS
    IF ADDRESS/0X400!=$/0X400
        FPAGE   ADDRESS/0X400
    ENDIF
        JMP ADDRESS
ENDM

FCALL MACRO ADDRESS
    IF ADDRESS/0X400!=$/0X400
        FPAGE   ADDRESS/0X400
        CALL    ADDRESS
        FPAGE   $/0X400
    ELSE
        CALL    ADDRESS
    ENDIF
ENDM
```

Example3: (EM78P447S)

FPAGE, FCALL, and **FJMP** are reserved words. These two words will determine whether the program code is over page or not.

k) MACEXIT: this instruction is only used in the macro defined instructions. If the MACEXIT instruction is assembled, then the remaining macro instructions (if any) are not assembled.

Syntax**: MACEXIT**

Example: Source:

```
test  var   5
bank0macro
      bc  0x04,6
      if  test>4
        macexit
      endif
      bc  0x04,7
endm
bank0
```

After assembly…(the first column is address)

```
0000 bc  0x04,6
```

Because the “test” variable is equal to five, so the expression “test>4” is true, and the “macexit” instruction is assembled. Accordingly, as “macexit” instruction is assembled, the remaining macro instructions, “bc 0x04,7,”is not assembled.

l) MESSAGE: display the user defined message in the **Output** window.

Syntax: **MESSAGE “<characters>”**

Example**:**

```
org       0x00
message “set bank to 0 !!”
          bc   0x04,6
          bc   0x04,7
```

The user defined message (see below) is displayed in the **Output** window after assembly.

USER MESSAGE: set bank to 0!!

IMPORTANT!!

The maximum number of message is 500.

m) $: current program counter value
"$" is used as an operand.

Example1: jmp $
"jmp $" means to jump at same line as dead loop

Example2: bc 0x04,6
jmp $-1
"jmp $-1" means to jump back to "bc 0x04,6"

4.6 Conditional Assembly

a) IF If the statement after "IF" expression is true, then the following instructions are assembled until "ELSEIF" or "ELSE" or "ENDIF".

Syntax**: IF <expression>**
<statements1>
[ELSEIF <expression>
<statements2>]
[ELSE
<statements3>]
ENDIF

Example**:**
```
org 0x00
bank  macro  num
if num==0
    bc  0x04,6
    bc  0x04,7
elseif num==1
    bs  0x04,6
    bc  0x04,7
elseif num==2
    bc  0x04,6
    bs  0x04,7
elseif num==3
    bs  0x04,6
    bs  0x04,7
else
    message "error : bank num over max number!!!"
endif
endm
```

b) IFE: If the statement after "IFE" expression is false, then the following instructions are assembled until "ELSEIFE" or "ELSE" or "ENDIF".

Syntax: **IFE <expression>**
<statements1>
[ELSEIFE <expression>
<statements2>]
[ELSE
<statements3>]
ENDIF

c) IFDEF: If the statement after "IFDEF" label is defined, then the following instructions are assembled until "ELSEIFDEF" or "ELSE" or "ENDIF".

Syntax: **IFDEF <label>**
<statements1>
[ELSEIFDEF <label>
<statements2>]
[ELSE
<statements3>]
ENDIF

Example:
```
org 0x00
ice456  equ  456
ifdef ice456
        bc  0x04,6
        bc  0x04,7
endif
```

d) IFNDEF: If the statement after "IFNDEF" label is not defined, then the following instructions are assembled until "ELSEIFNDEF" or "ELSE" or "ENDIF".

Syntax: **IFNDEF <label>**
<statements1>
[ELSEIFNDEF <label>
<statements2>]
[ELSE
<statements3>]
ENDIF

4.7 Reserved Word

4.7.1 Directives, Operators

+	-	*	/	==
!=	$	@	#	(
)	!	~	%	<<
>>	&	|	^	&&
||	<	<=	>	>=
DS	ELSE	ELSEIF	ELSEIFDEF	ELSEIFE
ELSEIFNDEF	END	ENDIF	ENDM	ENDMOD
ENDP	EQU	EXTERN	IF	IFE
IFDEF	IFNDEF	INCLUDE	MACRO	MACEXIT
MODULE	NOP	PAGE	ORG	PROC
PUBLIC				

4.7.2 Instructions Mnemonics

ADD	AND	BC	BS	CALL
CLR	COM	COMA	CONTR	CONTW
DAA	DEC	DECA	DISI	DJZ
DJZA	ENI	INC	INCA	INT
IOR	IOW	JBC	JBS	JMP
JZ	JZA	LCALL	LJMP	MOV
NOP	OR	RET	RETI	RETL
RLC	RLCA	RRC	RRCA	SLEP
SUB	SWAP	TBL	WDTC	XOR
LCALL	LPAGE	PAGE	BANK	

NOTE

Some MCU do not have LCALL, LJMP, PAGE, and BANK instructions. You have to study the specific MCU specification.

4.8 Pseudo Instruction

The following pseudo instructions are supported for MCUs that have LJMP/LCALL instructions.

Macro Instruction	Syntax	Note	Equivalent Instruction
XCALL	XCALL Label	If “label” is in current page, XCALL will be “CALL, otherwise it will be “LCALL”	CALL OR LCALL
XJMP	XJMP Label	If “label” is in current page, XJMP will be “JMP, otherwise it will be “LJMP”	JMP or LJMP
JCF	JCF Label	If carry flag =1, then JMP to label	JBC 0x3,0 XJMP label
JZF	JZF Label	If zero flag =1, then JMP to label	JBC 0x3,2 XJMP label
ADDCF	ADDCF R	R + Carry -> R	JBC 0x3,0 INC R
SUBCF	SUBCF R	R – Carry -> R	JBC 0x3,0 DEC R

Chapter 5

C Fundamental Elements

5.1 Comments

For a single line comment**:**

//	All data in the line after the comment symbol (twin-slash mark) will be ignored.

For Multi line comments**:**

/* … */	All data in the line located within the comment symbols (slash mark + asterisk) will be ignored.

Comments are used to help you understand the program code. It can be placed anywhere in the source program. The compiler will ignore the comment segment of the source code, thus no extra memory is required in the program execution.

Example:

```
// This is a single line comment

/*
This is the comment line 1
This is the comment line 2 */
```

5.2 Reserved Words

The reserved words for eUIDE C Compiler are made up of both the ANSI C conformity reserved words and the EM78 Series unique reserved words. The following table summarizes all the applicable reserved words for this compiler.

ANSI C Conformity Words					
const	default	goto	switch	typedef	sizeof
break	do	if	short	union	extern
case	else	int	signed	unsigned	
char	enum	long	static	void	
continue	for	return	struct	while	

EM78 Series Unique Words				
indir	ind	page	on	off
io	iopage	_intcall	rpage	
low_int	_asm	bit	bank	

NOTE

- ***Double*** *and* ***float*** *are* ***NOT*** *supported by the EM78 Series C Compiler.*
- *_asm is added for the EM78 Series C compiler. Do not use _ASM.*
- *indir, ind, io, iopage, rpage are used for MCU hardware definition and declaration.*

5.3 Preprocessor Directives

Preprocessor directives always begin with a pound sign (#). The directives are recognized and interpreted by the preprocessor in order to compile the source code properly.

5.3.1 #include

#include "file_name":	The preprocessor will search the working directory to find the file.
#include <file_name>:	The preprocessor will search through the working directory first to look for the file. If the file cannot be found in the working directory, it will search the file from the directory specified by the environment variable ELAN_TCC_INCLUDE.

"#include" tells the preprocessor to add the contents of a header file into the source program.

NOTE

- *It is not recommended to include C file. Errors may occur if C file is included.*
- *Suppose uaa is declared as global, unsigned int (unsigned int uaa) variable in headfile.h. Then, uaa is used in testcode.c. If you want to use uaa in kkdr.c, first you have to declare extern unsigned int uaa before you using it in kkdr.c file. The same approach should be used in the third or more others c source files that are to be included in the same project.*

Example 1:

```
#include <EM78.h>
#include “project.h”
#include “ad.c”              // It may meet errors.
```

Example 2:

```
unsigned int uaa;           //in headfile.h
…

#include ”headfile.h”       //in testcode.c file
#include “kkdr.h”           //in testcode.c file
#include “pprr.h”           //in testcode.c file
main ()                     //in testcode.c file
{
   uaa=0x21;
   test1();
   test2();
…
}

test1();                    //  in kkdr.h file
..

#include “pprr.h”           //in kkdr.c file
extern unsigned int uaa;    //in kkdr.c file
void test1()                //in kkdr.c file
{
   uaa=0x38;
   test2();
……
   uaa=0x43;

}

test2()                     //in pprr.h file
……

extern unsigned int uaa;    //in pprr.c file
void test2()                //in pprr.c file
{
   uaa=0x29;
}
```

5.3.2 #define

#define identifier
#define identifier token_list
#define identifier (parameter_list) token_list
#define identifier( ) token_list

The "#define" directive is used to define a string constant which will be substituted into source code by the preprocessor. It makes the source program more clear and logical.

NOTE

Multi-line macro definition should be cascaded with a backslash (\ ) in between the lines. When using assembly code in macro, use ONLY one instruction in a line.

Example:

```
#define MAXVAUE 10
#define sqr2(x, y) x * x + y * y
```

5.3.3 #if, #else, #elif, #endif

#if constant_expression
#else
#elif constant_expression
#endif

The #if directive is used for conditional compilation. It should be terminated by #endif. #else can be used to provide an alternative compilation. If necessary, the program can use #elif for an alternative compilation which should only be used for valid expressions.

Example:

```
#define RAM 30
#if (RAM < 10)
        #define MAXVALUE 0
#elif (RAM < 30)
        #define MAXVALUE 10
#else
        #define MAXVALUE 30
#endif
```

5.3.4 #ifdef, #ifndef

#ifdef identifier **#ifndef identifier**

The “#ifdef” directive is used for conditional compilation of definitions for the identifier. The “#ifndef” directive is used when the conditional compiling codes of the specified symbol is not defined. Both these two directives must be terminated by “#endif” and can be optionally used with “#else.”

Example:

```
#define DEBUG   1
#ifdef (DEBUG)
       #define MAXVALUE 10
#else
       #define MAXVALUE 1000
#endif
```

5.4 Literal Constants

5.4.1 Numeric Constant

Decimal:	Default
Hexadecimal constant:	Digit prefix with “0x”
Binary digit:	Digit prefix with “0b”

“Numeric constants” can be presented in decimal and hexadecimal, depending on the prefix modifier. Binary and octonary numerics are not supported.

Example:

```
12, 34      // Decimal
0x5A, 0xB2  // Hexadecimal
0b10111001  // Binary digit
```

5.4.2 Character Constant

'character'

Character constants are denoted by a single character enclosed by single quotes. ANSI C Escape Sequences as shown below are treated as a single character.

ANSI C Escape Sequence		
Escape Character	**Meaning**	**Hexadecimal**
\0	Null	00
\a	Bell (Alert)	07
\b	Backspace	08
\f	Form Feed	0C
\n	New Line	0A
\r	Carriage Return	0D
\t	Horizontal Tab	09
\v	Vertical Tab	0B
\\	Backslash	5C
\?	Question Mark	3F
\'	Single Quote	27
\"	Double Quote	22

Example:

```
'a', 'b','c', /x00
```

5.4.3 String Constant

"character_list"

String constants are series of characters enclosed in double quotes, and which have an implied null value ('\0') after the last character.

NOTE

It takes one more character space for constant string to store the null value.

Example:

```
"Hello World"
"Elan Micro"
```

5.5 Data Type

The size and range (maximum and minimum values) of the basic data type are as shown below.

Type	Range	Storage Size (Byte)
void	N/A	None
(unsigned) char	0 ~ 255	1
signed char	–128 ~ 127	1
(signed) int	-128 ~ 127	1
unsigned int	0 ~ 255	1
(signed) short	–32768 ~ 32767	2
unsigned short	0 ~ 65535	2
(signed) long	–2147483648 ~ 2147483647	4
unsigned long	0 ~ 4294967295	4
bit	0 ~ 1	1 (Bit)

NOTE

1. ***Floating*** *and* ***Double*** *types are not supported.*
2. *See Section 6.4 of Chapter 5 for more details on "Bit Data Type".*
3. *If you use* ***long*** *data type for multiplication, division, modulus, and compare operations, the 0x20 ~ 0x24 (5 bytes) of Bank 0 are exclusively reserved for compiler use. Therefore, do not assign these addresses to any variable when you perform the above mentioned operations.*

When an arithmetic operator, such as, "*", "/", and "**%**" is used with different data types, conversion of right-aligned variables to left-aligned data type is done before the operator takes effect. We suggest you use the same data type to develop program.

Example:

```
Int I1, I2;
Short S1, S2, S3;
Long L1, L2;
I1 = 0x11;
I2 = 0x22;
S1 = I1 * I2;  ➔  change to S1 = (short) I1 * (short) I2;
        // If forgot to add "(short)" before I1 and I2, the final
        // result (in S1) will be 1 byte only
S1 = 0x1111;
S2 = 0x02;
L1 = S1 / S2;  ➔  change to L1 = (long) S1 * (long) S2;
        // If forgot to add "(long)" before S1 and S2, the final
        // result (in L1) will be 2 bytes only.
```

Tiny C compiler uses two's compliment to perform mathematical negative signed declaration.

Example:

Assume abc is allocated at 0x20, Bank 1. Then from **RAM Bank** window you will see E8 being displayed at that location.

