WicePlus C Compiler

for EM78 Series Microcontrollers

USER'S GUIDE

Doc. Version 2.1

ELAN MICROELECTRONICS CORP.

Mar 2007

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 (WicePlus) 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 Road 1
Hsinchu Science Park
Hsinchu, Taiwan 30077
Tel: +886 3 563-9977
Fax: +886 3 563-9966
http://www.emc.com.tw

Hong Kong:

Elan (HK) Microelectronics Corporation, Ltd.

Rm. 1005B, 10/F Empire Centre
68 Mody Road, Tsimshatsui
Kowloon , HONG KONG
Tel: +852 2723-3376
Fax: +852 2723-7780
elanhk@emc.com.hk

USA:

Elan Information Technology Group

1821 Saratoga Ave., Suite 250
Saratoga, CA 95070
USA
Tel: +1 408 366-8223
Fax: +1 408 366-8220

Shenzhen:

Elan Microelectronics Shenzhen, Ltd.

SSMEC Bldg., 3F, Gaoxin S. Ave.
Shenzhen Hi-Tech Industrial Park
Shenzhen, Guandong, CHINA
Tel: +86 755 2601-0565
Fax: +86 755 2601-0500

Shanghai:

Elan Microelectronics Shanghai Corporation, Ltd.

23/Bldg. #115 Lane 572, Bibo Roac
Zhangjiang Hi-Tech Park
Shanghai, CHINA
Tel: +86 021 5080-3866
Fax: +86 021 5080-4600

DEVELOPMENT NOTE

How do users control Tiny C complier inWicePlus2? Basic, New version Tiny C compiler is almost the same with previous version. So, users can run the project built in these two versions. But users have to pay attention to something differences in these two versions.

1. Uninstall WicePlus1.xxx . User have to remove the previous WicePlus version completely. So, during uninstalling process, users have to choose remove option. After uninstalling, users have to install WicePlus2. Out new C Compiler is involved in WicePlus2.

2. Delete system.inc and sysdef.inc in user's project. For example users created a version 1 project prg1.c in the path of D:\develop\. So, there are two Tiny C compiler system files, system.inc and sysdef.inc, in the same folder. Users have to delete these two files in the path of D:\develop\.

3. Assign rpage , iopage, bank clearly. In version 1, rpage 0, iopage 0 and bank 0 can be omitted if you want to declare a variable in these registers. But in version 2, users can't ignore rpage 0, iopage 0 and bank 0. Users must declare variable clearly in these " 0 " state. Although MCU has just one rpage , iopage or bank, variables declared in these position must be assigned which page or bank.

4. We provide a good efficient C compiler. Users can read converse table in page 57.

5. Compiler will dynamic to occupy general common register. Compiler will tell users which common registers have to save and backup in interrupt service routine. Please reference to sec. 5.10.3.

We hope we provide an ideal tool for developing product. If you have any problems about C compiler, please mail us with these IP:

myjian@emc.com.tw

Contents

1 Introduction 1

2 WicePlus Inferface 2

3 Getting Started 18

Appendix

User's Guide Revision History		
Doc. Version	Revision Description	Date
1.0	User's Guide initial version	2005/07/27
1.1	Add usage of long data to "Note" at section "4.5 Data type"	2005/08/31
1.2	Add Data type int and unsigned int	2006/07/27
2.0	Users have declare clearly in rpage 0, iopage 0 and bank 0	2006/02/12
2.1	1. Add "PAGE @0X0" in interrupt save subroutine 2. Write more detail about backup and restore instruction in example and note, 5.10 3. Add inline assembly multiple instruction "MUL" for EM78569, EM78367 and EM78369. 4. Optimized c=(a+b) << 1; unsigned int a, b, c. 5. Add note about using #include "xxx.c" , sec 4.3. 6. Illustrate Multiple- source- file programs, page 26 7. Compiler will dynamic to occupy general common register. WicePlus will tell users which common registers 0x10~0x1F have to be save and backup in interrupt service routine, sec 5.10.3	2006/03/27

Chapter 1

Introduction

1.1 Overview

The EM78 Series C Compiler is a supplementary language translator that allows user to write his application in C language. User's source code can then be translated via this compiler into assembly source code to generate the binary machine code.

NOTE

- *Please note that WicePlus can only be installed in the predefined directory (C:\EMC\WicePlus). This restriction is to prevent user from assigning an installation path that contains space char which may cause serious error while compilation.*
- *The file path (.cpj , *.c or *.h) CAN'T contain space in it. I f there are spaces in the path , error will occurred while compilation .*

1.2 System Requirements

1.2.1 Host Computer

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

- IBM PC (Pentium 100 or higher is recommended) or compatible computers
- Win2000, WinME, NT, or WinXP
- At least 10 MB **(**or more) free hard disk space
- At least 16MB of RAM. 32MB or more is recommended
- Mouse and USB connectors are highly recommended

1.3 Software Installation

The compiler is included in WicePlus, the EM78 Series Integrated Development Environment (IDE). When installing WicePlus, the compiler will also be installed.

1.4 ANSI Compatibility

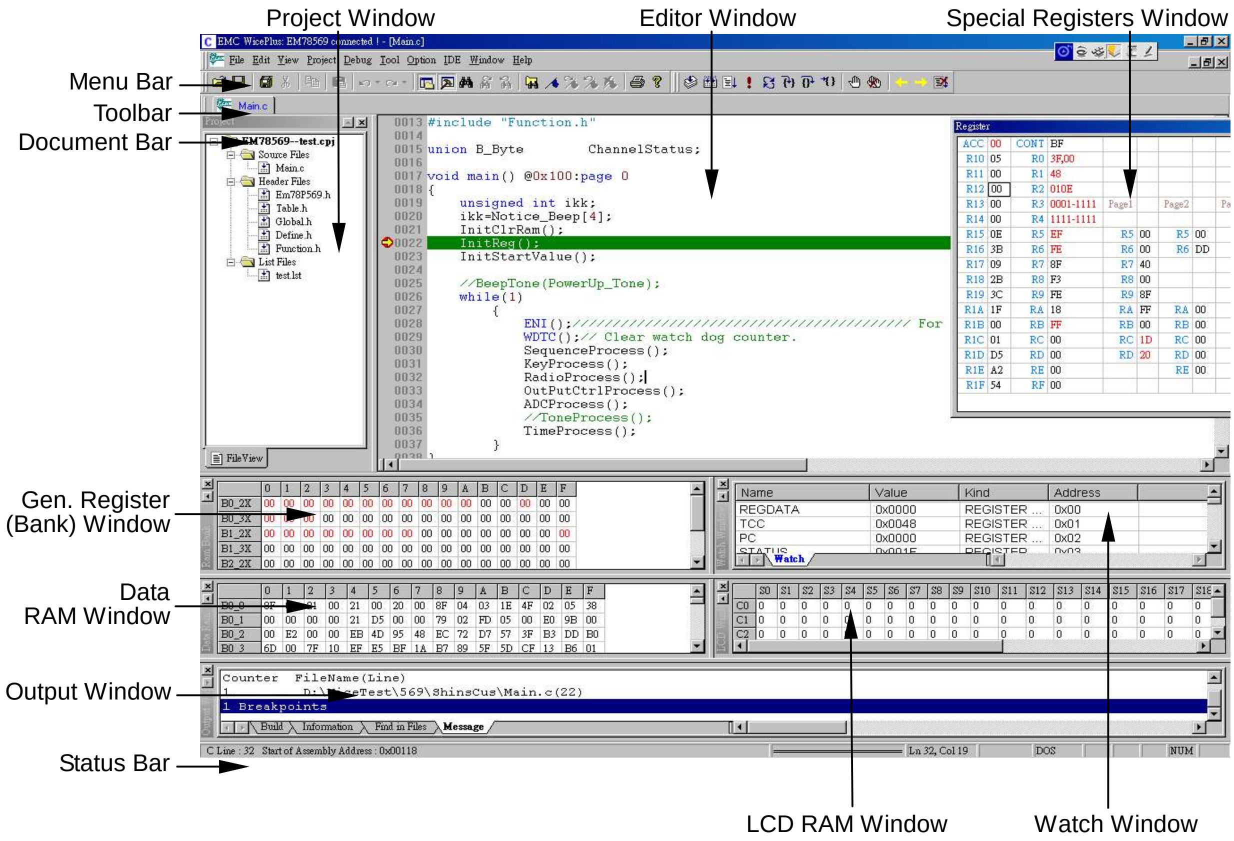
Compliance with the ANSI standard is limited to free-standing C to accommodate the unique design characteristics of the EM78 Series microcontrollers.

Chapter 2

WicePlus Interface

2.1 Overview

WicePlus 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.



```
#include "Function.h"

union B_Byte        ChannelStatus;

void main() @0x100:page 0
{
    unsigned int ikk;
    ikk=Notice_Beep[4];
    InitClrRam();
    InitReg();
    InitStartValue();

    //BeepTone(PowerUp_Tone);
    while(1)
        {
            ENI();////////////////////////////////////////////// For
            WDTC();// Clear watch dog counter.
            SequenceProcess();
            KeyProcess();
            RadioProcess();
            OutPutCtrlProcess();
            ADCProcess();
            //ToneProcess();
            TimeProcess();
        }
}
```

ACC	00	CONT	BF					
R10	05	R0	3F,00					
R11	00	R1	48					
R12	00	R2	010E					
R13	00	R3	0001-1111	Page1		Page2		Pa
R14	00	R4	1111-1111					
R15	0E	R5	EF	R5	00	R5	00	
R16	3B	R6	FE	R6	00	R6	DD	
R17	09	R7	8F	R7	40			
R18	2B	R8	F3	R8	00			
R19	3C	R9	FE	R9	8F			
R1A	1F	RA	18	RA	FF	RA	00	
R1B	00	RB	FF	RB	00	RB	00	
R1C	01	RC	00	RC	1D	RC	00	
R1D	D5	RD	00	RD	20	RD	00	
R1E	A2	RE	00			RE	00	
R1F	54	RF	00					

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
B0_2X	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
B0_3X	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
B1_2X	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
B1_3X	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
B2_2X	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Name	Value	Kind	Address
REGDATA	0x0000	REGISTER ...	0x00
TCC	0x0048	REGISTER ...	0x01
PC	0x0000	REGISTER ...	0x02
STATUS	0x001E	REGISTER	0x03

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
B0_0	8F	[illegible]	21	00	21	00	20	00	8F	04	03	1E	4F	02	05	38
B0_1	00	00	00	00	21	D5	00	00	79	02	FD	05	00	E0	9B	00
B0_2	00	E2	00	00	EB	4D	95	48	EC	72	D7	57	3F	B3	DD	B0
B0_3	6D	00	7F	10	EF	E5	BF	1A	B7	89	5F	5D	CF	13	B6	01

	S0	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14	S15	S16	S17	S1
C0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

```
Counter   FileName(Line)
1           D:\WiceTest\569\ShinsCus\Main.c(22)
1 Breakpoints
```

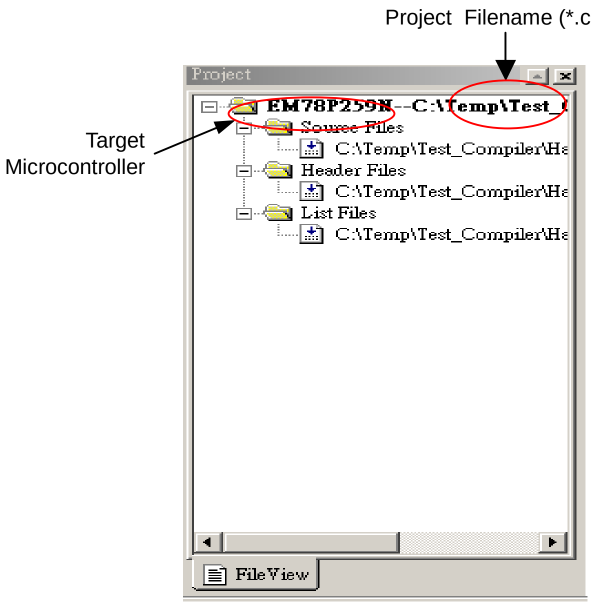
C Line : 32   Start of Assembly Address : 0x00118   Ln 32, Col 19   DOS   NUM

Fig. 2-1 WicePlus Main Window Layout

2.2 WicePlus Sub-Windows

The sub-window may be displayed or hidden by clicking on the pertinent window commands from the View menu (see Section 4.3.3.3)

2.2.1 The "Project" Window



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

Where**:**

Source Files (*.*c*) – are the assembly source files that are added into the current project.

