MF909-03

CMOS 4-BIT SINGLE CHIP MICROCOMPUTER **E0C63 Family**

ASSEMBLER PACKAGE MANUAL

SEIKO EPSON CORPORATION

NOTICE

No part of this material may be reproduced or duplicated in any form or by any means without the written permission of Seiko Epson. Seiko Epson reserves the right to make changes to this material without notice. Seiko Epson does not assume any liability of any kind arising out of any inaccuracies contained in this material or due to its application or use in any product or circuit and, further, there is no representation that this material is applicable to products requiring high level reliability, such as medical products. Moreover, no license to any intellectual property rights is granted by implication or otherwise, and there is no representation or warranty that anything made in accordance with this material will be free from any patent or copyright infringement of a third party. This material or portions thereof may contain technology or the subject relating to strategic products under the control of the Foreign Exchange and Foreign Trade Control Law of Japan and may require an export license from the Ministry of International Trade and Industry or other approval from another government agency. Please note that "E0C" is the new name for the old product "SMC". If "SMC" appears in other manuals understand that it now reads "E0C".

MS-DOS, Windows and Windows NT are registered trademarks of Microsoft Corporation, U.S.A.
PC/AT and IBM are registered trademarks of International Business Machines Corporation, U.S.A.
Pentium is a registered trademark of Intel Corporation.
All other product names mentioned herein are trademarks and/or registered trademarks of their respective owners.

© **SEIKO EPSON CORPORATION** 1999 All rights reserved.

Introduction

This document describes the development procedure from assembling source files to debugging. It also explains how to use each development tool of the E0C63 Family Assembler Package common to all the models of the E0C63 Family.

How To Read the Manual

This manual was edited particularly for those who are engaged in program development. Therefore, it assumes that the reader already possesses the following fundamental knowledge:
• Basic knowledge about assembler language
• Basic knowledge about the general concept of program development by an assembler
• Basic operating methods for Windows®95 or Windows NT®4.0

Before installation

See Chapter 1. Chapter 1 describes the composition of this package, and provides a general outline of each tool.

Installation

Install the tools following the installation procedure described in TBD ("Install.txt (Install.pdf)").

To understand the flow of program development

See the program development flow in Chapter 2.

For coding

See the necessary parts in Chapter 4. Chapter 4 describes the grammar for the assembler language as well as the assembler functions. Also refer to the following manuals when coding:

E0C63xxx Technical Manual
Covers device specifications, and the operation and control method of the peripheral circuits.

E0C63000 Core CPU Manual
Has the instructions and details the functions and operation of the Core CPU.

For debugging

Chapter 8 gives detailed explanation of the debugger. Sections 8.1 to 8.8 give an overview of the functions of the debugger. See Section 8.9 for details of the debug commands. Also refer to the following manuals to understand operations of the In-Circuit Emulator ICE63 and the Peripheral Circuit Board PRC63xxx:

E0C63 Family In-Circuit Emulator (ICE63) Manual
Explains the functions and handling methods of the In-Circuit Emulator ICE63.

E0C63 Family Peripheral Circuit Board (PRC63xxx) Manual
Covers the functions and handling methods of the peripheral circuit board that provides the hardware specifications of each model to the ICE63.

For details of each tool

Chapters 3 to 8 explain the details of each tool. Refer to it if necessary.

Once familiar with this package

Refer to the listings of instructions and commands contained in Appendices.

Manual Notations

This manual was prepared by following the notation rules detailed below:

(1) Sample screens

The sample screens provided in the manual are all examples of displays under Windows®95. These displays may vary according to the system or fonts used.

(2) Names of each part

The names or designations of the windows, menus and menu commands, buttons, dialog boxes, and keys are annotated in brackets [ ]. Examples: [Command] window, [File | Exit] menu item ([Exit] command in [File] menu), [Key Break] button, [q] key, etc.

(3) Names of instructions and commands

The CPU instructions and the debugger commands that can be written in either uppercase or lowercase characters are annotated in lowercase characters in this manual, except for user-specified symbols.

(4) Notation of numeric values

Numeric values are described as follows:

Decimal numbers: Not accompanied by any prefix or suffix (e. g., 123, 1000).
Hexadecimal numbers: Accompanied by the prefix "0x" (e. g., 0x0110, 0xffff).
Binary numbers: Accompanied by the prefix "0b" (e. g., 0b0001, 0b10).