```
int abc;
unsigned int uir;
abc=-0x18;
```

5.6 Enumeration

enum identifier
enum idenftifier {enumeration-list [=int_value]...}
enum {enumeration-list}

Enumeration defines a series of named integer constants. With the definition, the integer constants are grouped together with a valid name. For each name enumerated, you can specify a distinct value.

Example:

```
enum tagLedGroup {LedOff, LedOn} LEDStatus;
```

5.7 Structure and Union

struct (union)-type-name:
struct (union) identifier
struct (union) identifier {member-declaration-list}
struct (union) member-declaration-list

member-declaration-list:
member-declaration
member-declaration-list member-declaration

member-declaration:
member-declaration-specifiers declaration-list

member-declaration-specifiers:
member-declaration-specifier
member-declaration-specifiers member-declaration-specifier

The structure group related data and each data in the structure can be accessed through a common name. Unions are groups of variables that share the same memory space.

NOTE

- *Do not use **bit** data type in structure and union. use bit field instead.*
- *Structure and union cannot be used in function parameter.*

Example 1:

```
struct st
{
   unsigned int b0:1;
   unsigned int b1:1;
   unsigned int b2:1;
   unsigned int b3:1;
   unsigned int b4:1;
   unsigned int b5:1;
   unsigned int b6:1;
   unsigned int b7:1;
};
struct st R5@0x05: rpage 0;  //struct R5 is related to 0x05,
                                rpage 0
```

Example 2:

```
struct tagSpeechInfo{
   short rate;
   long size;
} SpeechInfo;

union tagTest{
   char Test[2];
   long RWport;
} Test;
```

5.8 Array

declarator:
array-declarator:
array-declarator
[constant-expression]
array-declarator [constant-expression]

An “Array” is a collection of same type data and can be accessed with the same name.

NOTE

- *If “const” is used to declare an array, the data will be placed at the program ROM.*
- *The maximum size of a constant array variable is 255 bytes.*
- *The maximum size of an array is 32 bytes (RAM bank).*

Example:

```
int array1 [3] [10]
char port [4]
const int myarr [2] = {0x11, 0x22};
            // 0x11, 0x22 will be put at program rom
```

5.9 Pointer

declarator
type-qualifier-list * declarator

A pointer is an index which holds the location of another data or a NULL constant. All types of pointer occupy 1 byte.

NOTE

Function pointer is not supported.

Example:

```
int *pt;
```

5.10 Operators

5.10.1 Types of Supported Operators

The supported operators for the C expression are as follows**:**

- Arithmetic operators
- Increment and decrement operators
- Assignment operators
- Logical operators
- Bitwise operators
- Equality and relational operators
- Compound assignment operators

The table below shows the detailed description of each of the operators**:**

Arithmetic Operators		
Symbol	**Function**	**Expression**
+	addition	expr1 + expr2
–	subtraction	expr1 – expr2
*	multiplication	expr1 * expr2
/	division	expr1 / expr2
%	modulo	expr1 % expr2

Arithmetic Operators		
Symbol	**Function**	**Expression**
++	increase by 1	expr ++
--	decrease by 1	expr --

Arithmetic Operators		
Symbol	**Function**	**Expression**
=	equal	expr1 = expr2

Arithmetic Operators		
Symbol	**Function**	**Expression**
&	bitwise AND	expr1 & epxr2
|	bitwise OR	expr1 | expr2
~	bitwise NOT	~expr
>>	right shift	expr1 >> expr2
<<	left shift	expr1 << expr2
^	bitwise XOR	expr1^expr2

Equality, Relational, and Logical Operators			
Symbol	**Function**	**Expression**	**Example**
<	Less than	expr < expr	x < y
<=	Less than or equal	expr <= expr	x <= y
>	Greater than	expr > expr	x > y
>=	Greater than or equal	expr >= expr	x >= y
==	Equality	expr == expr	x == y
!=	Inequality	expr != expr	x != y
&&	Logic AND	expr && expr	x && y
||	Logic OR	expr || expr	x || y
!	Logic NOT	!expr	!x

Compound Assignment Operators		
Symbol	**Function**	**Example**
+=	y=y + x	x += y
-=	y = y - x	x -= y
<<=	y = y << x	y<<=x
>>=	y= y >> x	y>>=x
&=	y= y & x	y&=x
^=	y= y ^ x	y^=x
|=	y= y | x	y |= x

5.10.2 Prefix of Operators

Priority	Same Level Operators, from Left To Right
Highest	( ) [ ] - >
	! ~ ++ -- -(unary) +(unary) (type_cast) *(indirection) & (address) sizeof
	* / %
	+ -
	<< >>
	< <= > >=
	== !=
	&
	^
	|
	&&
	||
	?:
	= += -= *= /= %= >>= <<= &= |= ^=
Lowest	,

5.11 If-else Statement

if (expression) statement
else statement

"If" statement executes the block of codes associated with it when the evaluated condition is true. It is optional to have an "else" block is executed when the evaluated condition is false.

Example:

```
if (flag == 1)
{
      timeout=1;
      flag=0;
}
else
      timeout=0;
```

5.12 Switch Statement

```
switch (expression)
{
        case const-expr: statements
        case const-expr: statements
        default: statements
}
```

"Switch" statement is flexible to be set with multiple branches depending on a single evaluated value.

NOTE

The expression will be checked as INT type, thus only 256 cases can be used in a switch.

Example:

```
switch (I)
{
        case 0: function0();
          break;
        case 1: function1();
          break;
        case 2: function2();
          break;
        default: funerror();
}
```

5.13 While Statement

```
while (expression) statement
```

"While" statement will first check the expression. If the expression is true, it will then execute the statement.

Example:

```
while (value != 0)
{
      value--;
      count++;
}
```

5.14 Do-while Statement

```
do
{
        statement
} while (expression);
```

"Do-while" will first execute the statement and then check the expression. If the expression remains true, then it proceeds to the next statement until the expression becomes FALSE.

Example:

```
do {
         value --;
         count++;
} while (value != 0);
```

5.15 For Statement

```
for (expr1; expr2; expr3) statement;
```

"For" statement is equivalent to the following statement**:**

```
expr1;
while (expr2)
{
        statement;
        expr3;
}
```

"expr1" is executed first. Normally "expr1" will be the initial condition. "While" statement is executed in the same manner.

Example:

```
for (i = 0; i < 10; i++)
{
         value = value + i;
}
```

5.16 Break and Continue Statements

```
break;
continue;
```

The “break” statement exits from the innermost loop or switch block. The “continue” statement on the other hand will skip the remaining part of the loop and jump to the next iteration of the loop. “Continue” is useful in loop statements but it cannot be used in switch loops.

Example:

```
break exampl see switch.


for (i = 0; i < 10; i++)
{
         flag = indata(port);
         if (flag == 0) continue;
         outdata(port);
}
```

5.17 Goto Statement

```
goto label;
    …
label: …
```

“goto” statement is used to jump to any place of a function. It is useful to skip from a deep loop.

Example:

```
for (i = 0; i < 10; i++)
         for (j = 0; j < 100; j++)
                  for (k = 0; k < 100; k++)
                  {
                  flag = crccheck(buffer);
                  if (flag != 0) goto error;
                  outbuf(buffer);
                  }
error:
         //clear up buffer;
```

5.18 Function

"Function" is the basic block of the C language. It includes function prototype and function definition.

5.18.1 Function Prototype

```
<return_type> < function_name>  (<parameter_list>);
```

A "function prototype" should be declared before the function can be called. It contains the return value, function name, and parameter types.

NOTE

- *The total parameters passed to a function should be a fixed number. The compiler does not support uncertain parameter_list.*
- *Recursive functions are not supported in the compiler.*
- *Do not use "struct" or "union" as parameter for function.*
- *Function pointer is not supported.*
- *Bit data type cannot be used as a return value.*
- *For reduced RAM bank wastage, it is suggest that you use global variable in function, instead of using argument.*

Example (Function Prototype):

```
unsigned char sum(unsigned char a,unsigned char b);
…
```

5.18.2 Function Definition

```
<return_type> < function_name>  (<parameter_list>)
{
    statements
}
```

Example (Function Content):

```
unsigned char sum(unsigned char a,unsigned char b)
{
        return (a+b);
}
```

Chapter 6

C Control Hardware Related Programming

6.1 Register Page (rpage)

<variable name> @<address>[: rpage <register page number >];

The data type is used to declare a variable at a certain register page. Users have to declare clearly which register page (including rpage 0) the variable is declared with.

NOTE

- *When a variable is declared as "rpage," it cannot be declared as "bank," "iopage," or "indir" at the same time.*
- *Only global variable can be declared as "rpage" data type.*
- *When an MCU has "rpage 0" only, <register page number> still needs to be assigned.*
- *Some MCUs use "register bank" instead of "register page" under special register. When this condition occurs, keep on declaring in "rpage" just like in other cases.*
- *Usually common Registers 0x10~0x1F are reserve for C compiler. However, you may use and declare a variable in 0x10~0x1F with "rpage 0" if C Compiler is not using the registers. For example, you declare unsigned "int uiR16 @ 0x16: rpage 0". It is okay if C compiler is not using Register 0x16. But if C Compiler uses 0x16, it will report an error and notifies you how many continuous common registers space are needed.*

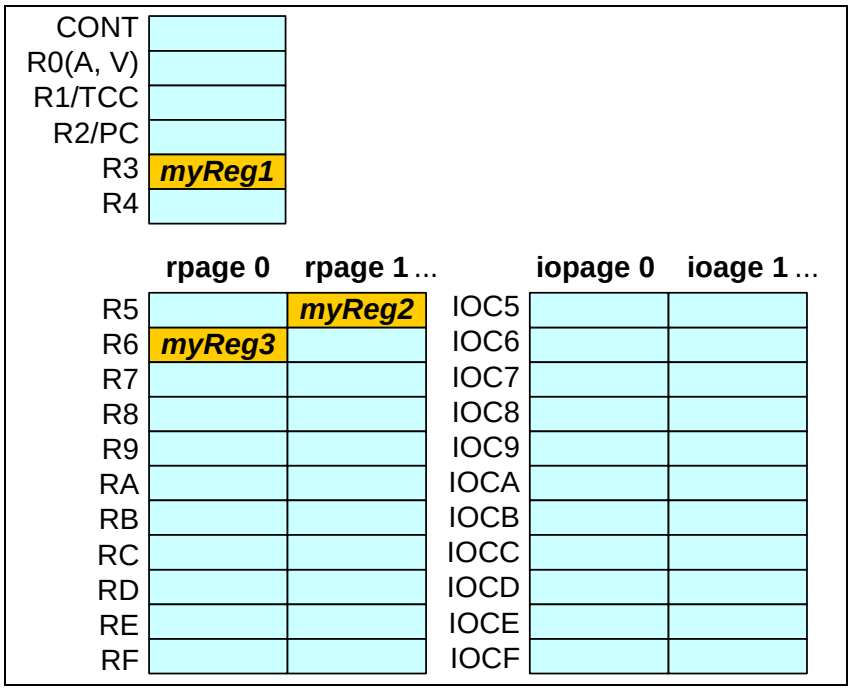
Example:

```
unsigned int myReg1 @0x03: rpage 0;
        // myReg1 is at address 0x03 of register page 0
        // Although the specific register only have one register
        // page,the register page number cannot be ignored.

unsigned int myReg2 @0x05: rpage 1;
        // myTest is at address 0x05 of register page 1
        // If the specific register have more than one register
        // page, user should point out in which register page
        // the variable is located.

struct st
{
```

```
    unsigned int b0:1;
    unsigned int b1:1;
    unsigned int b2:1;
    unsigned int b3:1;
    unsigned int b4:1;
    unsigned int b5:1;
    unsigned int b6:1;
    unsigned int b7:1;
};
struct st myReg3@0x06: rpage 0;
```



Declare variables in common registers 0x10~0x1F

Example:

```
unsigned int uiR12 @ 0x12: rpage 0; //declare uiR12 at common
                                       register 0x12
unsigned int uiRDD @ 0x16: rpage 0;
unsigned int uiR17 @ 0x17: rpage 0;
unsigned int uiR18 @ 0x18: rpage 0;
unsigned int uiR19 @ 0x19: rpage 0;
unsigned int uiR1A @ 0x1A: rpage 0;
unsigned int uiR1B @ 0x1B: rpage 0;
unsigned int uiR1C @ 0x1C: rpage 0;
unsigned int uiR1D @ 0x1D: rpage 0;
```

6.2 I/O Control Page (iopage)

io <variable name> [@<address>[: iopage <io control page number>]];

Declare the variable at the register page position. You have to clearly declare at which "iopage" the "io" variable is located, in spite of the fact that there is only one "io" control page.

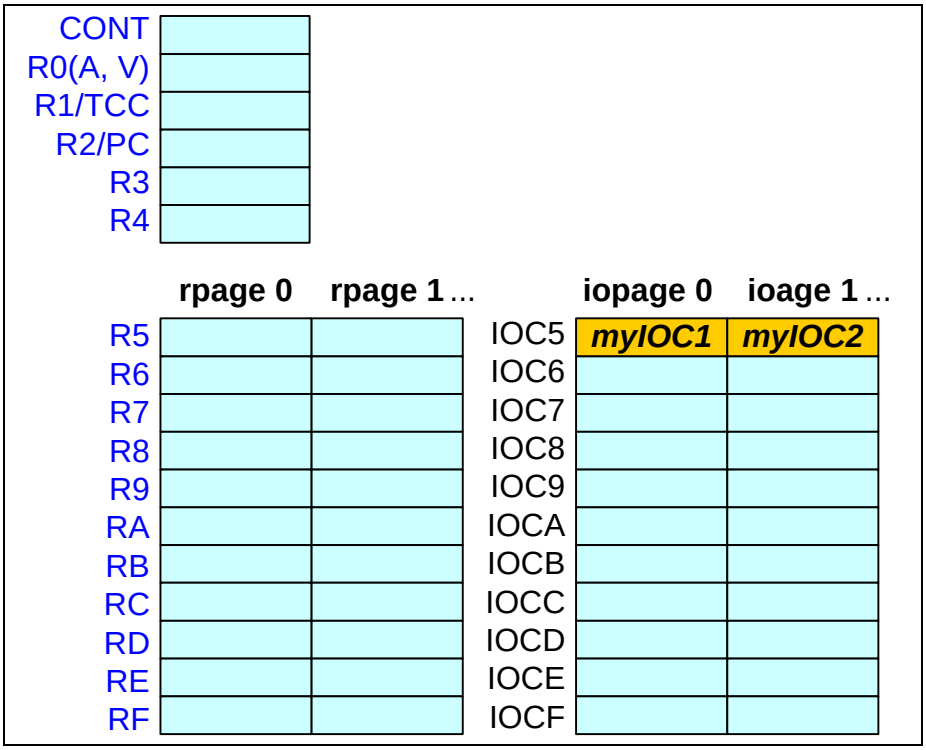
NOTE

- *If a variable is declared as "iopage," it cannot be declared as "bank," "rpage," or "IND" at the same time.*
- *Only global variable can be declared as "iopage" data type.*
- *When an MCU has "iopage 0" only, <io control page number> still needs to be assigned.*

Example:

```
io unsigned int myIOC1 @0x05: iopage 0;
        // myIOC1 is at address 0x05 of io control page 0

io unsigned int myIOC2 @0x05: iopage 1;
        // myIOC2 is at address 0x05 of io control page 1
```



6.3 Ram Bank

<variable name> [@<address>[: bank <bank number>]];

Declare the variable at which RAM bank it is located. The <bank number > has to be indicated, including the variable that is declared at Bank 0.

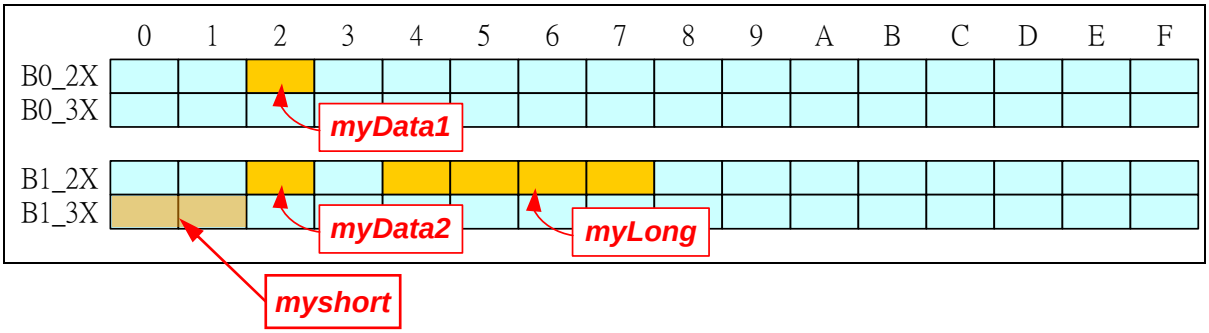
NOTE

- *If a variable is declared as "bank," it cannot be declared as "rpage," "iopage," or "indir" at the same time.*
- *Only global variable can be declared as "bank" data type.*

Example:

```
unsigned int myData1 @0x22: bank 0;
        // myData1 is at address 0x22 of ram bank 0
unsigned int myData2 @0x22: bank 1;
        // myData2 is at address 0x22 of ram bank 1
unsigned short myshort @0x20: bank 1;
        // myshort is at address 0x20 and 0x21 of ram bank 1
unsigned long myLong @0x24: bank 1;
        // myLong is at address 0x24~0x27 of ram bank 1
```

RAM Bank:



6.4 Bit Data Type

bit <variable name> [@<address> [@bitsequence] [: bank <bank number> / rpage <page number>]];

Bit data type occupies only one bit.

NOTE

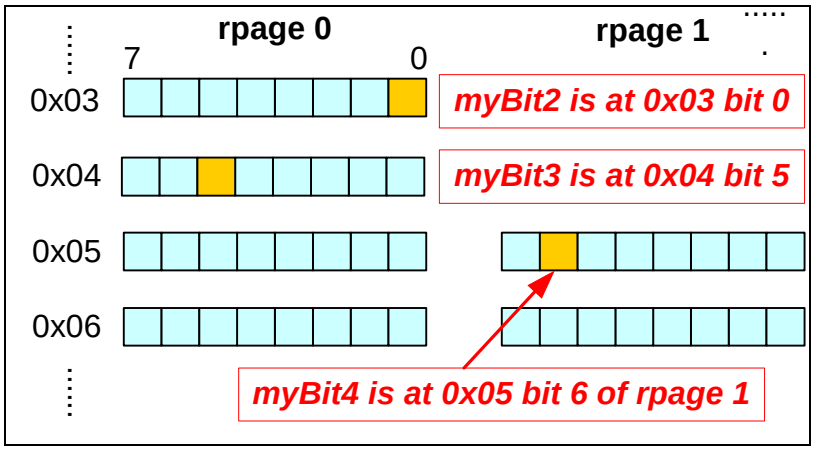
- *Bit data type cannot be used in "struct" and "union." It is recommended to use "bitfield" instead, such as:*

```
union mybit {
        unsigned int b0:1
        unsigned int b1:1
        unsigned int b2:1
        unsigned int b3:1
        unsigned int b4:1
        unsigned int b5:1
        unsigned int b6:1
        unsigned int b7:1
};
```

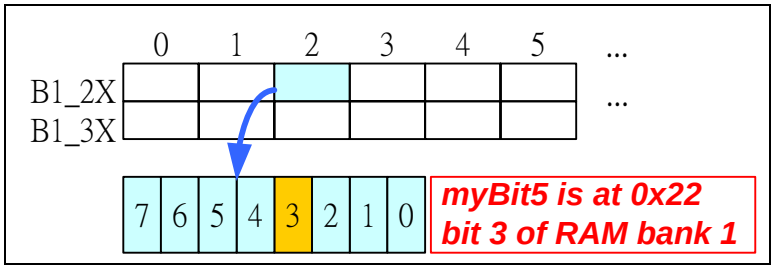
- *Bit data type cannot be used in function parameter.*
- *Bit data type cannot be used as a return value.*
- *Bit data type cannot be operated by arithmetic operator with other data type.*
- *Bit data type is not supported in the IO control register.*
- *Bit is a reserved word, so DO NOT use it as a name of "struct" or "union".*
- *Only global variable can be declared as "bit" data type.*
- *You cannot assign location for Bit data in local field. Otherwise compilation error will occur.*

Example:

```
bit myBit1;                     // location of myBit1 is assigned
                                // by linker
bit myBit2 @0x03 :rpage 0;
                                // if doesn't declare bit
                                // sequence,the default location
                                // is at bit 0.  Therefore myBit2
                                // is at bit 0 of 0x03 of rpage 0
bit myBit3 @0x04 @5: rpage 1; // myBit3 is at bit 5 of 0x04,
                                // rpage 1
bit myBit4 @0x05 @6: rpage 1; // myBit4 is at 0x05 bit 6 of
                                // rpage 1
bit myBit5 @0x22 @3: bank 1;  // myBit5 is at 0x22 bit 3 of ram
                                // bank 1
```



RAM Bank:



6.5 Data/LCD RAM Indirect Addressing

indir <variable name> [@<address>[: ind <ind number>]];

Declare the variable at which indirect data RAM or LCD ram is located. The <ind number > has to be indicated if address is assigned.

If the MCU has Data RAM, use “ind 0” (indirect RAM 0)

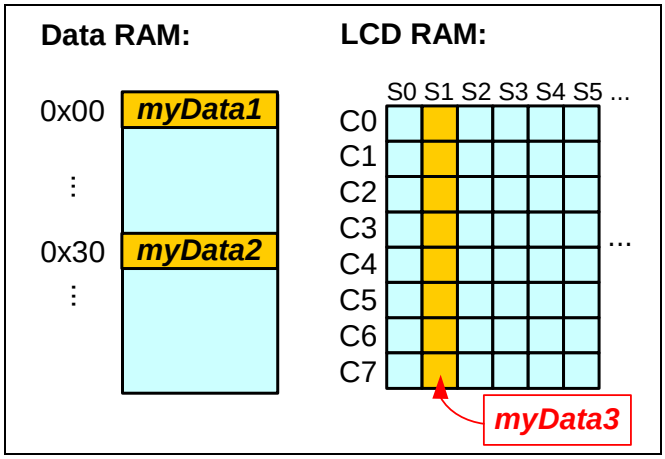
If the MCU has an LCD RAM, use “ind 1” (indirect RAM 1)

NOTE

- *If the specified MCU does not support IND bank, the compiler will generate an error message, e.g., “Symbol ‘WriteIND’ undefined”.*
- *Only global variable can be declared as “indir” data type.*
- *“Indir” data type does not support array or point variable.*

Example:

```
indir int nData1;//default is “ind 0”, so nData1 is at Data Ram
indir int nData2 @0x30: ind 0;
                 //nData2 is at Data Ram because “ind 0” is used
indir int nData3 @0x01: ind 1;
                 //nData3 is at LCD Ram because “ind 1” is used
```



6.6 Allocating C Function to Program ROM

```
<return value> <function name>(<parameter list>) @<address> [: page <page number>]
{
        ......
}
```

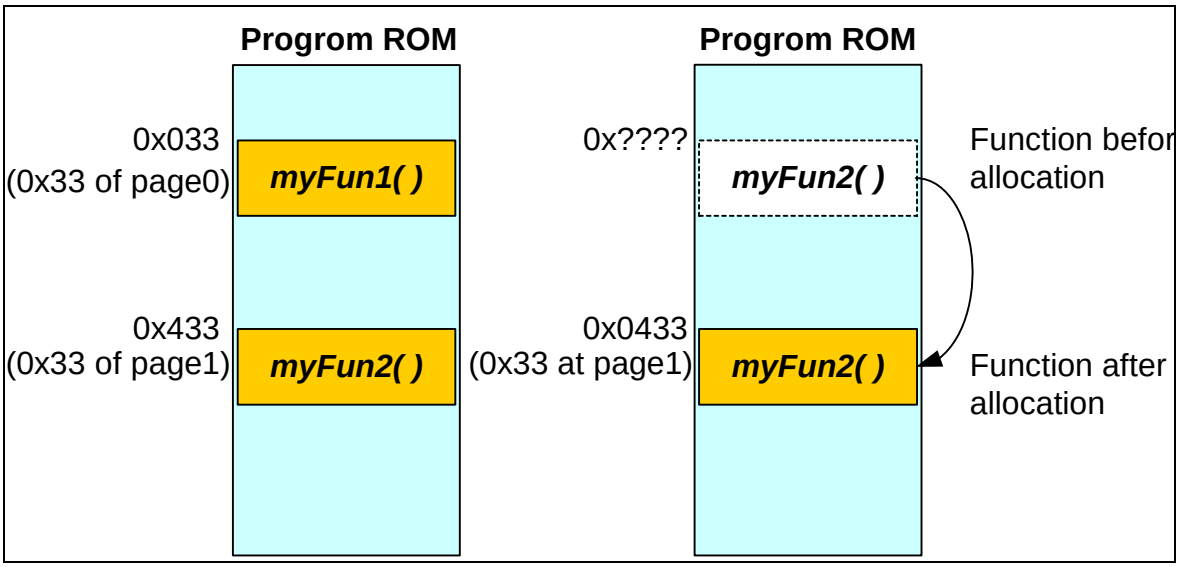
You can place C function at the dedicated address of the program ROM, and use "page" instruction to allocate which page in the program ROM you wish to assign.

NOTE

- *Only C functions can be declared as "page."*
- *Do not allocate the interrupt save procedure, nor the interrupt service routine at the dedicated address of the program ROM.*

Example:

```
void myFun1(int x, int y) @0x33
          // myFun1() is placed at 0x33 of ROM page 0 (default
          // page)
{
          ......
}
void myFun2(int x, int y) @0x33: page 1
          //myFun2() is placed at 0x33 of ROM page 1
{
          ......
}
```



6.7 Putting Data in ROM

const <variable name>;

Some data cannot be altered during program execution. Hence, you need to store such data into the program ROM to save limited RAM space. The Compiler uses the "TBL" instruction to incorporate such data into the program ROM.

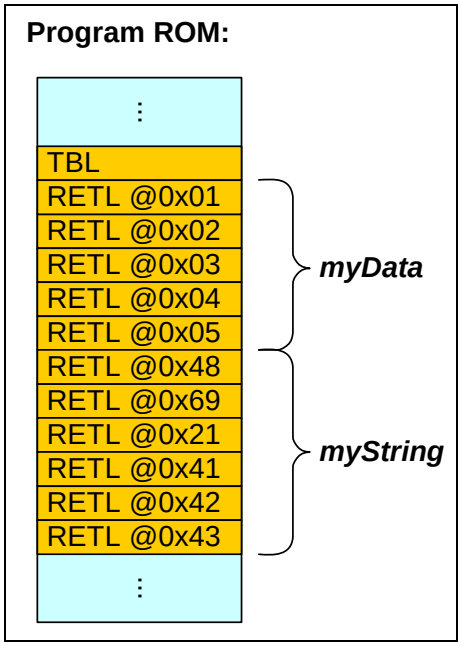
NOTE

- *Use constant data type to store data into the ROM.*
- *Only global variable can be declared as "const" data type.*
- *The maximum size of a constant array variable is 255 bytes.*

Example:

```
const int myData[] = {1, 2, 3, 4, 5};

const char myString[2][3] = {
       "Hi!",
       "ABC"
};
```



NOTE

If the specified MCU does not support TBL instruction, a page has only one ROM data area (below 0x100); otherwise a page has a maximum of two ROM data areas.

6.8 Inline Assembler

The compiler has an in line assembler which allows you to enhance the functionality of your program.

6.8.1 Reserved Word

The reserved words for the inline assembler are**:**

```
_asm
{
        …… //write assembly code here
}
```

All the assembly instructions (in upper or lower case) of the EM78 series are supported.

NOTE

- *You can declare variable in 0x10~0x1F when C Compiler is not using it. See Section 6.1, "Register Page (rpage)" for the details.*
- *If you have to switch "rpage," "iopage," or "bank" in the inline assembly, the original "rpage," "iopage," or "bank" must be saved at the beginning and restored at the end of the inline assembly program section. Refer to Example 1 in the next section (Section 6.8.2).*
- *If you use 0x10~0x1F in inline assembler, compiler will not report a warning or error message, but it may encounter some other unexpected errors.*
- *You cannot use "_ASM" (upper case) to replace "_asm" (lower case).*
- *If you use register address, io control address, or RAM bank address directly, compiler will not be able to recognize and differentiate between registers, io control, or RAM during assembly. You have to change the page & bank registers ("bs 0x03, 7 bs 0x3, 6" in the Example 1 below).*

  Before changing the page & bank registers, you need to save the procedure (see top section of the Example 1) and restore it at the end of the program (see bottom section of the Example 1).

6.8.2 Use of C Variable in the Inline Assembly

The Compiler allows you to access the C variable in the inline assembly as follows**:**

```
mov a, %<variable name>   //move variable value to ACC
mov a, @%<variable name>  //move address of variable to ACC
```

Example 1:

```
_asm
{
// Save procedure of rpage, iopage and bank register
        mov a,0x3
        mov %nbuf, a
        mov a, 0x04
        mov %nbuf+1, a
        bs 0x03, 7
        bs 0x03, 6    //Switch to other rpages
        ……
        //Restore procedure of rpage, iopage and bank
        mov a, %      //register
        mov 0x03, a
        mov a, %nbuf + 1
        mov 0x04, a
}
```

NOTE

If R3 Bit 6, Bit 7 are not page selection bits, or R4 is not a RAM selection register, refer and follow the procedure specified by the pertinent MCU Product Specification.

Example 2:

```
int temp;
temp=0x03;              //assume temp is at 0x21 of bank 0
_asm {mov a, %temp}     //move value 0x03 to ACC
_asm {mov a, @%temp}    //move address 0x21 to ACC
```

Example 3:

```
unsigned int temp_a @0x20: bank 0;
unsigned int temp_s @0x21: bank 0;
#define status 0x03;
void main()
{
_       asm
        {
        mov %temp_a, a   // ➔ mov 0x20, a
        mov a, status    // ➔ mov a, 0x03
        mov %temp_s, a   // ➔ mov 0x21, a
        }
}
```

6.9 Using Macro

You can use macro to control the MCU and shorten the program length.

NOTE

- *Use "#define" to declare a macro.*
- *Use "\" to join more than one line assembly codes.*
- *Do not add any character after "\" (even a block character is not allowed). Otherwise, an error will occur.*
- *Do not use constant as variable in macro. It will result to error.*

Example:

```
#define SetIO(portnum, var)_asm {mov a, @var} \
        _asm {iow portnum}

#define SetReg(reg, 3)_asm {mov a, @3} \
        _asm {iow portnum}
```

Macro "SetReg" will encounter error for using constant as argument.

6.10 Interrupt Routine

In TCC2, three actors have to be taken into account in handling interrupts**:**

1) **Interrupt Save Procedure:** the procedure to save some registers before executing a service routine. Now many new IC save important register like ACC, R3 or R4 as interrupt occurs and restore them before it quits interrupt service. Also TCC2 can save and restore these registers that eUIDE supplies. If not, eUIDE will supplies these form when a new one file added in a new project. If not only one interrupt vectors in one IC, users have to save register 0x2 (PC) to ACC. TCC2 use the value of PC to know which interrupt source.
2) **Interrupt Service Routine:** is the action to be taken for Interrupt. Don't care how many interrupt vectors, users have to write interrupts service code in it and use switch case or if-else if to know which interrupt vector source is if need.
3) **Global Interrupt Vector Index variable (IntVecIdx):** Users have to declare IntVecIdx as global integer like "extern int IntVecIdx;". IntVecIdx will occupy 0x10. So users can't declare or use 0x10 anywhere in project. Compiler doesn't tell you if you declare or use but it will run a wrong result.

6.10.1 Interrupt Save Procedure

```
void _intcall <function name>_I(void) @<interrupt vector address>: low_int <interrupt vector number>
```

Users have to write "MOV A,0x2" first in inline assembly code.

6.10.2 Interrupt Service Routine

```
void _intcall <function name>(void) @int
```

The <interrupt vector number> is skipped in TCC2 interrupt service routine and just one interrupt service routine function.

In example 1,not only one interrupt vector in EM78P510N, write interrupt service each case relative to interrupt vector. Users can mark case and interrupt save procedure if they aren't used.

In example 2, although just one interrupt vector in EM78569, users can't use 0x10 but don't declare IntVecIdx.

In example 3, in EM78567, users can't use 0x10 but don't declare IntVecIdx. Hardware don't save and restore and register.

Users have to note that some IC in restoring inline assembly use reti in the first line. It tells compiler to restore automatically from this line. Compiler will restore automatically after inline assembly if no reti. Reti have to put the first order in restoring inline assembly.

6.10.3 Reserved Common Registers Operation

Compiler save common registers (0x11~0x1F) compiler uses. Note the use of reti in restoring inline assembly in some IC.

Example 1:

EM78P510N: Hardware save and restore ACC, R3 and R5 in interrupt action. Not only one interrupt vector