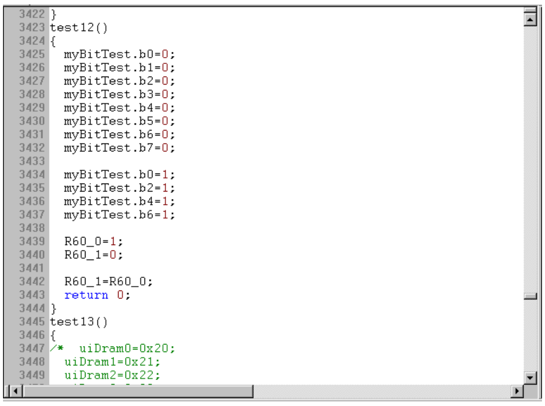
Header Files (*.*h)*– are the reference files required by source program.

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

Fig. 2-2 ***Project** Window*

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

2.2.2 The "Editor" Window



```
}
test12()
{
  myBitTest.b0=0;
  myBitTest.b1=0;
  myBitTest.b2=0;
  myBitTest.b3=0;
  myBitTest.b4=0;
  myBitTest.b5=0;
  myBitTest.b6=0;
  myBitTest.b7=0;

  myBitTest.b0=1;
  myBitTest.b2=1;
  myBitTest.b4=1;
  myBitTest.b6=1;

  R60_0=1;
  R60_1=0;

  R60_1=R60_0;
  return 0;
}
test13()
{
/*  uiDram0=0x20;
  uiDram1=0x21;
  uiDram2=0x22;
```

Fig. 2-3 ***Editor** Window*

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

The **Editor**'s major features are –

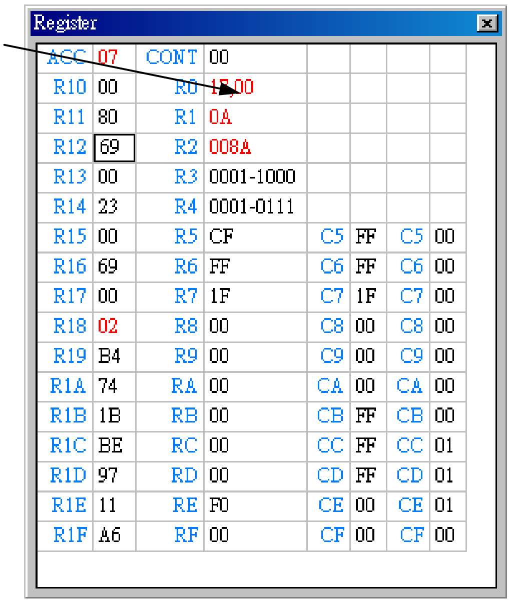
- Unlimited file size
- Multiple files can be opened and displayed at the same time
- Insert (overstrike) mode for editing
- Undo/Redo

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)

2.2.3 The "Special Register" Window

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

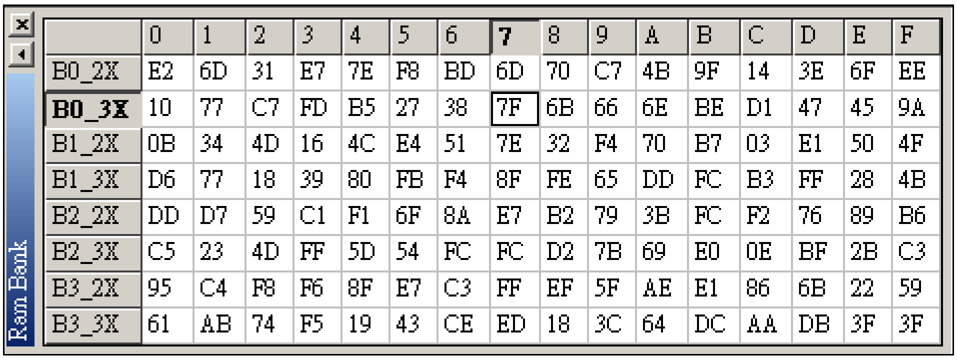


When value changes, it is shown in red

ACC	07	CONT	00				
R10	00	R0	1F00				
R11	80	R1	0A				
R12	69	R2	008A				
R13	00	R3	0001-1000				
R14	23	R4	0001-0111				
R15	00	R5	CF	C5	FF	C5	00
R16	69	R6	FF	C6	FF	C6	00
R17	00	R7	1F	C7	1F	C7	00
R18	02	R8	00	C8	00	C8	00
R19	B4	R9	00	C9	00	C9	00
R1A	74	RA	00	CA	00	CA	00
R1B	1B	RB	00	CB	FF	CB	00
R1C	BE	RC	00	CC	FF	CC	01
R1D	97	RD	00	CD	FF	CD	01
R1E	11	RE	F0	CE	00	CE	01
R1F	A6	RF	00	CF	00	CF	00

Fig. 2-4 ***Special Register*** *Window*

2.2.4 The "General Registers (Ram Bank)" Window

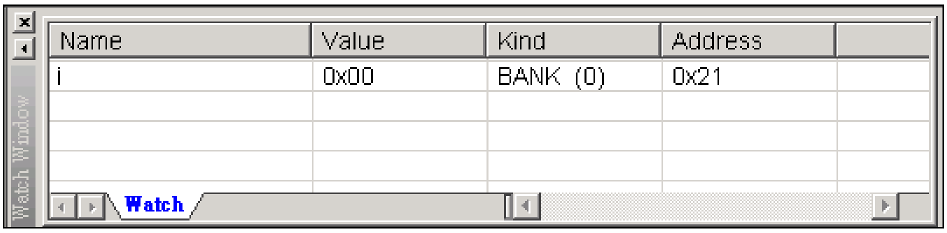


	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
B0_2X	E2	6D	31	E7	7E	F8	BD	6D	70	C7	4B	9F	14	3E	6F	EE
B0_3X	10	77	C7	FD	B5	27	38	7F	6B	66	6E	BE	D1	47	45	9A
B1_2X	0B	34	4D	16	4C	E4	51	7E	32	F4	70	B7	03	E1	50	4F
B1_3X	D6	77	18	39	80	FB	F4	8F	FE	65	DD	FC	B3	FF	28	4B
B2_2X	DD	D7	59	C1	F1	6F	8A	E7	B2	79	3B	FC	F2	76	89	B6
B2_3X	C5	23	4D	FF	5D	54	FC	FC	D2	7B	69	E0	0E	BF	2B	C3
B3_2X	95	C4	F8	F6	8F	E7	C3	FF	EF	5F	AE	E1	86	6B	22	59
B3_3X	61	AB	74	F5	19	43	CE	ED	18	3C	64	DC	AA	DB	3F	3F

Fig. 2-5 ***General Registers (Ram Bank)*** *Window*

The **General Registers (Ram Bank)** window shows the updated contents of the common ram bank registers.

2.2.5 The “Watch” Window



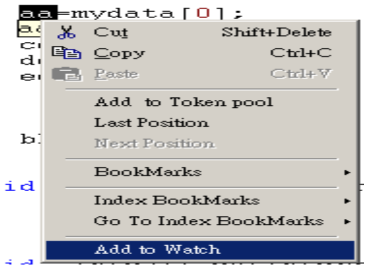
Name	Value	Kind	Address
i	0x00	BANK (0)	0x21

Fig. 2-6 ***Watch*** *Window*

In the **Watch** window, you can add variables that are declared in C. The **Watch** window will show the defined C variable information, such as name, contents, bank, and address.

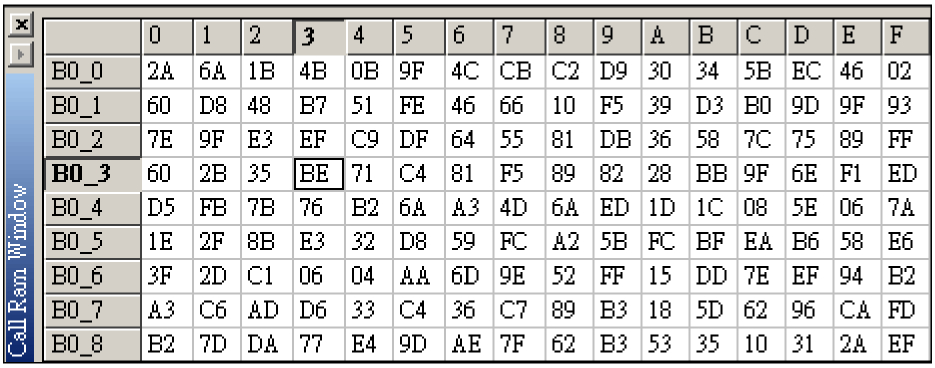
The step to add a variable to watch window :

1. Reverse the variable . (in such case is “aa”)
2. Click right button of mouse , and a menu popup .
3. Select “Add to Watch” item



```
aa=mydata[0];
```

2.2.6 “Data RAM” Window

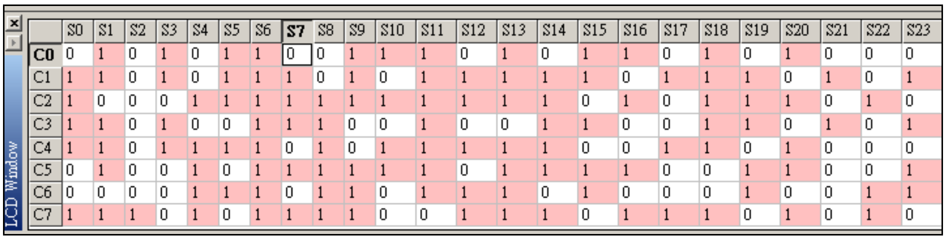


	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
B0_0	2A	6A	1B	4B	0B	9F	4C	CB	C2	D9	30	34	5B	EC	46	02
B0_1	60	D8	48	B7	51	FE	46	66	10	F5	39	D3	B0	9D	9F	93
B0_2	7E	9F	E3	EF	C9	DF	64	55	81	DB	36	58	7C	75	89	FF
B0_3	60	2B	35	BE	71	C4	81	F5	89	82	28	BB	9F	6E	F1	ED
B0_4	D5	FB	7B	76	B2	6A	A3	4D	6A	ED	1D	1C	08	5E	06	7A
B0_5	1E	2F	8B	E3	32	D8	59	FC	A2	5B	FC	BF	EA	B6	58	E6
B0_6	3F	2D	C1	06	04	AA	6D	9E	52	FF	15	DD	7E	EF	94	B2
B0_7	A3	C6	AD	D6	33	C4	36	C7	89	B3	18	5D	62	96	CA	FD
B0_8	B2	7D	DA	77	E4	9D	AE	7F	62	B3	53	35	10	31	2A	EF

Fig. 2-7 ***Data RAM*** *Window*

The **Data RAM** window is accessible only if RAM is available form the target microcontroller currently in use. The **Data RAM** window shows the contents of the data RAM.

2.2.7 “LCD RAM” Window



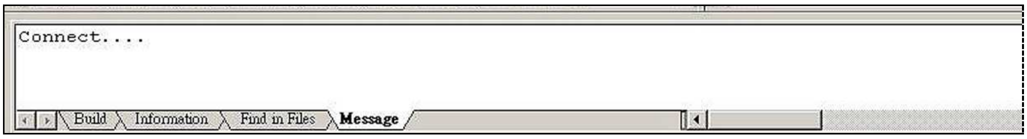
	S0	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14	S15	S16	S17	S18	S19	S20	S21	S22	S23
C0	0	1	0	1	0	1	1	0	0	1	1	1	0	1	0	1	1	0	1	0	1	0	0	0
C1	1	1	0	1	0	1	1	1	0	1	0	1	1	1	1	1	0	1	1	1	0	1	0	1
C2	1	0	0	0	1	1	1	1	1	1	1	1	1	1	1	0	1	0	1	1	1	0	1	0
C3	1	1	0	1	0	0	1	1	1	0	0	1	0	0	1	1	0	0	1	1	0	1	0	1
C4	1	1	0	1	1	1	1	0	1	0	1	1	1	1	1	0	0	1	1	0	1	0	0	0
C5	0	1	0	0	1	0	1	1	1	1	1	1	0	1	1	1	1	0	0	1	1	0	0	1
C6	0	0	0	0	1	1	1	0	1	1	0	1	1	1	0	1	0	0	0	1	0	0	1	1
C7	1	1	1	0	1	0	1	1	1	1	0	0	1	1	1	0	1	1	1	0	1	0	1	0

Fig. 2-8 ***LCD RAM*** *Window*

If supported by the target microcontroller in use, the **LCD RAM** window will show the contents of the LCD RAM. “**Cx**” denotes LCD signal “COM x.” “**Sx**” denotes LCD signal “Segment x.”

To modify the contents of the LCD RAM element, double click on the chosen element (grid block). The color of the elements will change to pink (1) from no color (0) and vise-versa. Any related messages will be shown in the **Output** window.

2.2.8 “Output” Window



```
Connect....
```

Fig. 2-9 ***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**, and **Message**, 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) in 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.

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

2.3 WicePlus Menu Bar and its Commands

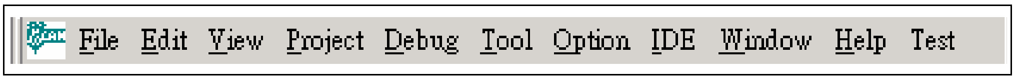


Fig. 2-10 ***Menu** Bar*

2.3.1 File Menu

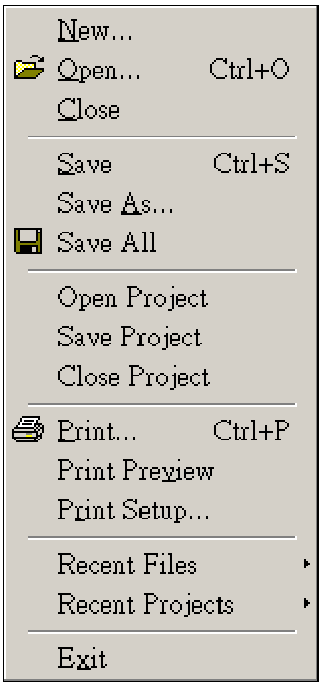


Fig. 2-11 ***File** Menu*

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 used file

Recent Projects View the record of the recently used project

Exit Exit from WicePlus Program

2.3.2 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

Find… Find the defined strings in the active window

Find in Files Find the defined string in the active and inactive files

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
Find…	Find the defined strings in the active window
Find in Files	Find the defined string in the active and inactive files
Replace…	Same as standard "find and replace" editing functions

Bookmarks	Bookmark the line at cursor position
Index Bookmarks	Clear all bookmarks and 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

2.3.3 View Menu

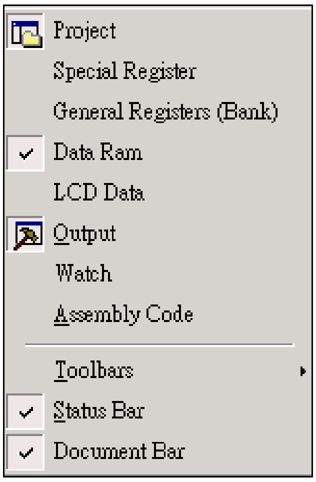


Fig. 2-13 ***View*** *Menu*

Project	Show/hide **Project** window
Special Registers	Show/hide **Special Register** window
General Registers (Bank)	Show/hide **General Register (Bank)** 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	Show/hide Assembly Code in/from **Editor** window
Toolbars	Show/hide Standard, Build, or both toolbars
Status Bar	Show/hide Status bar
Document Bar	Show/hide Document bar

2.3.4 Project Menu

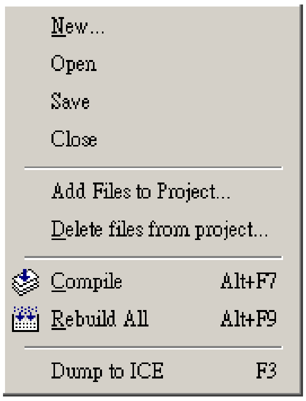


Fig.2-14 ***Project*** *Menu*

New…	Create a new project
Open	Open an existing project
Save	Save the active project together with all related files
Close	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
Compile	Compile the active file in the **Editor** window
Rebuild All	Compile all files
Dump to ICE	Dump the program code to ICE

2.3.5 Debug Menu

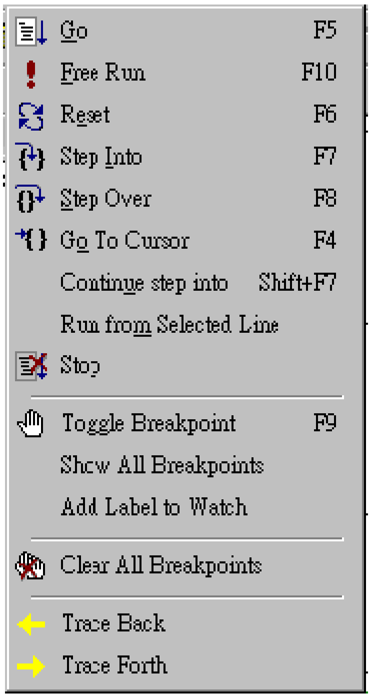


Fig.2-15 ***Debug*** *Menu*

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 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”

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 Execute instructions as “Step Into” but continuously. Users can see the change of registers. It is not the same as Go.

Run from Select Line Run program starting from the line where the cursor is.

Stop Stop running status

Toggle Breakpoint Set or remove a breakpoint

Show All Breakpoints Show all breakpoints set-up data in the **Output** window

Clear All Breakpoints Clear all breakpoints

2.3.6 Tool Menu

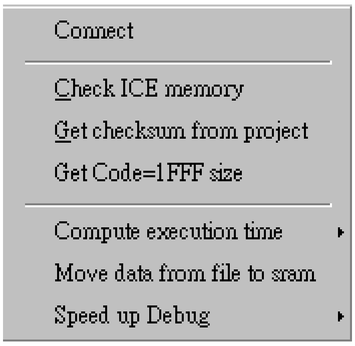


Fig. 2-16 ***Tool*** *Menu*

Connect Define printer port connection with ICE (default is 378H)

Check ICE Memory Check available memory from ICE

Get Checksum from Project Obtain checksum from the compiled program

Get Code=1FFF size Obtain the occupied program rom size and empty size

Compute Execution Time Compute the execution time between two breakpoints.

2.3.7 Option Menu

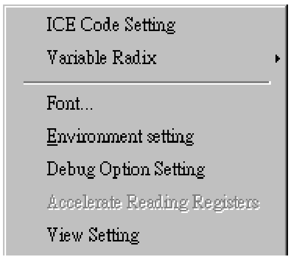


Fig. 2-17 Option Menu

ICE Code Setting Set code option for the selected microcontroller

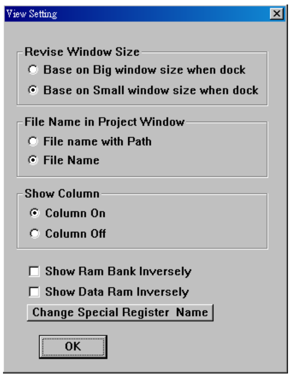
Variable Radix Select between decimal or hex option

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

Environment Setting To set WicePlus environment variable, for example, whether list file is created or not, whether map file is created or not and the number of editor window.

Debug Option Setting To set debugger variables options

View Setting GUI view setting, such as column on and off in Editor.


View Setting

Revise Window Size
○ Base on Big window size when dock
● Base on Small window size when dock

File Name in Project Window
○ File name with Path
● File Name

Show Column
● Column On
○ Column Off

□ Show Ram Bank Inversely
□ Show Data Ram Inversely

2.3.8 IDE Menu

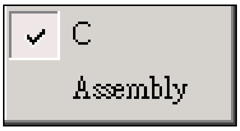


*Fig. 2-18 **IDE** Menu*

C Select C language editing

Assembly Select Assembly language editing

2.3.9 Window Menu

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

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

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

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

Cascade Rearrange all **Editor** window active files so that they appear

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

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 WicePlus window Menu bar.*

Close All Close all opened files

2.3.10 Help Menu

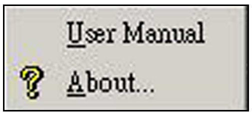
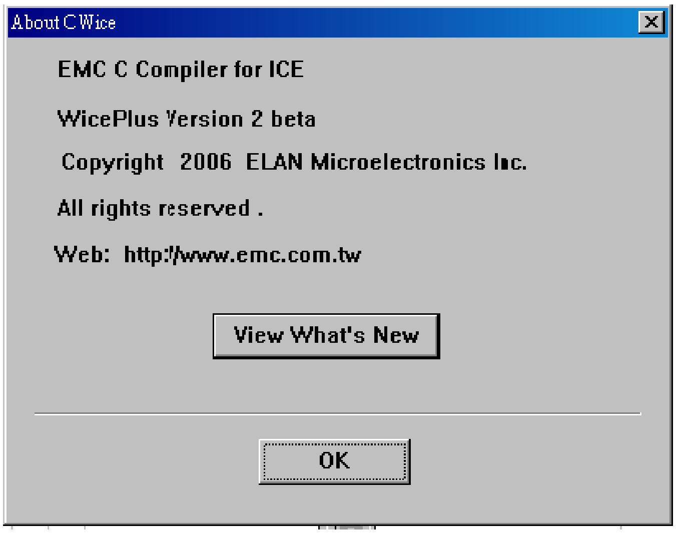


Fig. 2-20a ***Help*** *Menu*

User's Manual Open the WicePlus User's Manual .

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


About C Wice

EMC C Compiler for ICE

WicePlus Version 2 beta

Copyright 2006 ELAN Microelectronics Inc.

All rights reserved .