```
extern int IntVecIdx; //occupied 0x10:rpage 0
…
void _intcall AllInt(void) @ int
{
  switch(IntVecIdx)
  {
    case 0x4:    //write interrupt vector 0x3 in this case
    break;

    case 0x7:    //write interrupt vector 0x6 in this case
    break;

    case 0xA:    //write interrupt vector 0x9 in this case
   break;

    case 0xD:    //write interrupt vector 0xC in this case
    break;

    case 0x10:   //write interrupt vector 0xF in this case
    break;

    case 0x13:   //write interrupt vector 0x12 in this case
    break;

    case 0x16:   //write interrupt vector 0x15 in this case
    break;

    case 0x19:   //write interrupt vector 0x18 in this case
    break;

    case 0x1C:   //write interrupt vector 0x1B in this case
    break;

  }
```

```
/* User also can use if-else if
  if(IntVecIdx==0x4)
  {
  }else if(IntVecIdx==0x7)
  {
  } else if(IntVecIdx==0xA)
  {
  } else if(IntVecIdx==0xD)
  {
  } else if(IntVecIdx==0x10)
  {
  } else if(IntVecIdx==0x13)
  {
  } else if(IntVecIdx==0x16)
  …

*/
}

void _intcall TCC_l(void) @ 0x03:low_int 0
{
 _asm{MOV A,0x2}; //using LJMP to interrupt service procedure
}
void _intcall ExtInt_l(void) @ 0x06:low_int 1
{
 _asm{MOV A,0x2};
}
void _intcall WatchTime_l(void) @ 0x09:low_int 2
{
 _asm{MOV A,0x2};
}
void _intcall Time1_l(void) @ 0x0C:low_int 3
{
 _asm{MOV A,0x2};
}
void _intcall Time2_l(void) @ 0x0F:low_int 4
{
 _asm{MOV A,0x2};
}
void _intcall ADC_l(void) @ 0x12:low_int 5
{
 _asm{MOV A,0x2};
}
void _intcall UART_l(void) @ 0x15:low_int 6
{
 _asm{MOV A,0x2};
}
void _intcall SPI_l(void) @ 0x18:low_int 7
{
 _asm{MOV A,0x2};
}
void _intcall LVD_l(void) @ 0x1B:low_int 8
{
 _asm{MOV A,0x2};
}
```

Example 2:

EM78569: Hardware save and restore ACC, R3 and R5 in interrupt action. eUIDE will tell Compiler to save and restore R4. One interrupt vector

```
void main()
{
  _asm{MOV A,@0x10  //enable bit 4 ,CONT
       CONTW
  }
}
void _intcall interrupt(void) @ int
{
 // Write your code (inline assembly or C) here
}
void _intcall interrupt_l(void) @ 0x08:low_int 0
{
  _asm{PAGE @0x0}   //it's dangerous to skip this line.
}
```

Example 3:

EM78567: Hardware doesn't save and restore any register and Compiler just can save and restore R3 and R4. One interrupt vector

```
void _intcall interrupt(void) @ int
{
 // Write your code (inline assembly or C) here

 //restore ACC and R5
   _asm {
      reti     //tell compiler to recover common registers here
      SWAPA 0X1E
      MOV 0X5,A
      SWAP 0X1F
      SWAPA 0X1F
    }
}
void _intcall interrupt_l(void) @ 0x08:low_int 0
{
  //save ACC and R5
  _asm {
     MOV 0X1F,A
     SWAPA 0X5
     MOV 0X1E,A
     PAGE @0X0
   }
```

Chapter 7

Quick Workout on Tiny C Compiler

7.1 Introduction

This chapter introduces you to a quick way of understanding and controlling C compiler. Please execute eUIDE by double clicking the eUIDE icon.

7.2 Create a New Project

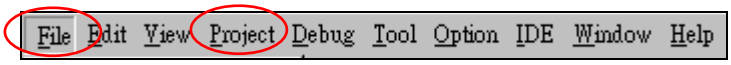


Figure 7-1 Main menu

Click New in File (Figure 7-2) or in Project (Figure 7-3) popup menu from menu (Figure 7-1) to call New dialog (Figure 7-4). Then we can select folder and type project name. Press OK button to create project. Don't type extend file name. eUIDE will add extend file name cpj for the project. In this case, we select EM78P468N for our target MCU during connection.

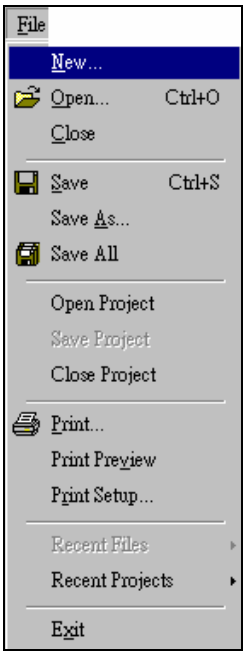


Figure 7-2 File Popup Menu

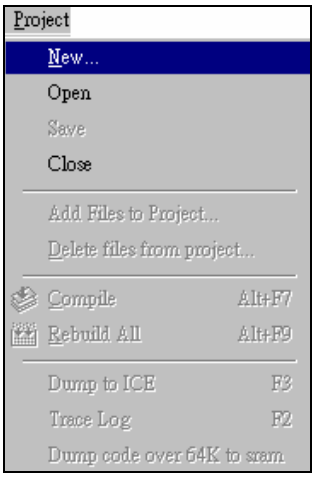


Figure 7-3 Project Popup Menu

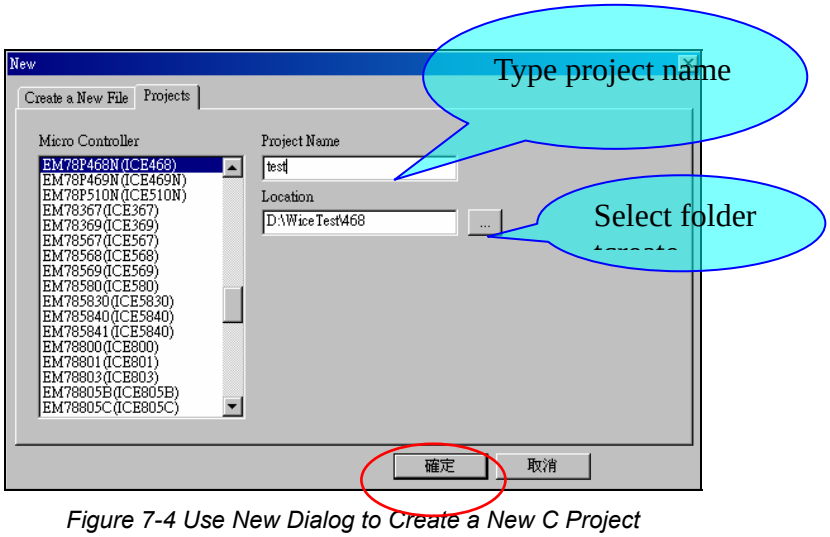


Figure 7-4 Use New Dialog to Create a New C Project

Now target MCU and the project name we create appear in Project window, Figure 7-5.

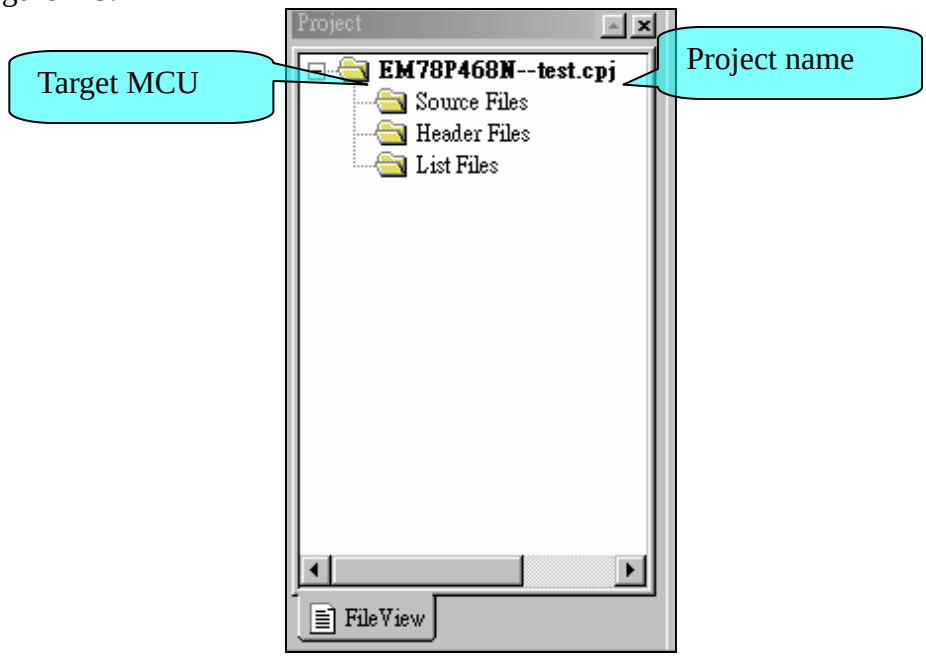


Figure 7-5 MCU and Project Name in Project Window

7.3 Add a New C File to the Project

Call New dialog like sec 8.1. This time New dialog appears and switch automatically to Create a New File sheet in New dialog, Figure 7-5. eUIDE will supplies basic Tiny C compiler main() file and interrupt save procedure and interrupt service routine frame if select extend file name as c in left combo box and leave Empty File check box uncheck. We'll introduce the frame later. Add new file to project box checked is default. Now type the file name for the first file to add into the project. Don't type extend file name. eUIDE will add extend file name automatically. Press OK button to add the c first file to the project.

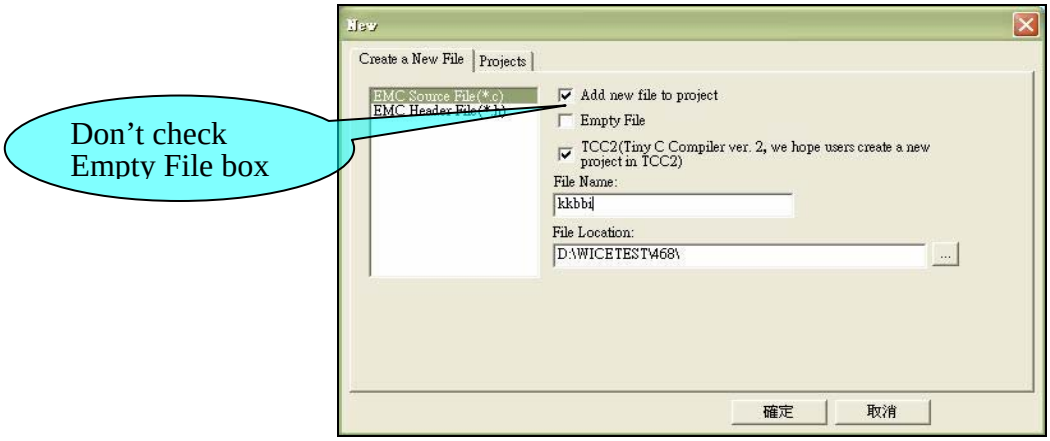


Figure 7-6 Add the First C File with Main() and Interrupt Save Procedure and Service Routine Frame

7.4 Add a New Second File or Head File to the Project

This time we have to check Empty File box in Create a New file sheet in New dialog.

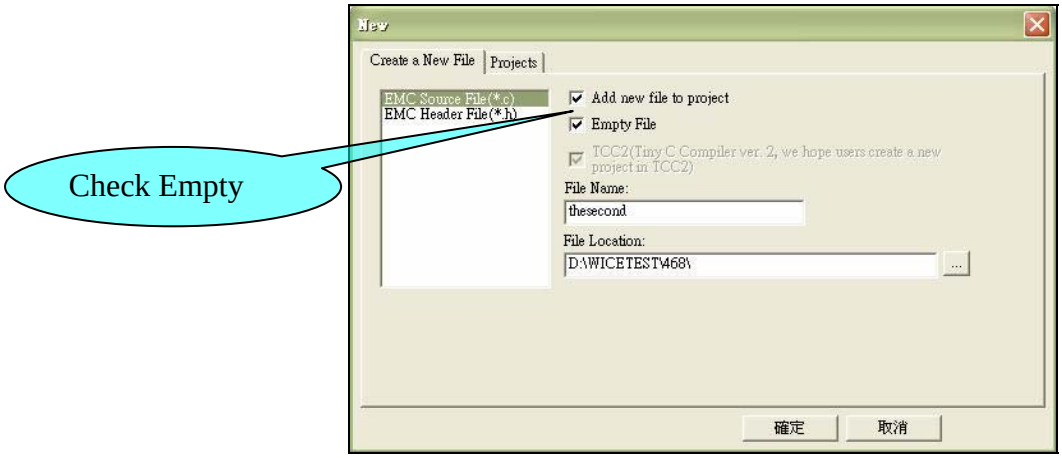


Figure 7-7 Add a New Second File or Head File to the Project

Type file name and press OK button eUIDE will create a new and empty c file. Head file will be added to project by empty and don't care about Empty File check box. Now we create a project with some c files and one head file, such as Figure 7-7.

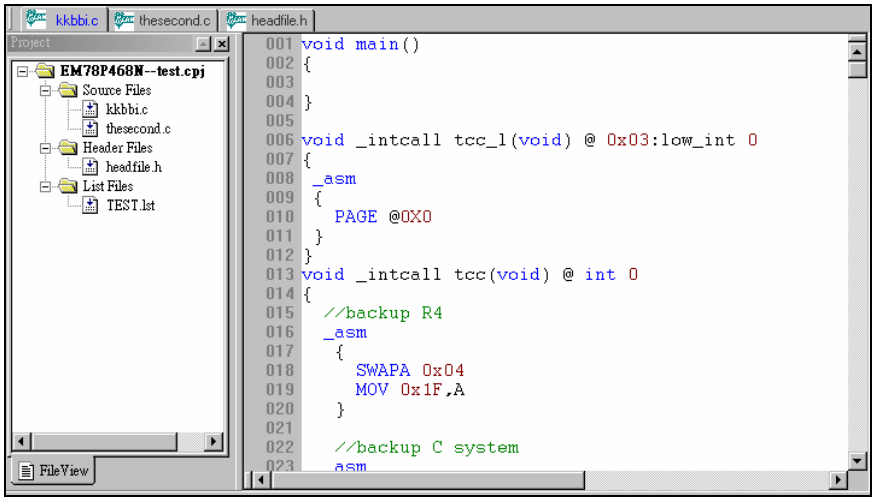


Figure 7-8 Created Project with Some C Files and One Head File

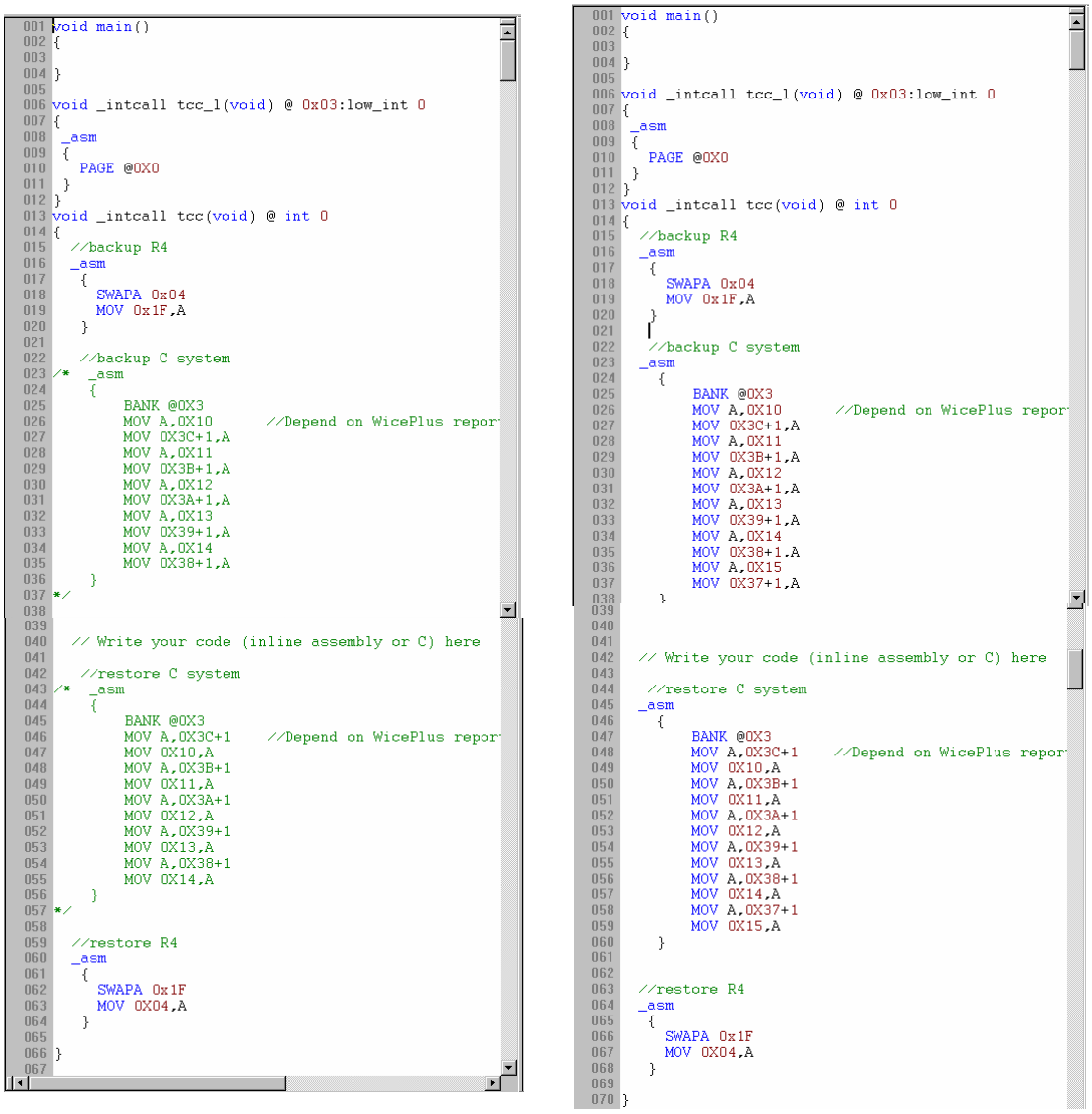
7.5 The Main() and Interrupt Save and Service Frame that eUIDE Supplies

Figure 7-8 shows part of the frame. Just only one main() function can exist in C project.

Usually users don't need to modify save and restore about R4, R3, ACC or R5 in the file that eUIDE supplies. But for safety reason we suggest users to check if these save or restore location are at the same ram bank. In this case, MCU EM78P468N hardware saves ACC and R3 before execute interrupt service routine and restore these two registers when interrupt service routine is completely. (Refer to *EM78P468 8-Bit Microcontroller Product Specification*). So we just have to save and restore R4 into 0x1F. For 0x1F is common register and belong to global field, we don't need to switch ram bank. We use swap instruction to avoid flag bit influenced. If we save R4 to 0x3F in ram bank 3, we have to restore 0x3F in ram bank 3 to R4. If we backup incorrectly, MCU may execute wrong PC (program counter) because R4 in EM78P468N control program page. We use inline assembly to do save and restore process (refer to Section 6.8, *Inline Assembler*). Save and restore ACC, R3, R4 or R5 we call save and restore MCU hardware.

In addition to save and restore MCU hardware, some of cases we have to save and restore C system. Some calculations outside interrupt service routine in C compiler don't only use ACC or the declared register or ram but some of common registers 0x10~0x1F. And in interrupt service routine also uses these common registers. We have to confirm these common registers' value are the same before running into interrupt save procedure and leaving interrupt service routine. So we have to save and restore these common registers correctly. C compiler will tell us after compiling succeed. Please see Sec. 8. 5. In the file frame, we suppose 0x10~0x14 common registers have to save and restore. Please remark these two backup C system and restore C system inline assembler code if need. If C compiler tell us more common registers over 0x10~0x14 have to save and restore, just add code like MOV A, 0x15 and MOV 0x37+1, A after backup 0x14 in backup C system inline assembler and MOV A, 0x37+1 and MOV 0x15, A after restore 0x14 in restore C system inline assembler.

Users write interrupt service code between backup C system and restore C system, such as //Write your code (inline assembly or C) here in the case.