Web: http://www.emc.com.tw

Fig. 2-20b ***About*** *Dialog*

2.4 Toolbar

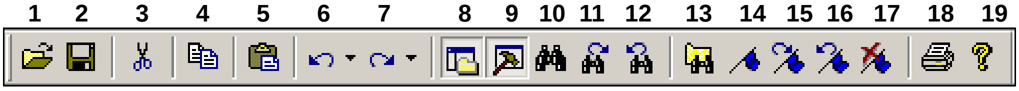


Fig. 2-21a WicePlus Main Window (Standard) Toolbar

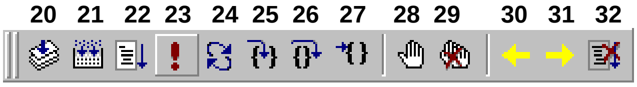


Fig. 2-21b WicePlus Main Window (Build) Toolbar

2.4.1 Toolbar Icons and Functions

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

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

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

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

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

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

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

7 **Redo:** cancel the last "undo" (restore the "undone" editing action)

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

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

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

11 **Find Down:** find string from cursor position toward the end of file

12 **Find Up:** find string from cursor position toward the beginning of file

13 **Find from Files:** find string from inactive files

14 **Bookmark:** bookmark the line at cursor position

15 **Jump Down to Bookmark:** jump to next bookmark from cursor position toward the end of file

16 **Jump Up to Bookmark:** jump to next bookmark from cursor position toward the beginning of file

17 **Clear Bookmarks:** clear all bookmarks

18 **Print:** print the active file (Ctrl + P)

19 **About:** about compiler version and other information

20 **Compile:** compile the active file in the **Editor** window (Alt + F7)

21 **Rebuild All:** compile all files in the project (Alt + F9)

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

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

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

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

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

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

28 **Insert/Remove Breakpoint:** insert/remove toggle for breakpoints

29 **Remove All Breakpoints:** remove all breakpoints

30 **Stop:** stop running

2.5 Document Bar

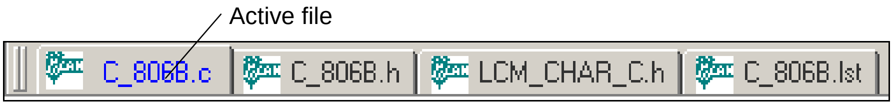
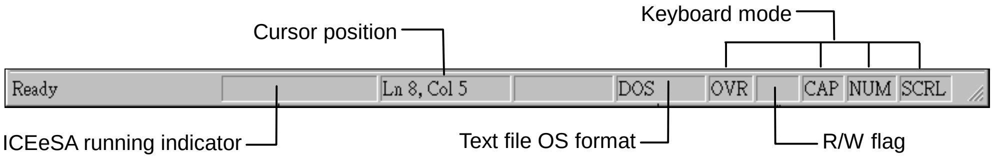

Active file

Fig. 2-22 WicePlus 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 (place in front of the **Editor** window to perform editing). Highlighted filename is the active file (function is similar with taskbar buttons under Windows).

2.6 Status Bar



A WicePlus 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 3
Getting Started

3.1 Hardware Power-up

With the E8-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 WicePlus IDE software when ICE and target board power-up is confirmed to function normally.

3.2 Starting the WicePlusPLUS Program

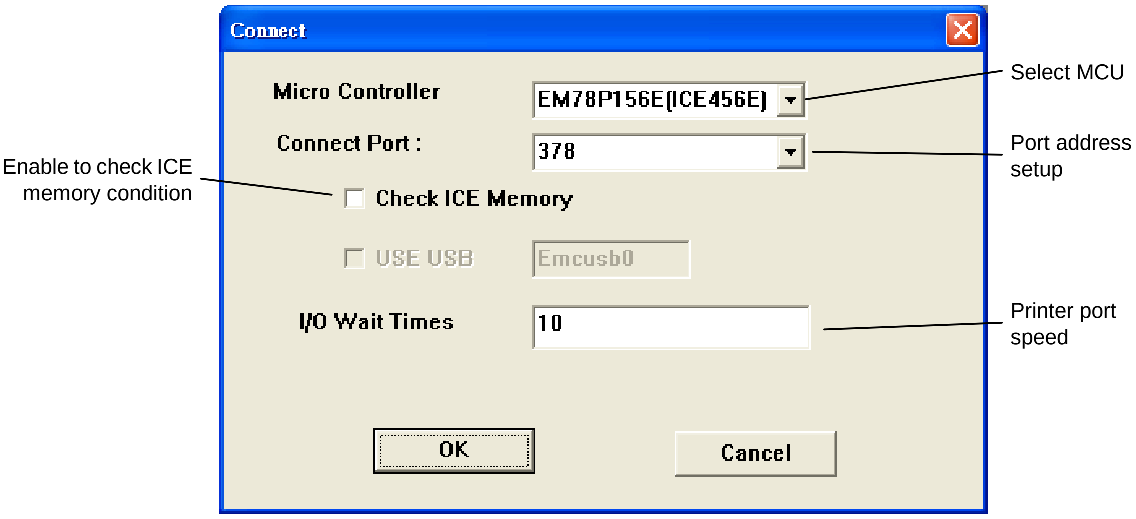
To start WicePlus Program, click on the WicePlus icon from desktop or from Windows Start menu. When starting from the Start menu, click Programs, then look for WicePlus group and click on WicePlus icon.

3.2.1 Connect Dialog

Once the program is started, the main window of the program will initially display the **Connect** dialog to prompt you to set the proper connection between your existing target microcontroller and printer port (default is 378H).

You may also enable the “Check ICE Memory” check box to check the condition of the ICE memory. “I/O Wait Time” depicts the I/O response speed. Increase the value for slower speed and decrease for faster speed.

Click **OK** button when done.



Connect

Micro Controller: EM78P156E(ICE456E)

Connect Port : 378

□ Check ICE Memory

□ USE USB Emcusb0

I/O Wait Times 10

Select MCU

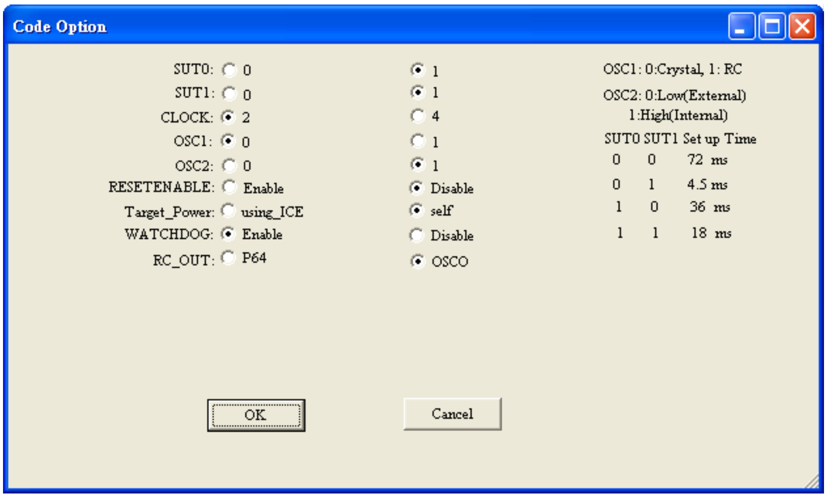
Port address setup

Enable to check ICE memory condition

Printer port speed

Fig. 3-1 WicePlus Program ***Connect*** *Dialog*

3.2.2 Code Option Dialog



Code Option

SUT0: ○ 0 ● 1

SUT1: ○ 0 ● 1

CLOCK: ● 2 ○ 4

OSC1: ● 0 ○ 1

OSC2: ○ 0 ● 1

RESETENABLE: ○ Enable ● Disable

Target_Power: ○ using_ICE ● self

WATCHDOG: ● Enable ○ Disable

RC_OUT: ○ P64 ● OSCO

OSC1: 0:Crystal, 1: RC

OSC2: 0:Low(External)
1:High(Internal)

SUT0	SUT1	Set up Time
0	0	72 ms
0	1	4.5 ms
1	0	36 ms
1	1	18 ms

*Fig. 3-2 WicePlus Program **Code** 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.

3.3 Create a New Project

To create a new project, you need to configure your project with the following steps**:**

1. From the Menu bar click on **File** or **Project** menu and choose **New** command from the resulting pull-down menu.

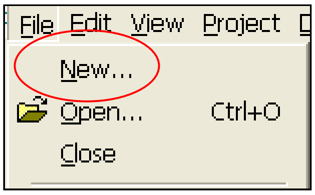


Fig. 3-3 "File" Menu

2. The **New** dialog (shown below) will then display if you have clicked the **New** command from the **File** menu. Otherwise, **New Project** dialog will display (Fig. 3-5) if the **New** command is derived from **Project** menu.

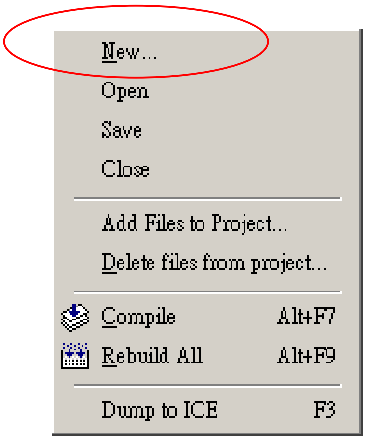
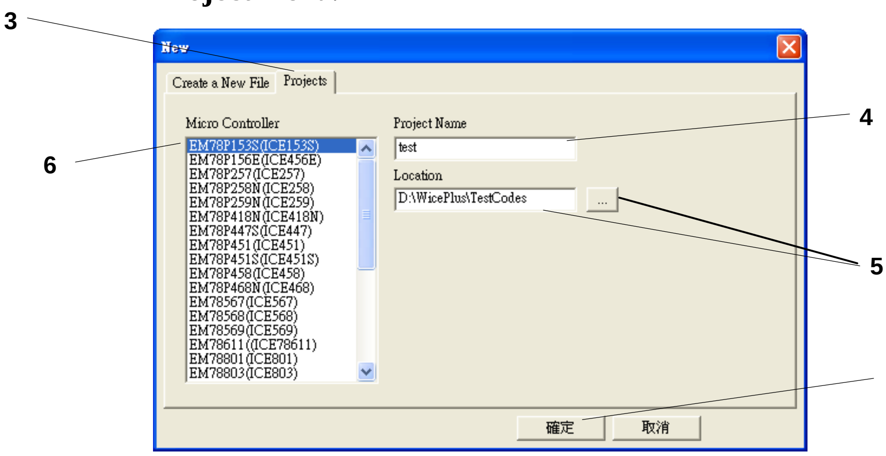


Fig. 3-4 "Project" Menu



New

Create a New File | Projects

Micro Controller:
EM78P153S(ICE153S)
EM78P156E(ICE456E)
EM78P257(ICE257)
EM78P258N(ICE258)
EM78P259N(ICE259)
EM78P418N(ICE418N)
EM78P447S(ICE447)
EM78P451(ICE451)
EM78P451S(ICE451S)
EM78P458(ICE458)
EM78P468N(ICE468)
EM78567(ICE567)
EM78568(ICE568)
EM78569(ICE569)
EM78611((ICE78611)
EM78801(ICE801)
EM78803(ICE803)

Project Name: test

Location: D:\WicePlus\TestCodes

Fig. 3-5 "New" Dialog Showing Project Tab for Creating New Project (Derived from **File** Menu)

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

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

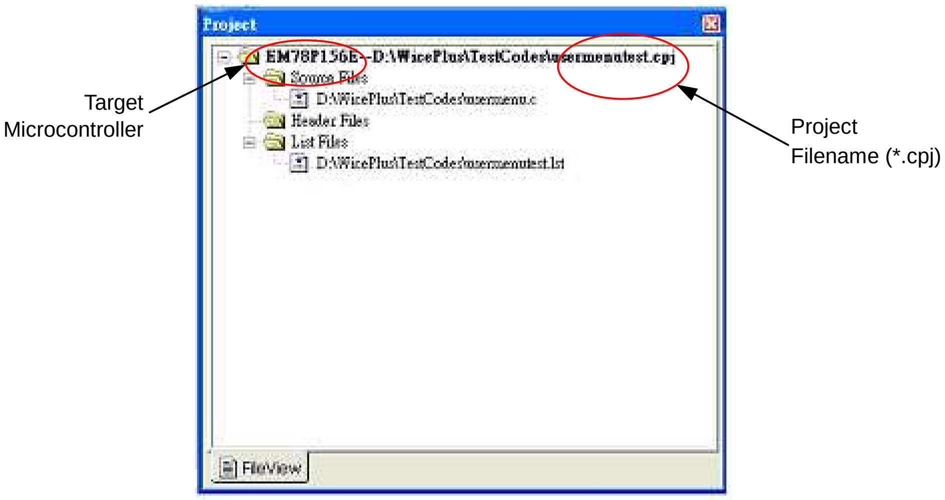

Project
EM78P156E--D:\WicePlus\TestCodes\usermenutest.cpj
Source Files
D:\WicePlus\TestCodes\usermenu.c
Header Files
List Files
D:\WicePlus\TestCodes\usermenutest.lst
FileView
Target Microcontroller
Project Filename (*.cpj)