```
001 void main()
002 {
003
004 }
005
006 void _intcall tcc_1(void) @ 0x03:low_int 0
007 {
008  _asm
009  {
010    PAGE @0X0
011  }
012 }
013 void _intcall tcc(void) @ int 0
014 {
015   //backup R4
016   _asm
017    {
018      SWAPA 0x04
019      MOV 0x1F,A
020    }
021
022    //backup C system
023 /*  _asm
024     {
025         BANK @0X3
026         MOV A,0X10      //Depend on WicePlus repor
027         MOV 0X3C+1,A
028         MOV A,0X11
029         MOV 0X3B+1,A
030         MOV A,0X12
031         MOV 0X3A+1,A
032         MOV A,0X13
033         MOV 0X39+1,A
034         MOV A,0X14
035         MOV 0X38+1,A
036     }
037 */
038
039
040   // Write your code (inline assembly or C) here
041
042    //restore C system
043 /*  _asm
044     {
045         BANK @0X3
046         MOV A,0X3C+1    //Depend on WicePlus repor
047         MOV 0X10,A
048         MOV A,0X3B+1
049         MOV 0X11,A
050         MOV A,0X3A+1
051         MOV 0X12,A
052         MOV A,0X39+1
053         MOV 0X13,A
054         MOV A,0X38+1
055         MOV 0X14,A
056     }
057 */
058
059   //restore R4
060   _asm
061    {
062      SWAPA 0x1F
063      MOV 0X04,A
064    }
065
066 }
067
```

```
001 void main()
002 {
003
004 }
005
006 void _intcall tcc_1(void) @ 0x03:low_int 0
007 {
008  _asm
009  {
010    PAGE @0X0
011  }
012 }
013 void _intcall tcc(void) @ int 0
014 {
015   //backup R4
016   _asm
017    {
018      SWAPA 0x04
019      MOV 0x1F,A
020    }
021
022    //backup C system
023   _asm
024     {
025         BANK @0X3
026         MOV A,0X10      //Depend on WicePlus repor
027         MOV 0X3C+1,A
028         MOV A,0X11
029         MOV 0X3B+1,A
030         MOV A,0X12
031         MOV 0X3A+1,A
032         MOV A,0X13
033         MOV 0X39+1,A
034         MOV A,0X14
035         MOV 0X38+1,A
036         MOV A,0X15
037         MOV 0X37+1,A
038     }
039
040
041
042   // Write your code (inline assembly or C) here
043
044    //restore C system
045   _asm
046     {
047         BANK @0X3
048         MOV A,0X3C+1    //Depend on WicePlus repor
049         MOV 0X10,A
050         MOV A,0X3B+1
051         MOV 0X11,A
052         MOV A,0X3A+1
053         MOV 0X12,A
054         MOV A,0X39+1
055         MOV 0X13,A
056         MOV A,0X38+1
057         MOV 0X14,A
058         MOV A,0X37+1
059         MOV 0X15,A
060     }
061
062
063   //restore R4
064   _asm
065    {
066      SWAPA 0x1F
067      MOV 0X04,A
068    }
069
070 }
```

Figure 7-9 eUIDE Supplies C Main and Interrupt Frame (Left) and Remark and Add 0x15 Save and Restore (Right).

7.6 Developing Your Project, Especially About Interrupt

Now we can write our c code to develop our product. In the case we wake up MCU by port 6 (Users can download the example from our Web http://www.emc.com.tw/database/Sa2/Gp/Dt/EM78P468C_Example.zip, file Emp7p468_pin wakeup.c) with Counter 1 underflow interrupt with interrupt vector 0xC. So we write some initial code and our interrupt service code like Figure 7-9 and Figure 7-10. Now we active one of c files and use Alt+F7 to compile the file to see if there is errors in the file. Alt+F7 just compile the active file. During developing we use Alt+F7 to compile c file one by one to save compiling time. After no errors in all c files in the project occur we use rebuild All (Alt+F9) to compile and link our object code and create execute file (cds). If rebuild All succeeds C Compiler will report many important and useful messages in Information sheet in Output Window. Figure 7-11 shows them.

```
005 #define ENI()   _asm{eni}
006 #define SLEP()  _asm{slep}
007 #define NOP()   _asm{nop}
008 #define uchar  unsigned char
009 void main()
010 {
011   ctest1();
012   while(1)
013   {};
014 }
015
016 void ctest1()
017 {
018   uchar temp;
019   abc=0x23;
020   def=0x39;
021
022   P6CR=0x80;                                    //
023   P6PH=0x80;                                    //
024   WDTCR=0x00;                          //wdt disa
025   temp=PORT6;                          //read por
026   ENI();
027   WUCR=0x0d;                                    //
028   IMR=0x80;                                     //
029 // SLEP();
030
031   ctest2();
032 }
```

Figure 7-10 Write Initial Code

Write interrupt service code here

```
225 void _intcall counter1_1(void) @ 0x0C:low_int 3
226 {
227  _asm
228  {
229    PAGE @0X0
230  }
231 }
232
233 void _intcall counter1(void) @ int 3
234 {
235   //backup R4
236   _asm
237    {
238      SWAPA 0x04
239      MOV 0x1F,A
240    }
241   //backup C system
242 /*  _asm
243     {
244         BANK @0X3
245         MOV A,0X10       //Depend on WicePlus repor
246         MOV 0X3C+1,A
247         MOV A,0X11
248         MOV 0X3B+1,A
249         MOV A,0X12
250         MOV 0X3A+1,A
251         MOV A,0X13
252         MOV 0X39+1,A
253         MOV A,0X14
254         MOV 0X38+1,A
255     }
256 */
257    // Write your code (inline assembly or C) here
258
259     ISR&=0xf7;
260     R57=!R57;
261   //restore C system
262 /*  _asm
263     {
264         BANK @0X3
265         MOV A,0X3C+1     //Depend on WicePlus repor
266         MOV 0X10,A
267         MOV A,0X3B+1
268         MOV 0X11,A
269         MOV A,0X3A+1
270         MOV 0X12,A
271         MOV A,0X39+1
272         MOV 0X13,A
273         MOV A,0X38+1
274         MOV 0X14,A
275     }
276 */
277
278   //restore R4
279   _asm
280    {
281      SWAPA 0x1F
282      MOV 0X04,A
283    }
284
285 }
```

Figure 7-11 Write Interrupt Service Code in Counter 1 Interrupt (Right)

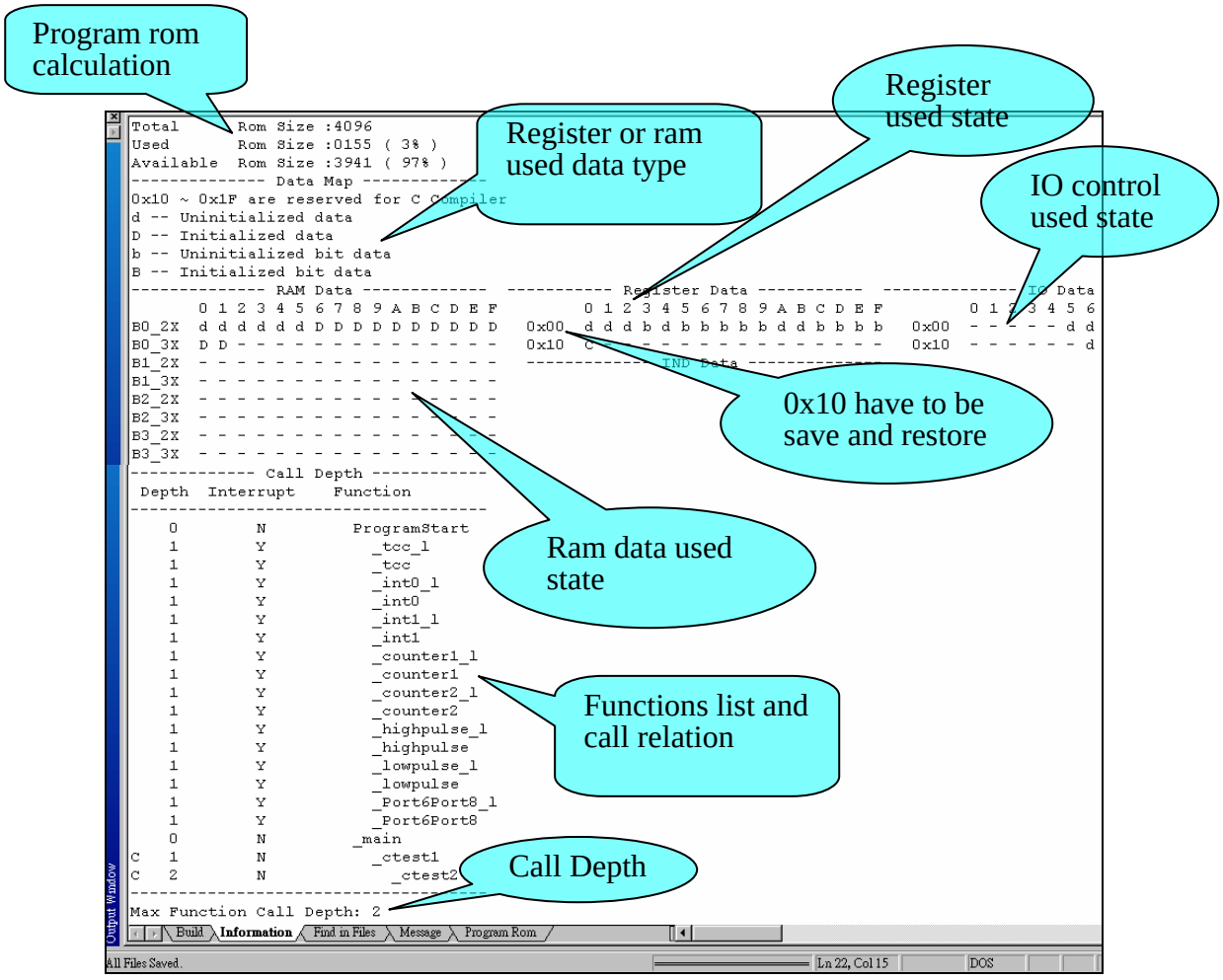


Figure 7-12 Useful Messages after Rebuild All Successfully

Figure 7-11 tells us which common registers we must save and restore. C char is located at 0x10. So we modify C system save and restore like Figure 7-12.