Fig. 3-6 "Project" Window

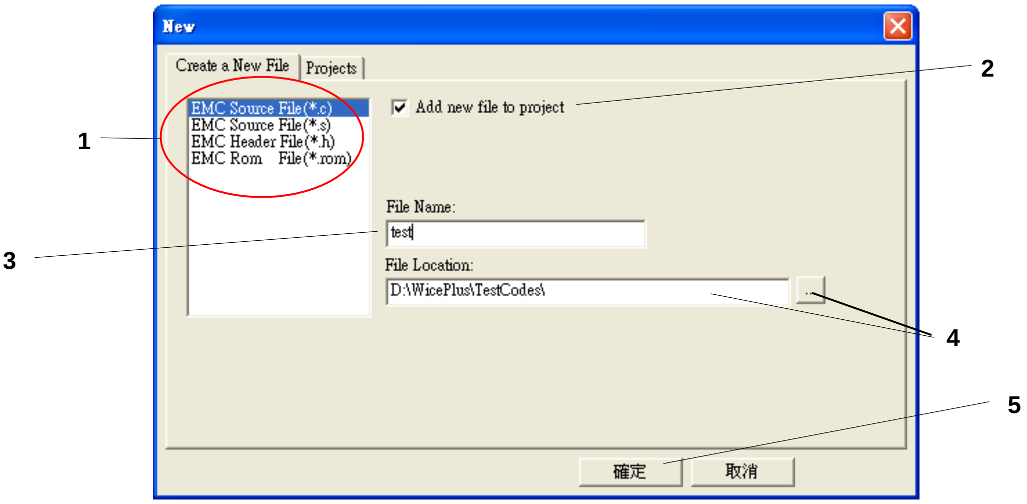
3.4 Add and Remove Source Files from/to Project

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

3.4.1 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 menu) to create your new source file and use the WicePlus text editor to compose its content.

1. Click the **File** tab of the **NEW** dialog and select the type of source file you want to create from the **EMC Source File** list box, i.e., ***.c*** (default) for assembly file; ***.h*** for header file.



EMC Source File(*.c)
EMC Source File(*.s)
EMC Header File(*.h)
EMC Rom File(*.rom)

☑ Add new file to project

File Name:
test

File Location:
D:\WicePlus\TestCodes\

Fig. 3-7 "New" Dialog Showing Project Tab for Creating a New Source File

2. Check **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** 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.

3.4.2 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** menu. Choose **Add Files to Project** command from the resulting pull-down menu, and then the **Open** dialog is displayed.

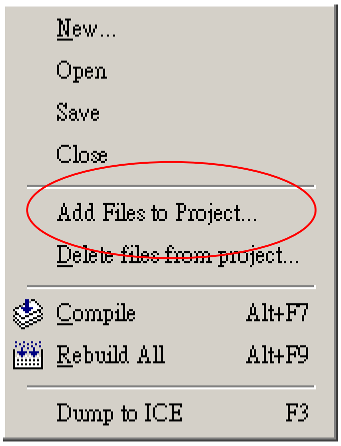


Fig. 3-8a "Add Files to Project" Command

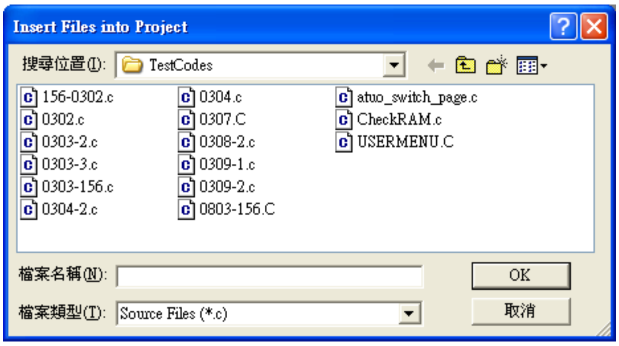


Fig. 3-8b "Open" Dialog

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

3.4.3 Deleting Source Files from Project

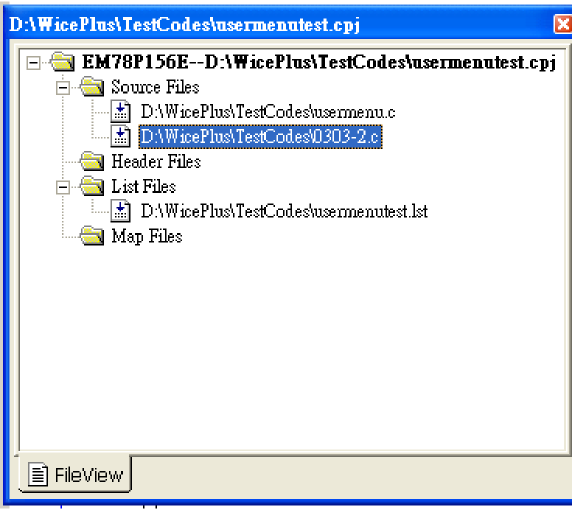

D:\WicePlus\TestCodes\usermenutest.cpj
EM78P156E--D:\WicePlus\TestCodes\usermenutest.cpj
Source Files
D:\WicePlus\TestCodes\usermenu.c
D:\WicePlus\TestCodes\0303-2.c
Header Files
List Files
D:\WicePlus\TestCodes\usermenutest.lst
Map Files
FileView

Fig. 3-9a Deleting Project Files Directly from "Project" Window

1. From the **Project** window, select the file(s) you wish to delete. Then press the **Delete** key from your keyboard.

2. You may also click on the **Delete Files from Project…** command from the **Project** pull-down menu to delete files from project.

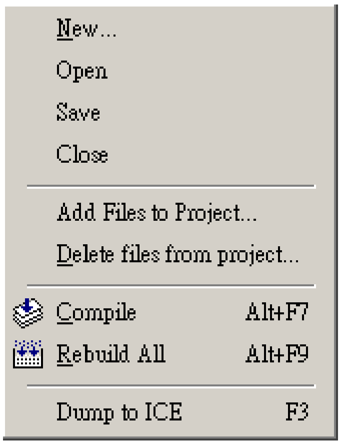


Fig. 3-9b Deleting Project Files from "Project"menu.

3.5 Editing Source Files from Folder/Project

3.5.1 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. From the Menu bar click on **File** or **Project** menu, choose **Open** command from the resulting pull-down menu.
2. From the resulting **Open** dialog (Fig. 3.8b above) click on the source file and the file is automatically opened in the **Editor** window.

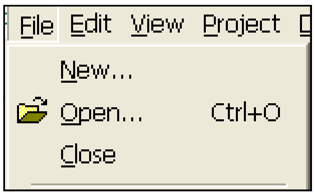
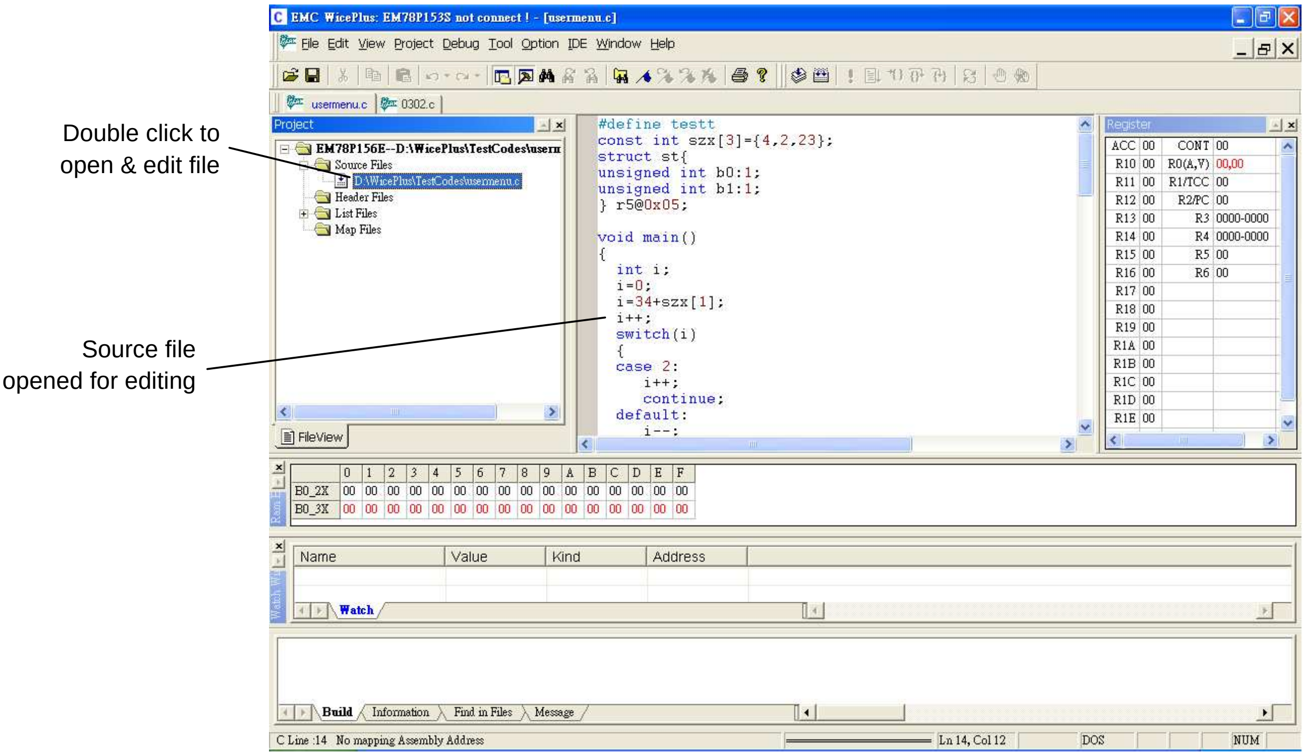


Fig. 3-10 Open &Edit Source File from "File" Menu

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

3.5.2 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.


Double click to open & edit file
Source file opened for editing
EMC WicePlus: EM78P153S not connect ! - [usermenu.c]
File
Edit
View
Project
Debug
Tool
Option
IDE
Window
Help
usermenu.c
0302.c
Project
EM78P156E--D:\WicePlus\TestCodes\userm
Source Files
D:\WicePlus\TestCodes\usermenu.c
Header Files
List Files
Map Files
FileView
Register
Watch
Build
Information
Find in Files
Message

```
#define testt
const int szx[3]={4,2,23};
struct st{
unsigned int b0:1;
unsigned int b1:1;
} r5@0x05;

void main()
{
  int i;
  i=0;
  i=34+szx[1];
  i++;
  switch(i)
  {
  case 2:
     i++;
     continue;
  default:
     i--;
```

ACC	00	CONT	00
R10	00	R0(A,V)	00,00
R11	00	R1/TCC	00
R12	00	R2/PC	00
R13	00	R3	0000-0000
R14	00	R4	0000-0000
R15	00	R5	00
R16	00	R6	00
R17	00		
R18	00		
R19	00		
R1A	00		
R1B	00		
R1C	00		
R1D	00		
R1E	00		

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
B0_2X	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
B0_3X	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Name	Value	Kind	Address

C Line :14 No mapping Assembly Address Ln 14, Col 12 DOS NUM

Fig. 3-11 Editing Source File Directly from "Project" window.