```
233 void _intcall counter1(void) @ int 3
234 {
235   //backup R4
236   _asm
237    {
238      SWAPA 0x04
239      MOV 0x1F,A
240    }
241    //backup C system
242   _asm
243     {
244        BANK @0X3
245        MOV A,0X10       //Depend on WicePlus repor
246        MOV 0X3C+1,A
247     }
248
249    // Write your code (inline assembly or C) here
250
251      ISR&=0xf7;
252      R57=!R57;
253    //restore C system
254   _asm
255     {
256        BANK @0X3
257        MOV A,0X3C+1     //Depend on WicePlus repor
258        MOV 0X10,A
259     }
260
261   //restore R4
262   _asm
263    {
264      SWAPA 0x1F
265      MOV 0X04,A
266    }
267
268 }
```

Figure 7-13 Save and Restore C System Correctly

After save and restore hardware system and C system correctly, we can debug or run our project.

Chapter 8

Tips on C Compiler Debugging

8.1 Speed Up Debug

During in C environment, User can press Speed Up Debug button to improve debugging speed with Step Into (F7).

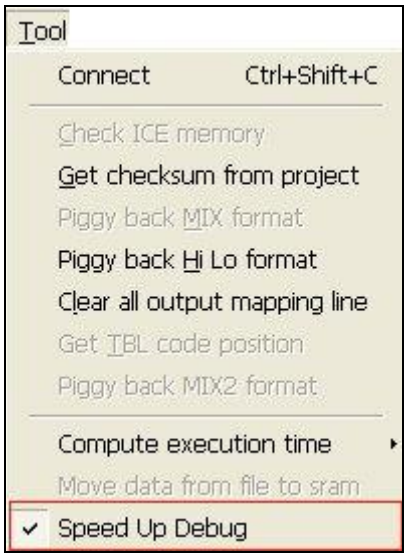


Figure 8-1 Speed Up Debug

If users don't press Speed Up Debug button, eUIDE runs Step Into function (F7) by program counter (PC) to program counter (PC) just as in assembly environment. But if Speed Up Debug pressed, eUIDE run to at first assembly code address corresponding to a C source line except some call function.

For example blow C source is at the left and the corresponding assembly codes are at the right. The next Step into (F7) will run from program counter 0x29 to 0x30 in assembly code without stop corresponding to line 2855 in C code.

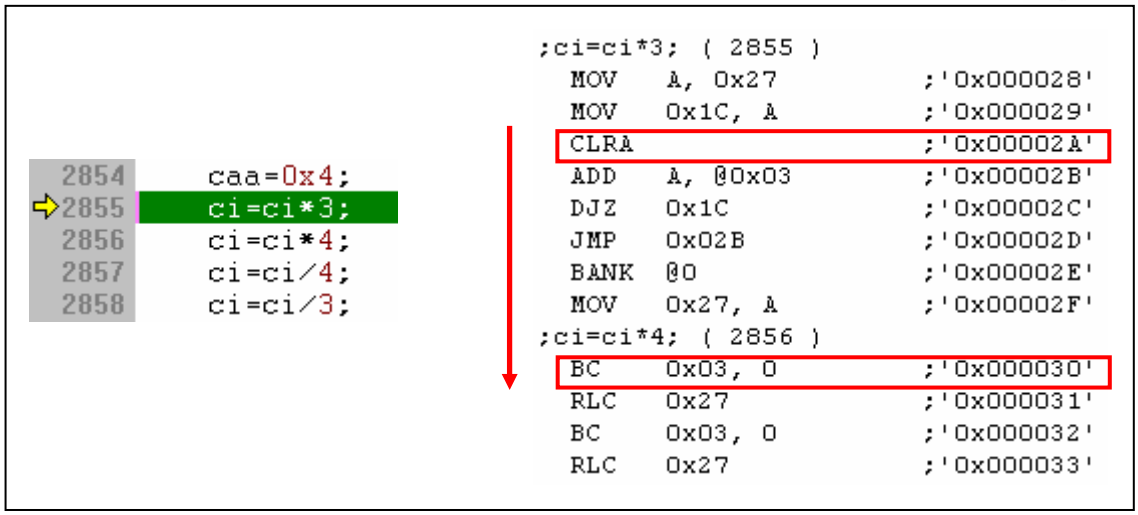


Figure 8-2 Speed Up Debug in C Step into (F7)Debug Mode

Please note some situations are right. For example, if the C code is ci++ in line 2856, so in Speed Up Debug after step into (F7) line 2588, the value of ci will increase 1.

8.2 View Corresponding Assembly Code in C Environment

After first dump code to ICE (F3) in C environment, users can press Assembly code like figure x. After that during debugging, users can see C and assembly code fully occupy in Edit window. Users have to active and set focus on C file if you want to press any button in menu.

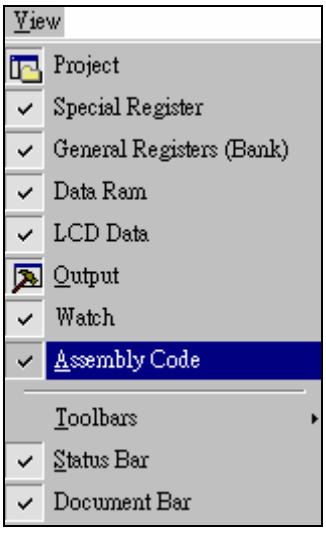


Figure 8-3 Press Assembly Code Button in View

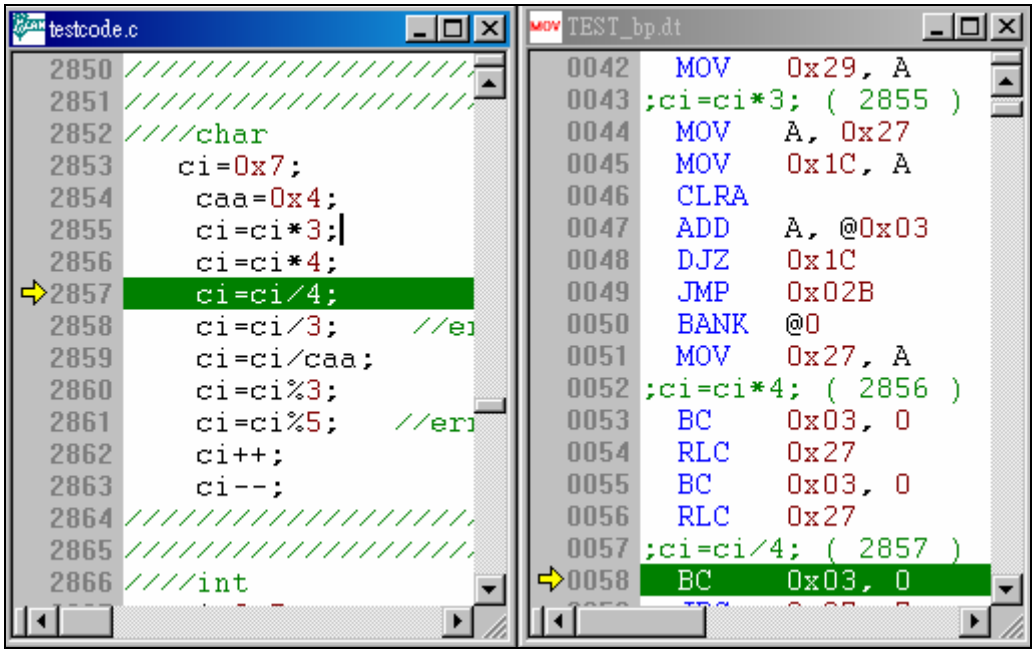
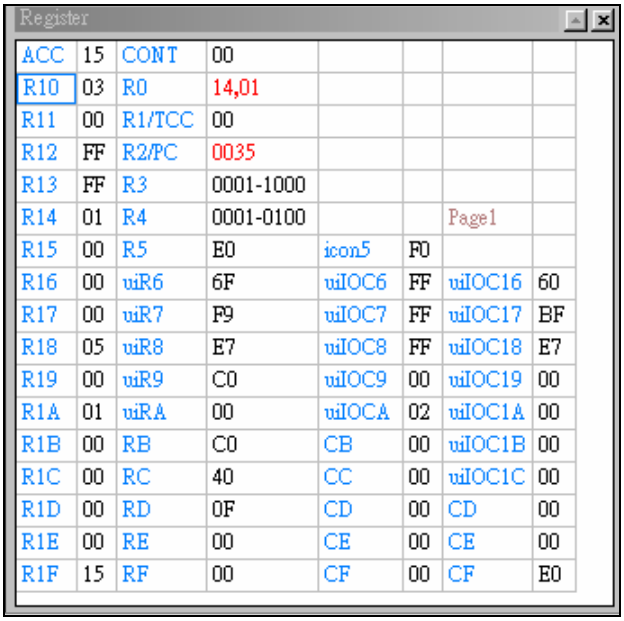


Figure 8-4 Assembly Code in C Debug Mode and Focus on C File.

Users have to active and set focus on assembly file if you want to press “StepInto”,“StepOver”,”StepOut”,”Go”,”FreeRun”,”Reset”,”GotoCursor”, “insert/Remove Breakpoint” buttons.

8.3 Watch Defined Variable in Register Window

Check Show defined label in Register window in view setting dialog after dumping, those variables declared in register field will show in register window. Registers name don’t show in default but users define.



8.4 Reduce Codes Size in Some Cases

Define signed variable and unsigned variable are not the same. Use unsigned variable will reduce program code size.

Appendix A

Assembly Error/ Warning Messages

A.1 Introduction

Error messages are categorized into the 4 classes which are Class M, Class A, Class L, and Class D. Any message of Class M pops out when main program is executed incorrectly. Any message of Class A is on the list if syntax errors, for instance, occur during assembling. Any message of Class L is showed in the list if there are any linking errors occurred during a project built or rebuilt. Any message of Class D describes the errors of debugging program.

A.2 Class M: Main Program Errors

1. "**error M001: Numbers of opening editor windows are over limitation.**"

   Reason: Too many opening windows to open one more.

   Solution: To close some unused editor windows.

2. "**error M002: Memory is not enough to allocate Editor Window.**"

   Reason: System has not enough memory to allocate editor window.

   Solution: To close some unused editor windows or unexecuted application programs in system.

3. "**error M003: File: [filename] is existed.**"

   Reason: The file name has been created already, so the same one cannot be created again.

   Solution: Rename and save again.

4. "**error M004: File: [filename] cannot be created.**"

   Reason: The application is notified that the file cannot be created by O.S.

   Solution: To check whether the disk is full or not and the system is stable or not.

5. "**error M005: One project has been opened.**"

   Reason: If one project has been opened in WICE, then users cannot open another project.

   Solution: To close the opening project, and then open another project.

6. "**error M006: Project: [filename] cannot be created.**"

   Reason: The application is notified that the project cannot be created by O.S.

   Solution: To check whether the project is empty or not.

7. "**error M007: The file [filename] has already existed in the project.**"

   Reason: The file has been included in the project.

   Solution: To stop trying to add the file into the project again.

8. "**error M008: File: [filename] cannot be saved.**"

   Reason: The file cannot be saved in the disk.

   Solution: To check whether the project is empty or not.

9. "**error M009: The project: [filename] is not the format of EMC project file.**"

   Reason: The content of the project is not the EMC project format.

   Solution: To create a new project, and add the file to project.

10. "**error M010: The file: [filename] does not exist.**"

    Reason: file cannot be found in the directory.

    Solution: To check whether the file exists or not.

11. "**error M011: The File: [filename] cannot be opened.**"

    Reason The file cannot be opened by O.S.

    Solution: To check whether the file is existed or not.

12. "**error M012: The file: [filename] exceeds [number]k of the max size [number]k.**"

    Reason: The file size is over the maximum size of buffer which editor is allocated.

    Solution: To divide the file into two or more.

13. "**error M013: The copy size: [number] exceeds [number]k of the max size [number]k.**"

    Reason: The size of copy is over the maximum size of copy buffer.

    Solution: To reduce content of the copy.

14. "**error M014: Memory cannot be allocated.**"

    Reason: System cannot allocate more memory to use.

    Solution: To close unused editor windows or unexecuted application programs in system.

15. "**error M015: Over 250 characters of a line.** "

    Reason: The maximum number of characters is 250 at a line in editor.

    Solution: To divide a line into two or more.

16. "**error M016: The extension name of the active file [filename] is not .dt or .asm.**"

    Reason: In this action, the extension name of the active file must be “DT” or “ASM”.

    Solution: To change the active file to another file with the extension name, “DT” or “ASM”.

17. "**error M017: No file to be assembled**."

    Reason: Cannot find a file to be assembled.

    Solution: To select a file to be assembled.

18. "**error M018: Project file must be created.**"

    Reason: No project is available.

    Solution: To open or create a project.

19. "**error M019: The editor number is over the max. editor number.**"

    Reason: The numbers of editors are limited.

    Solution: To close some of unused editor windows.

20. "**Error M020: No active editor window.**"

    Reason: The editor window is not selected now.

    Solution: To choose editor window.

21. "**Error M021: Text field must be input by characters.**"

    Reason: The text field can’t be empty.

    Solution: To input character.

A.3 Class A: Assembler Errors/Warnings

1. "**error A001: Cannot find the [filename] file.**"

   Reason: The file cannot be found in the directory.

   Solution: To check whether the file exists or not in the directory.

2. "**error A002: Main and subroutine programs cannot define [label name] local label.**"

   Reason: The local label (the preceding sign of label is “$”) cannot be defined in the main and subroutine programs.

   Solution: To remove the local label from the main or subroutine program.

3. "**error A003: The syntax form should be: operation [operand][,operand].**"
   Reason: The syntax form of statements must be “operation [operand][,operand]”.
   Solution: To correct error of the line.

4. "**error A004: The label [label name] is redefined.**"
   Reason: The label is defined more than once.
   Solution: To redefine the label name.

5. "**error A005: The EQU syntax is: label EQU operand.**"
   Reason: The EQU syntax is error.
   Solution: To correct error of the line.

6. "**error A006: The INCLUDE nest depth is over 256.**"
   Reason: The maximum depth of using “INCLUDE” is 256.
   Solution: To reduce the depth of “INCLUDE”.

7. "**error A007: The IF conditional expression is error.**"
   Reason: The expression of “IF conditional” is error.
   Solution: To correct the expression of “IF conditional”.

8. "**error A008: Attempt to divide by zero.**"
   Reason: The expression of number is divided by zero.
   Solution: To modify the expression.

9. "**error A009: The assembler does not support floating point.**"
   Reason: The assembler number system does not support floating point.
   Solution: To change the floating point to integer.

10. "**error A010: The symbol [symbol name] is not defined.**"
    Reason: The symbol is not defined.
    Solution: Please define the symbol.

11. "**error A011: The macro name [macro name] is redefined.**"
    Reason: The macro name is defined more than once.
    Solution: To change the macro name to isolate name.

12. "**error A012: The parameter name [parameter name] is the same as label.**"
    Reason: The parameter name is the same as label.
    Solution: To redefine the parameter name.

13. "**error A013: The parameter name [parameter name] is the same as another one.**"

    Reason: The parameter name must be different from another one in the macro definition.

    Solution: To redefine the parameter name.

14. "**error A014: The number of actual parameter does not match with formal parameter.**"

    Reason: The number of actual parameter is not matched with the number of formal parameter.

    Solution: To change the number of actual parameter or formal parameter.

15. "**error A015: The parameter number [number] does not exist.**"

    Reason: The position of actual parameter is not defined.

    Solution: To change the number of actual parameter.

16. "**error A016: The external symbol [symbol name] is the same name as defined label.**"

    Reason: The external symbol name is the same as internal defined label.

    Solution: To redefine the internal defined label.

17. "**error A017: Address of ORG is error.**"

    Reason: The address of ORG defined is error.

    Solution: To redefine the ORG address.

18. "**error A018: MACEXIT cannot be set outside macro.**"

    Reason: The “MACEXIT” instruction cannot be set outside the macro definition.

    Solution: To remove the “MACEXIT” instruction.

19. "**error A019: Parameter must be string variable.**"

    Reason: The formal parameter must be defined as string.

    Solution: To change the formal parameter to string.

20. "**error A020: Memory cannot be allocated.**"

    Reason: O.S. cannot allocate extra memory.

    Solution: To close the editor window which is not used currently or close application which is not used at present.

21. "**error A021: The source statements exceed [number] lines.**"

    Reason: The line number of source file is over the system default limit.

    Solution: To divide the program into two or more small programs.

22. "**error A022: The tree is error because of parser error.**"

    Reason: The syntax of the program line is error.

    Solution: To rewrite the program line.

23. "**error A023: [allocated memory type] memory fault when memory is allocated.**"

    Reason: O.S. cannot allocate extra memory.

    Solution: To close the editor window currently not used or close application currently not used.

24. "**error A024: Setting assembler variable must be integer value.**"

    Reason: The result of expression in the right side of “SET” instruction must be integer.

    Solution: To change the expression.

25. "**error A025: Address of ORG must be integer value.**"

    Reason: The address of ORG must be integer.

    Solution: To rewrite the address expression of “ORG” instruction.

26. "**error A026: The [number]address of PC is out of [number] ROM size.**"

    Reason: The address of program counter is over the program ROM size.

    Solution: To reduce the code size.

27. "**error A027: The assembler exceeds max. pass [number].**"

    Reason: The pass of assembler is over the maximum default pass.

    Solution: To check the assembler errors.

28. "**error A028: The operand [number] value dose not include the valid data.**"

    Reason: The operand value is invalid.

    Solution: To change the operand value.

29. "**error A029: The [number] address is conflicted.**"

    Reason: The program address is conflicted with another address.

    Solution: To use function of “[Edit]->[Find]” menu to find the address in the list file; then to update the address of the program position.

30. "**error A030: The file [file name] cannot be opened.**"

    Reason: The file cannot be opened.

    Solution: To check whether the file exists or not.

31. "**error A031: The configure file read error.**"
    Reason: The format of the configuration file is not correct.
    Solution: To setup WICE software again.

32. "**error A032: The file [file name] cannot be opened.**"
    Reason: The file cannot be opened.
    Solution: To check whether the file exists or not.

33. "**error A033: The macro is not defined.**"
    Reason: The macro name is not defined.
    Solution: To define a new macro or change a macro name.

34. "**error A034: The expression cannot be calculated.**"
    Reason: The format of the expression is not correct.
    Solution: To correct the format of the expression.

35. "**error A035: The operation [operation name] is not defined.**"
    Reason: The operation is not defined by WICE.
    Solution: To check the user menu of WICE.

36. "**error A036: The PUBLIC or EXTERN label [label name] must be address label.**"
    Reason: The label defined by “PUBLIC” or “EXTERN” instruction must be address label.
    Solution: To remove the public or extern variable.

37. "**error A037: The operand value cannot be calculated.**"
    Reason: The format of the operand expression is not correct.
    Solution: To correct the expression of the operand.

38. "**error A038: The symbol [symbol name] is not extern symbol.**"
    Reason: The symbol is not defined as internal or external.
    Solution: To define the symbol.

39. "**error A039: The reference number is over [number] limitation.**"
    Reason: The reference number in the “PUBLIC” or “EXTERN” instruction is over limitation.
    Solution: To divide the line into two or more.

40. "**error A040: The length of file name [file name] exceeds 256.**"
    Reason: The length of file name is over 256.
    Solution: To change the file name or directory.

41. "**Warning A050:The symbol length is exceed 32.**"
    Reason: The length of variable is over 32.
    Solution: To change the variable.

42. "**Warning A051:The number of message is over 500**"
    Reason: The number of message is over 500.
    Solution: To reduce the number of message.

43. "**Warning A052:The [number] to express [number]**"
    Reason: Transform negative number to hexadecimal.

44. "**Warning A053:The call/jmp address is not over page.**"