3.6 Compile the Project

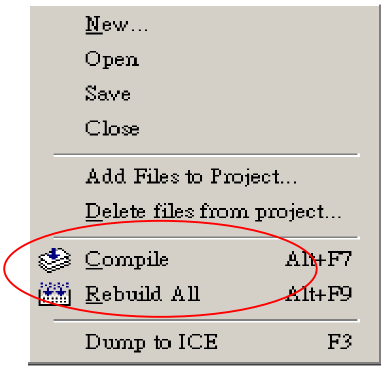


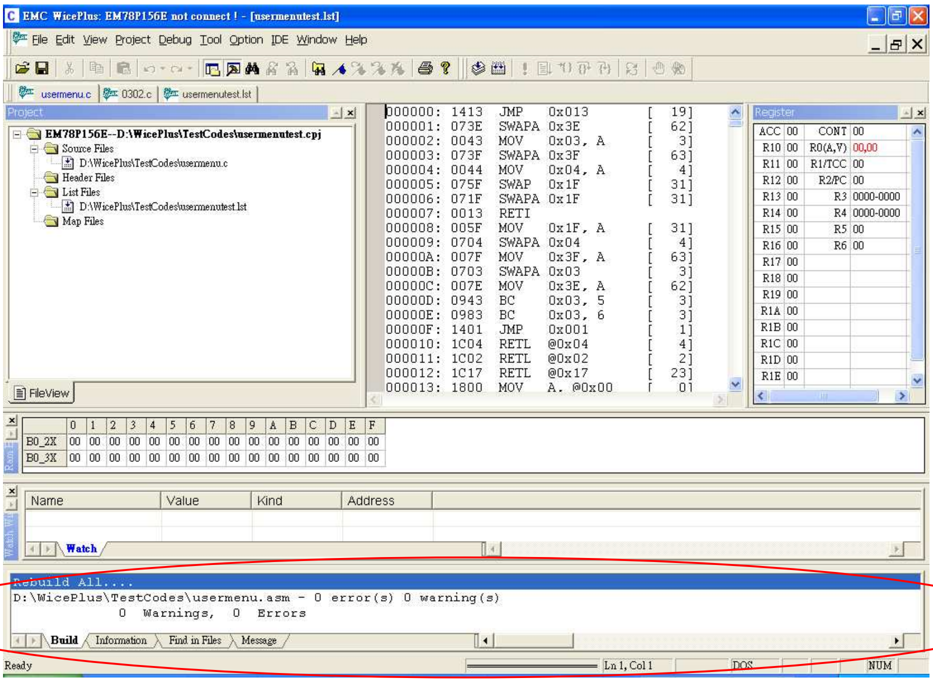
Fig. 3-12 Assemble & Link Commands

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

Rebuild All will generate objective (**.bbj*) file, list (**.lst*) file, 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.

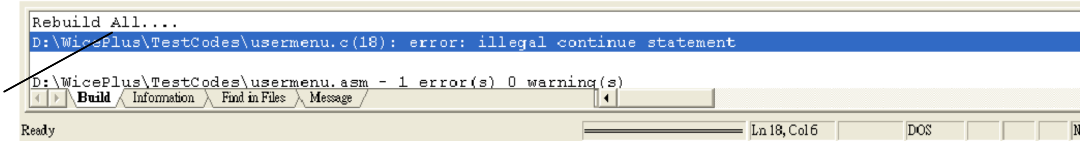

EMC WicePlus: EM78P156E not connect ! - [usermenutest.lst]
File Edit View Project Debug Tool Option IDE Window Help
usermenu.c
0302.c
usermenutest.lst
Project
EM78P156E--D:\WicePlus\TestCodes\usermenutest.cpj
Source Files
D:\WicePlus\TestCodes\usermenu.c
Header Files
List Files
D:\WicePlus\TestCodes\usermenutest.lst
Map Files
FileView
Register
Name
Value
Kind
Address
Watch
Build
Information
Find in Files
Message
Ready
Ln 1, Col 1
DOS
NUM

```
000000: 1413  JMP   0x013      [  19]
000001: 073E  SWAPA 0x3E       [  62]
000002: 0043  MOV   0x03, A    [   3]
000003: 073F  SWAPA 0x3F       [  63]
000004: 0044  MOV   0x04, A    [   4]
000005: 075F  SWAP  0x1F       [  31]
000006: 071F  SWAPA 0x1F       [  31]
000007: 0013  RETI
000008: 005F  MOV   0x1F, A    [  31]
000009: 0704  SWAPA 0x04       [   4]
00000A: 007F  MOV   0x3F, A    [  63]
00000B: 0703  SWAPA 0x03       [   3]
00000C: 007E  MOV   0x3E, A    [  62]
00000D: 0943  BC    0x03, 5    [   3]
00000E: 0983  BC    0x03, 6    [   3]
00000F: 1401  JMP   0x001      [   1]
000010: 1C04  RETL  @0x04      [   4]
000011: 1C02  RETL  @0x02      [   2]
000012: 1C17  RETL  @0x17      [  23]
000013: 1800  MOV   A. @0x00   [   0]
```

```
Rebuild All....
D:\WicePlus\TestCodes\usermenu.asm - 0 error(s) 0 warning(s)
             0  Warnings,  0  Errors
```

Fig. 3-13 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 open automatically.



```
Rebuild All....
D:\WicePlus\TestCodes\usermenu.c(18): error: illegal continue statement

D:\WicePlus\TestCodes\usermenu.asm - 1 error(s) 0 warning(s)
```

Double click to link to the source of error

Fig. 3-14 Output Window Showing Compilation Errors

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

3.7 Dumping the Compiled Program to ICE

With the source files deprived of its errors and successfully compiled, download your compiled program to ICE using the **Dump to ICE** command from **Project** drop-down menu or its corresponding shortcut key (F3).

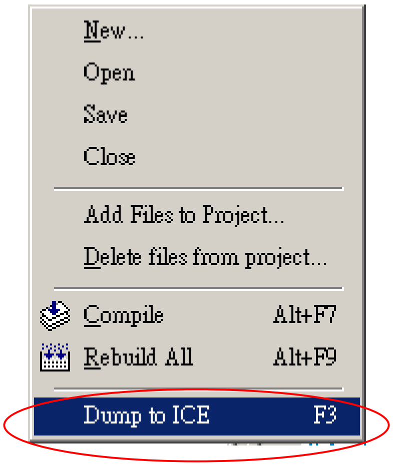


Fig. 3-15 "Dump to ICE" Command

3.8 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 WicePlus Program Toolbar.

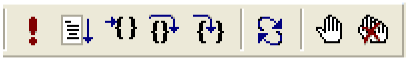


Fig. 3-16a Toolbar for Debugging Commands

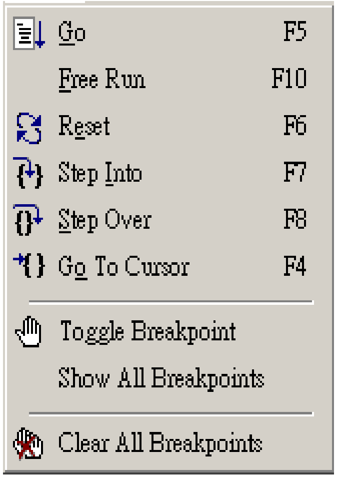


Fig. 3-16b Debugging Commands Drop-Down Menu

Toggle Breakpoint – Click with cursor positioned on the line where a breakpoint is going to be set or removed.

Clear All Breakpoints – Remove all already set breakpoints.

F5

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

F6

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

F7 **Step Into –** Execute instructions step-by-step (with register contents updated at the same time).

F8 **Step Over** – Same as "Step Into" command (see above), but the CALL instruction is executed as "Go" command.

F10 **Free Run** – 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.

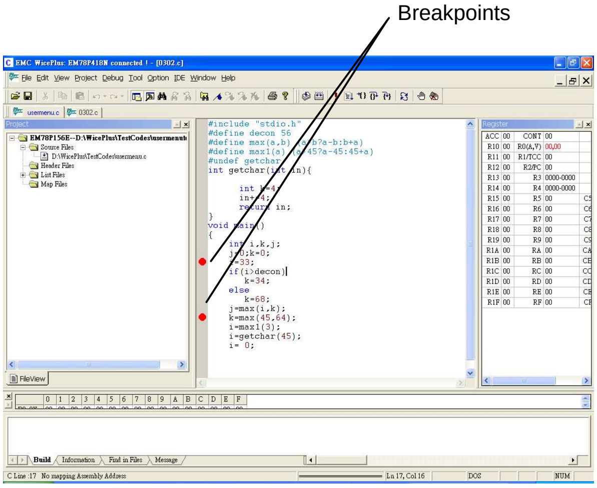
Stop – Stop running

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.

3.8.1 Breakpoints Setting

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 brown.

You can also click on the **Insert/ Remove Breakpoint** icon (hand shape) on the toolbar to set a breakpoint.


Breakpoints
EMC WicePlus: EM78P418N connected ! - [0302.c]
File Edit View Project Debug Tool Option IDE Window Help
usermenu.c
0302.c
Project
EM78P156E--D:\WicePlus\TestCodes\usermenu
Source Files
D:\WicePlus\TestCodes\usermenu.c
Header Files
List Files
Map Files
FileView
Register
Build
Information
Find in Files
Message
NUM

```
#include "stdio.h"
#define decon 56
#define max(a,b) (a>b?a-b:b+a)
#define max1(a) (a>45?a-45:45+a)
#undef getchar
int getchar(int in){

      int k=4;
      in+=4;
      return in;
}
void main()
{
    int i,k,j;
    j=0;k=0;
    i=33;
    if(i>decon)
       k=34;
    else
       k=68;
    j=max(i,k);
    k=max(45,64);
    i=max1(3);
    i=getchar(45);
    i= 0;
```

Fig. 3-17 Active Source File with a Defined Breakpoint

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 positioned on the defined breakpoint. To clear all existing breakpoints, click **Clear All Breakpoints** command from **Debug** menu.

Chapter 4

C Fundamental Elements

4.1 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 part from 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 */
```

4.2 Reserved Words

The reserved words for WicePlus 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.*
- *indir, ind, io, iopage, rpage are for MCU hardware definition and declaration.*

4.3 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.

4.3.1 #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 EMC_INCLUDE.

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

NOTE

- *We don't suggest users to include c file. Maybe compiler will meet errors it users include c file.*
- *Suppose that uaa is declared global, unsigned int (unsigned int uaa) variable in headfile.h. Now 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 use it in kkdr.c file. The same way to use in the third or more others c source file that are 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
…
main ()             //in testcode.c file
{
   uaa=0x21;
…
}

extern unsigned int uaa;  //in kkdr.c file
void ()
{
   uaa=0x38;
….
   Uaa=0x43;
}

void ()
{
   uaa=0x29;
}
```

4.3.2 #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 legible.

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
```

4.3.3 #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
```

4.3.4 #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 conditionally compiling codes with the specified symbol 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
```

4.4 Literal Constants

4.4.1 Numeric Constant

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

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
```

4.4.2 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
```

4.4.3 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”
```

4.5 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
(signed) char	–128 ~ 127	1
unsigned char	0 ~ 255	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 5.4 of Chapter 5for more details on “Bit Data Type.”*
3. *If user use* ***long*** *data type for multiplication,division,modulus,compare operation , 0x20 ~ 0x24 (5 bytes) of bank 0 are occupied by compiler. Therefore , don’t assign these address to any variable when you do those 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 users 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.
```

4.6 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;
```

4.7 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 groups 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, in stead, use bit field.*
- *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 ;  //struct R5 is related to 0x05
```

Example 2:

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

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

4.8 Array

declarator:

array-declarator:

array-declarator

[constant-expression]

array-declarator [constant-expression]

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 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
```

4.9 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;
```

4.10 Operators

4.10.1 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

Increment Operators		
Symbol	Function	Expression
++	increase by 1	expr ++
--	decrease by 1	expr --

Assignment Operators		
Symbol	Function	Expression
=	equal	expr1 = expr2

Bitwise 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

4.10.2 Prefix of Operators

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

4.11 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 which will be executed when the evaluated condition is false.

Example:

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

4.12 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();
}
```

4.13 While Statement

```
while (expression) statement
```

“While” statement will check the expression first, if the expression is true it will then execute the statement.

Example:

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

4.14 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 statement until the expression becomes FALSE.

Example:

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

4.15 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;
}
```

4.16 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);
}
```

4.17 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;
```

4.18 Function

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

4.18.1 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 the parameter for function.*
- *Function pointer is not supported.*
- *Bit data type cannot be used as a return value.*
- *For reduced ram bank wastage ,We suggest users using global variable in function instead of using argument.*

Example (Function Prototype):

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

4.18.2 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 5

Hardware Related Programming

5.1 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 is, including rpage 0.

NOTE

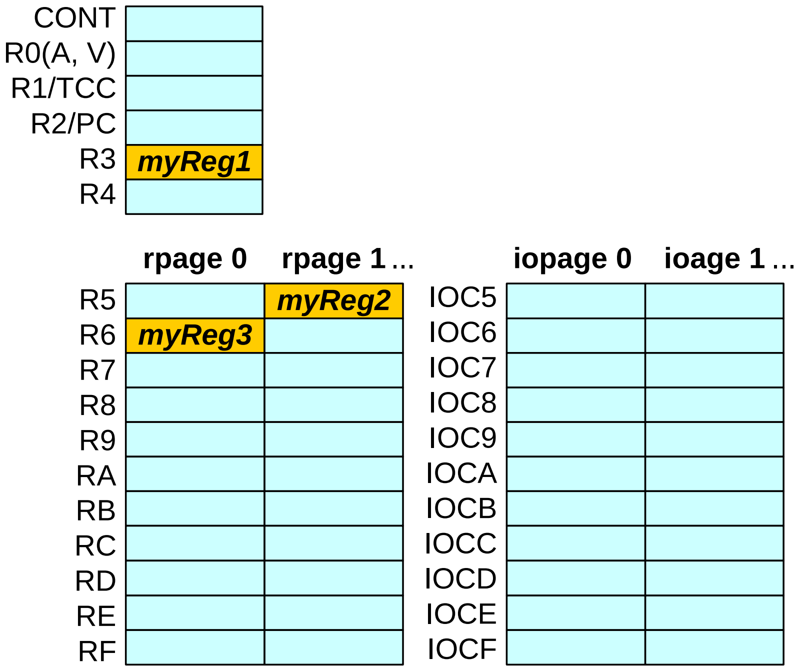
- ■ *If a variable is declared as "rpage," it cannot be declared as "bank," "iopage," or "indir" at the same time.*
- ■ *Only global variable can be declared as "rpage" data type.*
- ■ *Although an MCU just has rpage 0 , but <register page number> must be assigned.*

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;
```



CONT	
R0(A, V)	
R1/TCC	
R2/PC	
R3	*myReg1*
R4	

	rpage 0	rpage 1 ...		iopage 0	ioage 1 ...
R5		*myReg2*	IOC5		
R6	*myReg3*		IOC6		
R7			IOC7		
R8			IOC8		
R9			IOC9		
RA			IOCA		
RB			IOCB		
RC			IOCC		
RD			IOCD		
RE			IOCE		
RF			IOCF		

5.2 I/O Control Page (iopage)

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

Declare the variable at the register page it is located. Users have to declare clearly that the io variable is located at which iopage, though there is only one io control page.

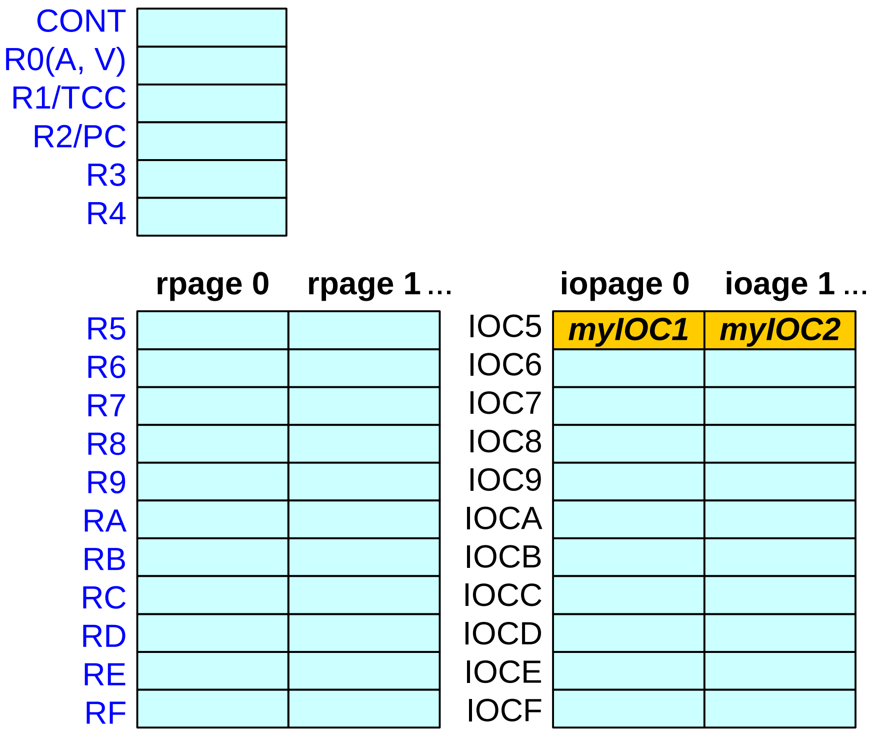
NOTE

- *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.*
- *Although an MCU just has iopage 0 , but <io control page number > must 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
```



5.3 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 variable is declare at at Bank 0.

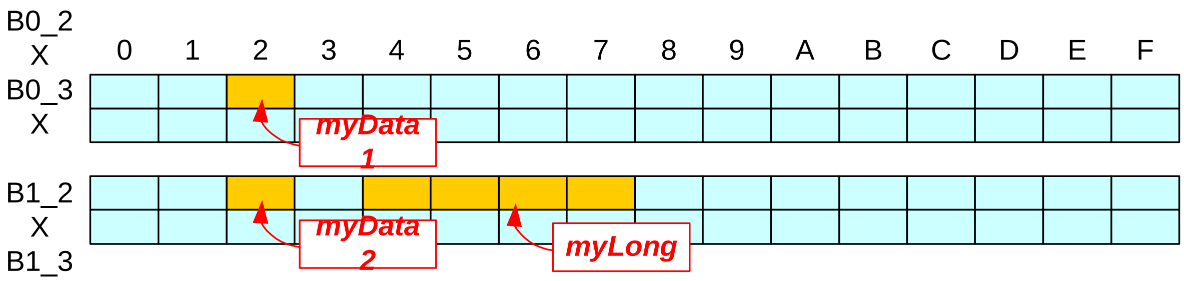
NOTE

- *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;
        // Test is located at default ram bank 0
unsigned int myData2 @0x22: bank 1;
        // myTest is at address 0x22 of ram bank 1
unsigned short myshort @0x20: bank 0;
         // myshort is at address 0x29 and 0x2A of ram bank 2
unsigned long myLong @0x24: bank 1;
        // myLong is at address 0x24~0x27 of ram bank 1
```

RAM Bank:



5.4 Bit Data Type

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

Bit data type occupies only one bit.

NOTE

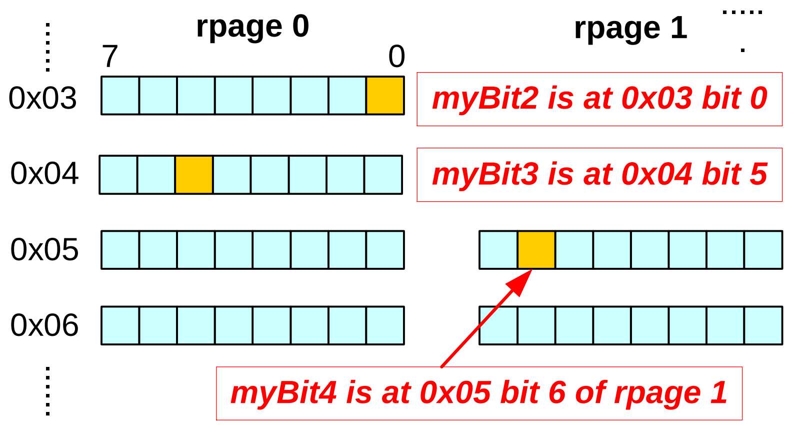
- *Bit data type cannot be used in struct and union. It is recommended to use bitfield in struct and union, such as:*

```
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 DON NOT use it as a name of “struct” or “union”.*
- *Only global variable can be declared as “bit” data type.*

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
```

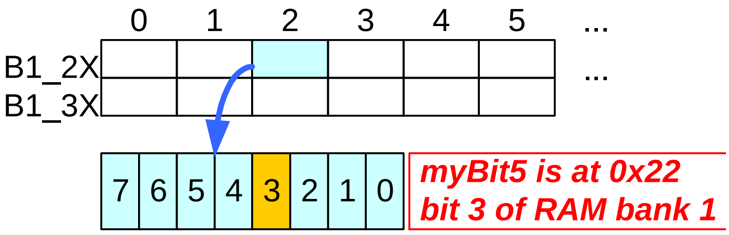


myBit2 is at 0x03 bit 0

myBit3 is at 0x04 bit 5

myBit4 is at 0x05 bit 6 of rpage 1

RAM Bank:



myBit5 is at 0x22 bit 3 of RAM bank 1

5.5 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)

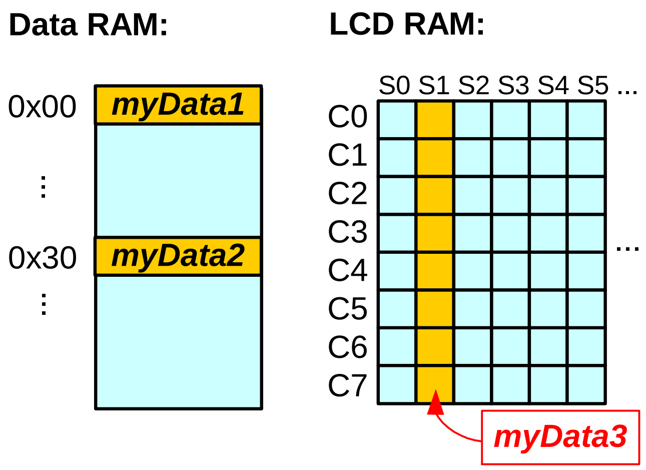
If the MCU has an LCD RAM, use “ind 1” (indirect RAM 1)