    Reason: The target address and “call” or “jmp” instruction is in the same page.

45. "**Warning A054: '%s' :unreferenced variable.**"
    Reason: The variable not use in the project.

A.4 Class L: Linker Error

1. "**error L001: Memory of [stack type] stack overflows.**"
   Reason: O.S. cannot allocate extra memory.
   Solution: To close unused editor window, or close unexecuted application.

2. "**error L002: The file [file name] cannot be found.**"
   Reason: The file cannot be found.
   Solution: To check whether the file exists or not.

3. "**error L003: The Object file format does not belong to EMC object file format.**"
   Reason: The object file format does not belong to EMC object file format.
   Solution: To reassemble the source file.

4. "**error L004: Symbol [symbol name] is not defined.**"
   Reason: The symbol is not defined.
   Solution: To define the symbol.

5. "**error L005: Public symbol [symbol name] is conflicted.**"
   Reason: The public symbol is defined more than once.
   Solution: To redefine the public symbol.

6. "**error L006: ROM address [number] is conflicted.**"

   Reason: The program address is conflicted with another.

   Solution: To use function of "[Edit]->[Find]" menu to find the address in the list file; then to update the address of the program position.

7. "**error L007: The file [file name] cannot be created.**"

   Reason: The file cannot be created.

   Solution: To check whether the disk is full or not and whether the system is stable or not.

8. "**error L008: Line [number]: The machine address [number] exceeds ROM size [number].**"

   Reason: The address of program address is over the ROM size.

   Solution: To reduce the program code.

9. "**error L009: No project file is active.**"

   Reason: No project is opened.

   Solution: To open or create a project.

10. "**error L010: No output window is found.**"

    Reason: Output window is not opened.

    Solution: To select [View]->[Output], and then an output window displays.

A.5 Class D: Debugger Error

1. "**error D001: The ICE memory is error .**"

   Reason: The SRAM in the ICE is error.

   Solution: To change the ICE SRAM.

2. "**error D002: CDS size = [number] does not match the ROM size = [number].**"

   Reason: The CDS size is not the same as the ROM size.

   Solution: To check project microcontroller name whether it is the same as WICE system micro controller name or not.

3. "**error D003: The project MCU type [type name] does not match [ICE name] ICE.**"

   Reason: The project micro controller name is not the same as WICE system microcontroller name.

   Solution: To create a new project or reconnect with another ICE.

4. "**error D004: The line of file does not transfer to machine address.**"
   Reason: The program line is unable to assemble.
   Solution: To check the syntax of the program line.

5. "**error D005: The breakpoint group number is over 64.**"
   Reason: The breakpoint group number is over 64.
   Solution: To remove the extra breakpoint groups.

6. "**error D006: The ICE is not connected to PC.**"
   Reason: The ICE is not connected to PC.
   Solution: To check power supply, ICE crystal, connected printer port, and so on.

7. "**error D007: The printer port is not connected.**"
   Reason: The printer port is not connected.
   Solution: To check power supply, connected parallel port, and so on.

8. "**error D008: The number is invalid.**"
   Reason: The number is invalid.
   Solution: To rewrite the number expression.

9. "**error D009: The number of messages is over [number].**"
   Reason: The number of messages is over limit in output window.
   Solution: To reduce the message is defined by user, example for "MESSAGE" instruction.

10. "**Warning D010: The address [number] does not match the source file.**"
    Reason: The program address does not match the source file line.
    Solution: To add the program source of the address, or to check whether the crystal is normal or not.

11. "**Warning D011: Memory checked is error.**"
    Reason: The ICE SRAM memory is error.
    Solution: To change the ICE SRAM, or to check whether the crystal is normal or not.

12. "**error D012: Syntax error.**"
    Reason: The syntax of source file has a severe error.
    Solution: To correct the source file.

13. "**error D013: Memory address [number] is error !**"
    Reason: The address of ICE SRAM is error.
    Solution: To change the ICE memory.

14. "**error D014: Can't find the breakpoint address of [program line].**"
    Reason: The breakpoint line does not have program address.
    Solution: To redefine the breakpoint.

15. "**error D015: The number of symbols is over [number].**"
    Reason: Too many symbols are defined.
    Solution: To divide the program into two or more small programs

16. "**error D016: Please dump program before adding label to watch**."
    Reason: To dump program before adding label to watch.
    Solution: To dump program to ICE.

17. "**error D017: Trace log is empty.**"
    Reason: No more data is found in trace log.
    Solution: Cannot execute the operation.

18. "**error D018: No trace item in trace log**"
    Reason: No more trace item is found in trace log.
    Solution: Cannot execute the operation.

Appendix B
C Conversion Table

B-1 Conversion between C and Assembly Codes

The assembly code was generated by the eUIDE.

Description	C Statement Example	Assembly Code	Conversion Rate (Compiler's Code Size / General User's Code Size * 100%)
Integer Variable	intVar1 = 0xFF;	MOV A, @0xFF MOV %intVar1, A	100% (2 / 2 * 100)
	intVar2 = intVar1;	MOV A, %intVar1 MOV %intVar2, A	100% (2 / 2 * 100)
Character Variable	charVar1 = 0xFF;	MOV A, @0xff MOV %charVar1, A	100% (2 / 2 * 100)
	charVar2 = intVar1;	MOV A, %charVar1 MOV %charVar2, A	100% (2 / 2 * 100)
Short Variable	shortVar1 = 0x1234;	MOV A, @0x34 MOV %shortVar1, A MOV a, @0x12 MOV %shortVar1+1, A	100% (4 / 4 * 100)
	shortVar2 = shortVar1;	MOV A, %shortVar1 MOV %shortVar2, A MOV A, %shortVar1+1 MOV %shortVar2+1, A	100% (4 / 4 * 100)
Long Variable	longVar1 = 0x123456;	MOV A, @0x56 MOV %longVar1, A MOV A, @0x34 MOV %longVar1+1, A MOV A, @0x12 MOV %longVar1+2, A	100% (6 / 6 * 100)
	longVar2 = longVar1	MOV A, %longVar1 MOV %longVar2, A MOV A, %longVar1+1 MOV %longVar2+1, A MOV A, %longVar1+2 MOV %longVar2+2, A	100% (6 / 6 * 100)

Description	C Statement Example	Assembly Code	Conversion Rate (Compiler's Code Size / General User's Code Size * 100%)
For loop	for (i = 0; i < 5; i++) { …… }	CLR %i JMP L2 L1: …… L2: INC %i MOV A, @0x05 SUB A, 0x14 JBS 0x03, 0 JMP L1	100% (7 / 7 * 100)
While statement	while (cnt != 1) { …… }	L1: … MOV A, %cnt XOR A, @0X01 JBS 0X03,2 JMP L1	100% (4 / 4 * 100)
Do-while statement	do { …… } while (cnt != 1);	L1: …… MOV A, %cnt MOV A, @0x01 XOR A, @0x01 JBS 0x03, 2 JMP L1	100% (4 / 4 * 100)
Do-while statement	do { Var_c2++; }while(-- var_c1);	L1: INC %var_c2; DJZ %var_c1; JMP L1	100%(3/3*100)
If-else statement	unsigned int cnt; if (cnt == 0) { …… } else if (cnt < 5) { …… } else { …… }	MOV A, %cnt JBS 0x03, 2 JMP L1 …… JMP ENDIF L1: MOV A,@0X05 SUB A, %cnt JBC 0x03, 0 JMP ENDIF …… JMP L2 L2: …… ENDIF:	100% (10 / 10 * 100)

Description	C Statement Example	Assembly Code	Conversion Rate (Compiler's Code Size / General User's Code Size * 100%)
Switch statement	unsigned int cnt; switch(cnt) { case 1: …… break; case 2: …… break; case 3: …… break; default: …… break; }	MOV A,%cnt MOV 0X14,A MOV A,0X14 XOR A,@0x01 JBC 0X03,2 JMP case 2 MOV A,0X14 XOR A,@0X02 JBC 0X03,2 JMP case 2 MOV A,0X14 XOR A,@0X3 JBC 0X03,2 JMP case 3 JMP default Case 1: JMP …ENDSWITCH Case 2: JMP ENDSWITCH Case 3: JMP ENDSWITCH Case 4: default ENDSWITCH	106% (18 / 17 * 100)
Function	main() { int i; i = fun(3); return; } int fun(int in) { return in+1; }	*; using EM78806B* MOV A, @0x03 BANK @0 MOV %in, A CALL FUN MOV A, 0x10 BANK @0 MOV %i, A RET FUN: MOV A, 0x14 BANK @0 MOV %temp1, A MOV A,%in MOV 0x14,A MOV 0X10,A MOV A,@0X01 ADD 0X10, A MOV A,%temp1 MOV 0X14,A RET	136% (19 / 14 * 100)

Description	C Statement Example	Assembly Code	Conversion Rate (Compiler's Code Size / General User's Code Size * 100%)
Const array	const int myConst[5] = {1, 2, 3, 4, 5}; main() { int i; i = myConst[3]; return; }	; using EM78569 MOV A, @0x3D MOV 0x19, A MOV A, @0x1F MOV 0x1A, A PAGE @0x0F CALL 0x280 PAGE @0x00 BC 0x04, 6 BC 0x04, 7 MOV 0x20, A … BC 0X03,0 RLCA 0x1A TBL … PAGE @0x0F JMP 0x2FE MOV A, 0x19 TBL …. RETL @0x01 RETL @0x02 RETL @0x03 RETL @0x04 RETL @0x05	162% (21 / 13 * 100)
Register page	unsigned int myR5P0 @0x05: rpage 0; unsigned int myR5P1 @0x05: rpage 1; unsigned int myR5P2 @0x05: rpage 2; myR5P0 = 0x12; myR5P1 = 0x34; myR5P2 = 0x56;	*; using EM78P468N* MOV A, @0x12 BS 0X03,6 MOV 0x05, A MOV A, @0x34 BS 0x03, 6 BC 0x03, 7 MOV 0x05, A MOV A, @0x56 BC 0x03, 6 BS 0x03, 7 MOV 0x05, A	100% (11 / 11 * 100)

Description	C Statement Example	Assembly Code	Conversion Rate (Compiler's Code Size / General User's Code Size * 100%)
I/O control page	io unsigned int myIO6P0 @0x06: rpage 0; io unsigned int myIO6P1 @0x06: rpage 1; io unsigned int myIO7P1 @0x07: rpage 1; myIO6P0 = 0x00; myIO6P1 = 0xFF; myIO7P1 = 0x55;	*; using EM78569* MOV A, @0x00 BC 0x03, 5 IOW 0x6 MOV A, @0xFF BS 0x03, 5 IOW 0x6 MOV A, @0x55 IOW 0x7	100% (8 / 8 * 100)
RAM bank	unsigned int myData1 @0x20: bank 0; unsigned int myData2 @0x21: bank 0; unsigned int myData3 @0x21: bank 1; myData1 = 1; myData2 = 2; myData3 = 3;	*; using EM78569* MOV A, @0x01 BC 0x04, 6 BC 0x04, 7 MOV 0x20, A MOV A, @0x02 MOV 0x21, A MOV A, @0x03 BS 0x04, 6 BC 0x04, 7 MOV 0x21, A	100% (10 / 10 * 100)
Bit data type	bit myB0R6P0 @0x06@0x00: rpage 0; bit myB2R6P0 @0x06@0x02: rpage 0; myB0R6P0 = 1; myB2R6P0 = myB0R6P0;	BS 0x06, 0 BC 0x06, 2 JBC 0x06, 0 BS 0x06, 2	133% (4 / 3 * 100)

Description	C Statement Example	Assembly Code	Conversion Rate (Compiler's Code Size / General User's Code Size * 100%)
Indirect addressing	indir unsigned myData1 @0x30: ind 0; indir unsigned myData2 @0x05: ind 1; myData1 = 0x55; myData2 = 0xAA;	*; using EM78806B* MOV A, @0x55 MOV 0x1B, A MOV A, @0x30 MOV 0x18, A MOV A, @0x00 MOV 0x19, A MOV A, @0x00 MOV 0x1A, A MOV A, 0x1B CALL INDIR MOV A, @0xAA MOV 0x1B, A MOV A, @0x05 MOV 0x18, A MOV A, @0x00 MOV 0x19, A MOV A, @0x01 MOV 0x1A, A MOV A, 0x1B CALL INDIR ……	146% (35 / 24 * 100)
		INDIR: BC 0x05, 0 MOV 0x1B, A MOV A, 0x1A JBS 0x03, 2 JMP 0x081 MOV A, 0x18 IOW 0x9 MOV A, 0x1B IOW 0xA RET LCDRAM: MOV A, 0x18 MOV 0x0A, A MOV A, 0x1B MOV 0x0B, A RET	

Description	C Statement Example	Assembly Code	Conversion Rate (Compiler's Code Size / General User's Code Size * 100%)
Bitwise operation (all variables are "unsigned int" data type)	f = e & d; (f = e ^ d;) (f = e | d;)	MOV A, %e AND A, %d (XOR A, %d) (OR A, %d) MOV %f, A	100% (3 / 3 * 100)
	f=~e;	COMA %e MOV %f, A	100%(2/2*100)
	f &=e; (f ^=e;) (f |= e)	MOV A, %e AND %f , A (XOR %f, A) (OR %f, A)	100%(2/2*100)
	f = e >> 1;	BC 0x03, 0 RRCA %e MOV %f, A	100% (3 / 3 * 100)
	f = e << 1;	BC 0x03, 0 RLCA %e MOV %f, A	100% (3 / 3*100)
	f>>=3;	BC 0x03, 0 RRC %f BC 0x03, 0 RRC %f BC 0x03, 0 RRC %f	100%(6/6*100)
	f<<=3	BC 0x03, 0 RLC %f BC 0x03, 0 RLC %f BC 0x03, 0 RLC %f	100%(6/6*100)
	f>>=4	SWAPA 0x06 AND A, @0x0F MOV 0x06, A	100%(3/3*100)
	f<<=4	SWAPA 0x06 AND A, @0xF0 MOV 0x06, A	100%(3/3*100)
	f>>=6;	SWAP 0x06 RRC 0x06 RRCA 0x06 AND A, @0x03 MOV 0x06, A	100%(5/5*100)
	f<<=6;	SWAP 0x06 RLC 0x06 RLCA 0x06 AND A, @0xC0 MOV 0x06, A	100%(5/5*100)

Description	C Statement Example	Assembly Code	Conversion Rate (Compiler's Code Size / General User's Code Size * 100%)
	f=(e<<5) | d;	MOV A, %e MOV 0x14, A SWAP 0x14 RLCA 0x14 AND A, @0xE0 OR A, %d MOV %f, A	100%(7/7*100)
	f=(f & const.1) | const. 2	MOV A, 0x06 AND A, const. 1 OR A, const. 2 MOV 0x06, A	100%(4/4*100)
Arithmetic expression (all variables are "int" data type)	f = e + d;	MOV A, %e ADD A, %d MOV %f, A	100% (3 / 3 * 100)
	f = e – d;	MOV A, %d SUB A, %e MOV %f, A	100%(3/3*100)
	f++;	INC %f	100% (1 / 1 * 100%)
	f—;	DEC %f	100% (1 / 1 * 100%)
	c = a * b;	MOV A, %a MOV 0X1C,A MOV A, %b MOV 0X18, A CLRA L1: ADD A, 0X1C DJZ 0X18 JMP L1 MOV %c, A	100%(9/9*100%)
	c = a / b;	MOV A, %a MOV 0x1C, A MOV A, %b CLR 0x18 L1: SUB 0x1C, A JBC 0x03, 0 INC 0x18 JBC 0x03, 0 JMP 0x3BB MOV A, 0x18 MOV %c, A	100%(11/11*100%)

Description	C Statement Example	Assembly Code	Conversion Rate (Compiler's Code Size / General User's Code Size * 100%)
Compound assignment (all variables are "int" data type)	f += e; (f -= e;) (f &= e) (f ^= e) (f |= e)	MOV A, %e ADD %f, A (SUB %f, A) (AND %f, A) (XOR %f, A) (OR %f, A)	100% (2 / 2 * 100%)
	f >>= 1;	BC 0x03,0 RRC %f	100% (2 / 2 * 100%)
	f <<= 1	BC 0x03,0 RLC %f	100% (2 / 2 * 100%)

Appendix C

Frequently Asked Questions (FAQ)

C.1 FAQ on Assembly

Q: ICE cannot connect to PC

A: Please check the following steps:

Step 1 Check the power supply with output around the range of 17~23 voltages.

Step 2 Check the printer cable between ICE and PC.

Step 3 Check the oscillator on the hardware.

Step 4 From the Option menu, choose ICE code option. Check the type of oscillator in the code option.

Step 5 If the problem still exists, then check again with another PC.

Step 6 Repeat from Step 1.

Step 7 If it is still not solved, please call EMC.

Q: The connection from ICE to PC is successful, but program dumping fails.

A: Please check the following steps:

Step 1 From the Option menu, choose ICE code option. To check the type of oscillator in the code option.

Step 2 Please check whether the type of microcontroller is the same as that in hardware.

Q：If ICE memory is checked, and the result of ICE memory is in error.

A: Please check the following steps:

Step 1 From the Option menu, choose ICE code option. To check the type of oscillator in the code option.

Step 2 Please check whether the type of microcontroller in software is the same as that in hardware.

Step 3 Please change ICE SRAM.

Q: If users try to reconnect the same ICE from PC, why does the execution become slower and slower ?

A: The timing of communication between ICE and PC is stretched if the connection is tried again and again.

Q: Why can't "Step Into" be executed?

A: Please check whether the type of microcontroller in software is the same as that in hardware.

Q: If source files are located in Novell file server, why is the program reassembled again at each execution time?

A: It is because of the time difference between PC and server. Therefore, it is better to locate all source files in one computer no matter that is network computer or local one.

C.2 FAQ on Tiny C Compiler

Q: What is the maximum number of the function parameters?

A: It depends on the RAM bank size (about 32 or 31 bytes).

Q: In a function, what is the maximum depth of the function call?

A: It depends on the hardware stack depth or size.

Q: What is the maximum array dimension as well as maximum array element?

A: It depends on the RAM bank size (about 32 or 31 bytes).

Q: Is there any error message when the code exceeds the ROM size?

A: Yes, the linker will report an allocation error.

Q: In a high level interrupt subroutine, can user allocate the address in the ROM? (e.g., using "page" data type, putting "_asm{ org xxx}" before a subroutine, etc.)

A: No! This may cause unpredictable error.

Q: Is "static" used in the same way as in ANSI C?

A: Yes.

Q: Is there any error message in case user defines too many variables in the "const" that exceeds the ROM space?

A: Yes, the linker will report an allocation error.

Q: How do I declare the variable in *.h file and using not only in one .c file?

A: For example, declare in *.h file like that:

```
extern io unsigned int DIRPORT6;
```

and you have to write like below just only one *.c file like that:

```
io unsigned int DIRPORT6  @0x06: iopage 0;
```

Q: Should I change any program page or bank?

A: If you just develop your program in C language, you don't have to change any program page, register page and ram bank, and so on. But if you use inline assembly in your program, you have to save and restore about page or bank.

Q: May I know how many stacks I have called?

A: Yes, in C developed environment, after compiling, user can know how many function call depth in Information, Output Window.

Q: Does C compiler just occupy 0x10~0x1F general purpose ram?

A: Well, almost C compiler just occupies 0x10~0x1F general purpose register. But If there are some arguments in call functions, compiler will use some others ram in 0x20~0x3F, bank 0 ~ bank 3. So, we suggest users use global variables to replace arguments in call function.

Users always have to note that there are some ram spaces used in interrupt save procedure and interrupt service procedure. If you don't use these ram space again.

Q: Does C compiler support this item, such as EM78P510N?

A: TCC2 supports all EM78 series IC except EM78x680, EM78x611.

Q: How do I use macro with variable?

A: for example:

```
io unsigned int P6CR@0x06:iopage 0;
…
#define set_output(port, bit)     _asm{ ior port};\
                                  _asm{ and a,@(~(1<<bit))};\
                                  _asm{ iow port}

…
set_output(%P6CR,0x04);
```

Q: Does C Compiler support all assembly instruction?

A: TCC2 supports all assembly instruction. But some instructions have to implement by the form of inline assembly. Such as MUL, TBRD, TBWD, and MOV R, R.

Q: How can clear all ram in all banks?

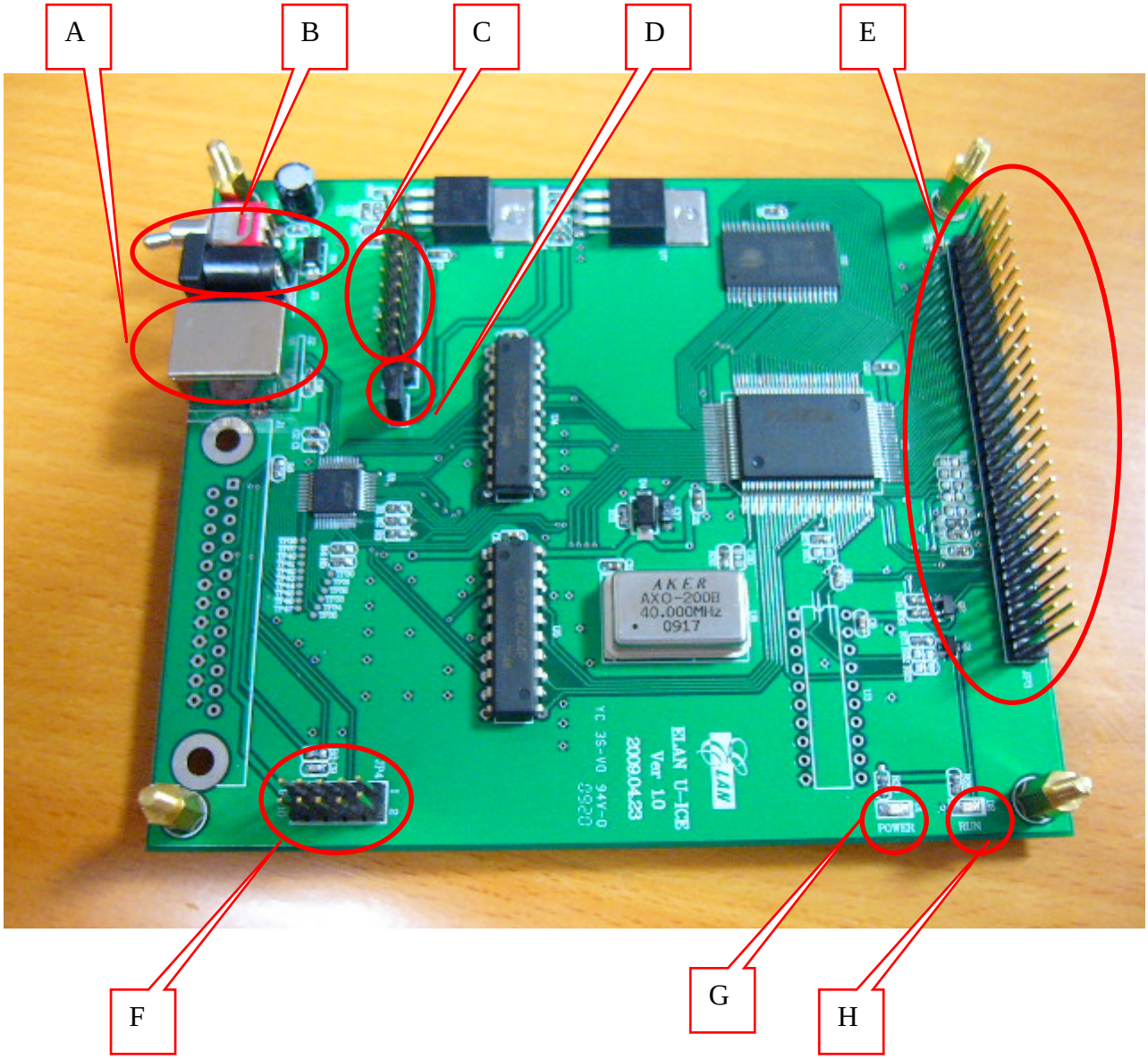
A: There are two key points in this issue. One is to keep bit 5, R4, as 1. The other is to switch bank correctly. But some IC uses a non-global register to switch ram bank, such as EM78P510N. So users have to know the peculiar to avoid fault. For example:

```
unsigned int iR0:0x0:rpage 0;
unsigned int RSR:0x4:rpage 0;
unsigned int BRSR:0x5:rpage 0;
unsigned int i:0x11:rpage 0;
…
for(i=0;i<0x8;i++)
{
   BRSR=i;
   For(RSR=0xFF;RSR>=0xE0;RSR--)
   {
      iR0=0;
    }
}
```

Appendix D

UICE Hardware Description

D.1 UICE and its Major Components/Functions



Description:

Symbol	Description
A	USB B type connector, this port can connect to PC.
B	DC power adaptor connector and switch. The power will support all ICE power.
C	ROMLESS program data bus connector. The connector will connect to ROMLESS board.
D	Power select jump. If short this jump, the system will work in 5V, if open this jump the system will support 3.3V.
E	ROMLESS program address bus and relative control signal connector. The connector will connect to ROMLESS board.
F	C8051 ISP port. This port can connect to C8051 ISP download cable to update firmware from PC at C8051 IDE tool.
G	System power LED display.
H	UICE free run LED display.

NOTE

1. *If you want to change U/ITxxx board (ROMLESS board) of UICE, the UICE power must be off and USB connector must be plug out firstly.*
2. *You must check to ensure target system is 3.3V or 5V application before turning on UICE power. If your target system is 3.3V application, the"JP2" (symbol D) on UICE PCB board must at open position before turning on UICE power. Otherwise, if your target system is 5V application, the "JP2" (symbol D) must at close position.*

Symbol D (JP2)	Description
Open	For 3.3V application
Close	For 5V application

D.2 Note for eUIDE and UIT660N

eUIIDE 1.00.13 later version can work on UIT660N, not work on IT660N

eUIIDE 1.00.12 before version can work on IT660N, not work on UIT660N