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 using “ind 0”.
indir int nData3 @0x01: ind 1;
                  //nData3 is at LCD Ram because using “ind 1”.
```



5.6 Allocating C Function to Program ROM

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

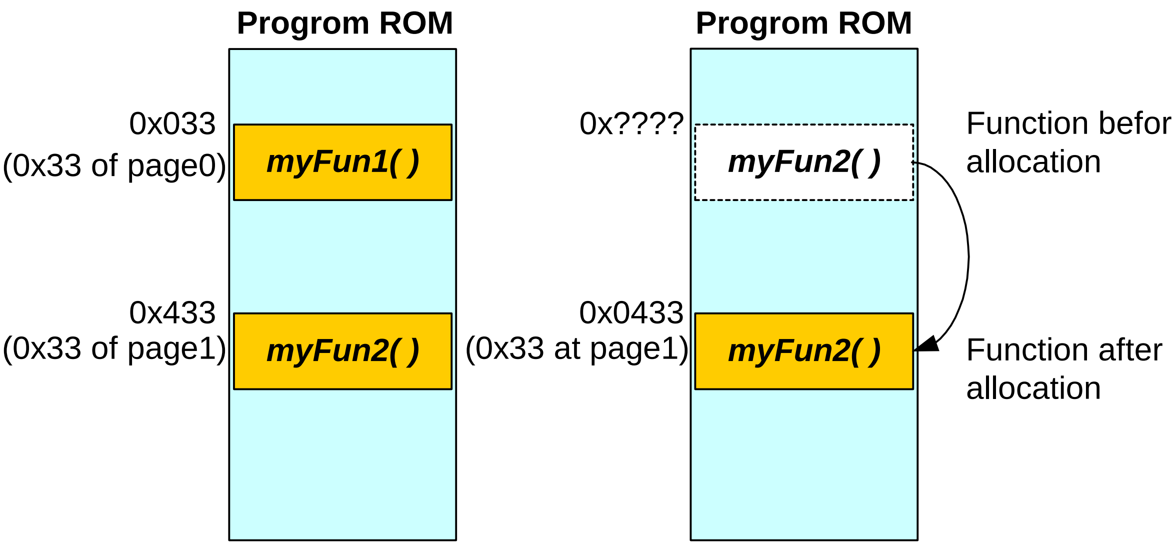
You can place function at the dedicated address of the program ROM, and use "page" instruction to allocate which page in the program ROM you wish to assign.

NOTE

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

Example:

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



5.7 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

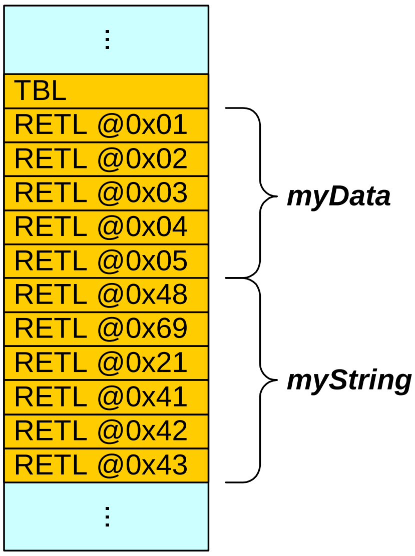
- *Use constant data type to store data into the ROM.*
- *Only global variable can be declared as “const” data type.*
- *The maximum size of a constant array variable is 255 bytes.*

Example:

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

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

Program ROM:



⋮
TBL
RETL @0x01
RETL @0x02
RETL @0x03
RETL @0x04
RETL @0x05
RETL @0x48
RETL @0x69
RETL @0x21
RETL @0x41
RETL @0x42
RETL @0x43
⋮

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.

5.8 Inline Assembler

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

5.8.1 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

- ■ *Registers in 0x10~0x1F are reserved for the C compiler. It is not advisable to use these reserved words.*
- ■ *If user has 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 5.8.2).*
- ■ *If users use 0x10~0x1F in inline assembler, compiler would not report warning or error message, but it may meet some unexpected errors.*

5.8.2 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,0x0
     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, %nbuf //register
     mov 0x03, a
     mov a, %nbuf + 1
     mov 0x04, a
}
```

Example 2:

```
int temp;
temp=0x03;                //we suppose 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
     }
}
```

5.9 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.*

Example:

```
#define SetIO(portnum, value)  _asm {mov a, @value} \
                               _asm {iow portnum}
```

5.10 Interrupt Routine

To handle Interrupts, two things have to be taken into account**:**

1. **Interrupt Save Procedure:** the procedure to save some registers before executing a service routine. For instance, ACC, R3 should be saved in the EM78P458 as Interrupts occur. Only inline assembly is allowed under Interrupt Save Procedure.

2. **Interrupt Service Routine:** is the action to be taken for Interrupt.

NOTE

- *You may ignore the details on setting interrupts as WicePlus will handle all these tasks. However, you need to concentrate more on the interrupt service routine.*
- *The "page" instructions cannot allocate the same ROM space as interrupt service routine (see Section 3.6 .*

5.10.1 Interrupt Save Procedure

void _intcall <function name>_l(void) @<interrupt vector address>: low_int <interrupt vector number>

It should be noted that "_l" (for low level interrupt) must be added after function name.

5.10.2 Interrupt Service Routine

void _intcall <function name>(void) @int <interrupt vector number>

The <interrupt vector number> means that if there are many interrupt vectors in the MCU, the sequence 0, 1, 2, 3… is provided to separate each interrupt vectors.

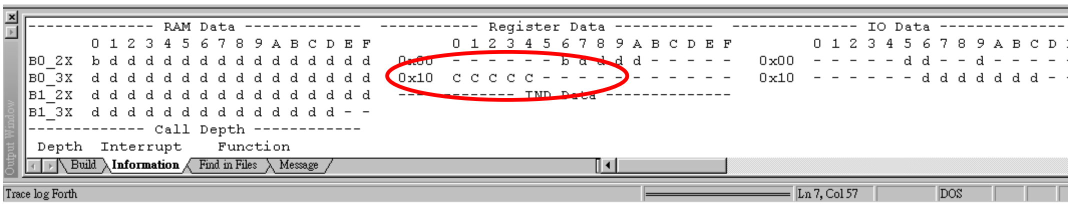
The compiler will automatically combine the saved procedure and the service routine in the <interrupt vector number>. That is MCU will jump from insterrupt save procedure to interrupt service procedure.

NOTE

- *Interrupt Save Procedure and Interrupt Service Routine cannot be assigned with parameters; otherwise the compiler will generate an error.*
- *Interrupt routine only supports one byte data operation (such as "int", "char"), otherwise the compiler will generate an error.*
- *Under interrupt service routine, you can call other functions. But long data types *, / and % operation are not allowed in the called function. Refer to Example 3 below.*
- *You must write an inline-assembly code to save some registers in the Interrupt Save Procedure per MCU type. For instance, ACC, R3, R4, and R5 should be saved in EM78R806B as interrupts occur. If the MCU supports hardware backup control for the interrupt routine, you may ignore saving the registers in the Interrupt Save Procedure. Please study the MUC spec about that. It is very important to switch to program page 0 at the end of Interrupt Save Procedure.*
- *Users have to confirm whether the bank of these saving address in interrupt save procedure is the same as the bank of restoring in interrupt service procedure.*
- *Skilled users, who only want to save fewer registers, must take note as to what operations have been done. (For example, if "*", "/" or "%" is not used at the Interrupt Service Routine program, you can skip to save register 0x1D and 0x1E). The following table shows certain operations that use special registers. Basing on this table, you can determine which registers need to be or not to be backed up.*
- *Users can't use these ram spaces that are used to backup ACC, R3, R4, R5 or other general purpose registers 0x10~0x1F.*

5.10.3 Reserved Common Registers Operation

Sixteen common registers (0x10~0x1f) are reserved for certain operation. When an interrupt occurs, it is strongly recommended that users to backup some common registers. After Compiled, WicePlus will tell users which registers have to backup from information window, Output window. In the picture below, users can see there are five C characters in the line of 0x10. These positions are 0x10, 0x11, 0x12, 0x13, 0x14. So users have to save these 5 common registers and restore them in interrupt service routine. That is users have to compile a time to know these message. Character C in the line 0x10 means C compiler occupied in other functions. Compiler will dynamic to use these in WicePlus 2.



```
-------------- RAM Data -------------   ----------- Register Data -----------   -------------- IO Data --------------
       0 1 2 3 4 5 6 7 8 9 A B C D E F          0 1 2 3 4 5 6 7 8 9 A B C D E F          0 1 2 3 4 5 6 7 8 9 A B C D
B0_2X  b d d d d d d d d d d d d d d d    0x00  - - - - - - b d d d d - - - - -    0x00  - - - - - d d - - d - - - -
B0_3X  d d d d d d d d d d d d d d d d    0x10  C C C C C - - - - - - - - - - -    0x10  - - - - - - d d d d d d d -
B1_2X  d d d d d d d d d d d d d d d d    ---------------- IND Data --------------
B1_3X  d d d d d d d d d d d d d d - -
------------- Call Depth ------------
 Depth  Interrupt    Function
```

Example 1:

```
Void _intcall INTERRUPT1_l(void) @0x08: low_int 0
{
    // backup ACC, R3, R4, R5
    _asm
    {
        MOV 0X1F, A
        SWAPA 0X4
        BS 0X4, 6// switch to ram bank 3
        BS 0X4, 7
        MOV 0X3F, A
        SWAPA 0X3
        MOV 0X3E, A
        SWAPA 0X5
        MOV 0X3D, A
        PAGE @0X0    //or Use BC to switch program page 0 if the
    }                //MCU doesn’t support page instruction
}

void _intcall INTTERRUPT1(void) @int 0
{
    // backup C system
    _asm
    {
        MOV A, 0X10 // use 2 byte C data type, C system backup
        MOV 0X3C, A //now save 0x10~0x19 to 0x3C,0X3B, 0X3A, 0X39,
        MOV A, 0X11 //0X38, 0X37 in bank 3 because switch to ram
        MOV 0X3B, A //bank 3 in _intcall INTERRUPT1_l
        MOV A, 0X12
        MOV 0X3A, A
        MOV A, 0X13
        MOV 0X39, A
        MOV A, 0X14
        MOV 0X38, A
    }
    // Write your code (inline assembly or C) here
    ……
    // restore C system
    _asm
    {
        BS 0X04, 6// switch to ram bank 3 to restore correctly
        BS 0X04, 7
        MOV A, 0X3C // use 2 byte C type, C system restore
        MOV 0X10, A
        MOV A, 0X3B
        MOV 0X11, A
        MOV A, 0X3A
        MOV 0X12, A
        MOV A, 0X39
        MOV 0X13, A
        MOV A, 0X38
        MOV 0X14, A
    }
    // restore ACC, R3, R4, R5 following backup C system
    _asm
    {
        SWAPA 0X3D //Users have to confirm whether in ram bank 3
        MOV 0X5, A  //or not. If not, have to switch to ram bank
        SWAPA 0X3E //3 to restore correctly
        MOV 0X3, A
        SWAPA 0X3F
        MOV 0X4, A
        SWAP 0X1F
        SWAPA 0X1F
    }
}
```

Example 2:

```
int nBuf[5];
void _intcall INTERRUPT2_l(void) @0x08: low_int 0
{
    // backup ACC, R3, R4, R5
    _asm
    {
        ……
    }
}
```

```
void _intcall INTERRUPT2(void) @int 0
{
    _asm //save registers
    {
        mov a, 0x10
        mov %nBuf, a
        mov a, 0x11
        mov %nBuf + 1, a
        mov a, 0x12
        mov %nBuf + 2, a
        mov a, 0x13
        mov %nBuf + 3, a
        mov a, 0x14
        mov %nBuf + 4, a
    }
    // do what you want to do as interrupt occurred.
    ……
    _asm //restore registers
    {
        mov a, %Buf
        mov 0x10, a
        mov a, %nBuf + 1
        mov 0x11, a
        mov a, %nBuf+2
        mov 0x12, a
        mov a, %nBuf + 3
        mov 0x13, a
        mov a, %nBuf + 4
        mov 0x14, a
    }
}
```

Example 3:

```
void _intcall INTERRUPT3_l(void) @0x08: low_int 0
{
    // backup ACC, R3, R4, R5
    _asm
    {
        ……
    }
}
void _intcall INTERRUPT3(void)@int 0
{
    long ans;
    ……
    ans = LongMult(0x1234, 0x5678 );
    ……
}
long LongMult(long a, long b)
{
    return (a * b);
        // multiple operation of long data type is NOT allowed!
```

Appendix A
Conversion Table

A-1 Conversion between C and Assembly Codes

The assembly code was generated by the WicePlus.

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)

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)

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)

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)

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)

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	
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;	MOV A, %e MOV 0x14, A BC 0x03, 0 RLCA 0x14 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)
	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%)

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 B

Frequently Asked Questions (FAQ)

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.