However, please note that some sample displays may indicate hexadecimal or binary numbers not accompanied by any symbol. Moreover, a hexadecimal number may be expressed as xxxxh, or a binary number as xxxxb, for reasons of convenience of explanation.

(5) Mouse operations

To click: The operation of pressing the left mouse button once, with the cursor (pointer) placed in the intended location, is expressed as "to click". The clicking operation of the right mouse button is expressed as "to right-click".

To double-click: Operations of pressing the left mouse button twice in a row, with the cursor (pointer) placed in the intended location, are all expressed as "to double-click".

To drag: The operation of clicking on a file (icon) with the left mouse button and holding it down while moving the icon to another location on the screen is expressed as "to drag".

To select: The operation of selecting a menu command by clicking is expressed as "to select".

(6) Key operations

The operation of pressing a specific key is expressed as "to enter a key" or "to press a key".
A combination of keys using "+", such as [Ctrl]+[C] keys, denotes the operation of pressing the [C] key while the [Ctrl] key is held down. Sample entries through the keyboard are not indicated in [ ].
Moreover, the operation of pressing the [Enter] key in sample entries is represented by "↵".
In this manual, all the operations that can be executed with the mouse are described only as mouse operations. For operating procedures executed through the keyboard, refer to the Windows manual or help screens.

(7) General forms of commands, startup options, and messages

Items given in [ ] are those to be selected by the user, and they will work without any key entry involved.
An annotation enclosed in < > indicates that a specific name should be placed here. For example, <file name> needs to be replaced with an actual file name.
Items enclosed in { } and separated with | indicate that you should choose an item. For example, {A | B} needs to have either A or B selected.

Contents

CHAPTER 1 GENERAL 1
1.1 Features 1
1.2 Tool Composition 2
1.2.1 Composition of Package 2
1.2.2 Outline of Software Tools 2
1.3 Working Environment 3
1.4 Installation 3
1.5 Directories and Files after Installation 4

CHAPTER 2 SOFTWARE DEVELOPMENT PROCEDURE 5
2.1 Software Development Flow 5
2.2 Development Using Work Bench 6
2.2.1 Starting Up the Work Bench 6
2.2.2 Creating a New Project 7
2.2.3 Editing Source Files 7
2.2.4 Configuration of Tool Options 9
2.2.5 Building an Executable Object 10
2.2.6 Debugging 11

CHAPTER 3 WORK BENCH 12
3.1 Features 12
3.2 Starting Up and Terminating the Work Bench 12
3.3 Work Bench Windows 13
3.3.1 Window Configuration 13
3.3.2 Window Manipulation 14
3.4 Toolbar and Buttons 18
3.4.1 Standard Toolbar 18
3.4.2 Build Toolbar 19
3.4.3 Window Toolbar 19
3.4.4 Toolbar Manipulation 20
3.4.5 [Insert into project] Button on a [Edit] Window 20
3.5 Menus 21
3.5.1 [File] Menu 21
3.5.2 [Edit] Menu 22
3.5.3 [View] Menu 22
3.5.4 [Insert] Menu 23
3.5.5 [Build] Menu 23
3.5.6 [Tools] Menu 24
3.5.7 [Window] Menu 24
3.5.8 [Help] Menu 24
3.6 Project and Work Space 25
3.6.1 Creating a New Project 25
3.6.2 Inserting Sources into a Project 26
3.6.3 [Project] Window 27
3.6.4 Opening and Closing a Project 27
3.6.5 Files in the Work Space Folder 28

3.7 Source Editor ... 29
3.7.1 Creating a New Source or Header File ... 29
3.7.2 Loading and Saving Files ... 30
3.7.3 Edit Function ... 31
3.7.4 Tag Jump Function ... 34
3.7.5 Printing ... 35
3.8 Build Task ... 35
3.8.1 Preparing a Build Task ... 35
3.8.2 Building an Executable Object ... 35
3.8.3 Debugging ... 36
3.8.4 Executing Other Tools ... 37
3.9 Tool Option Settings ... 39
3.9.1 Assembler Options ... 39
3.9.2 Linker Options ... 40
3.9.3 Debugger Options ... 42
3.9.4 HEX Converter Options ... 42
3.10 Work Bench Options ... 43
3.11 Short-Cut Key List ... 44
3.12 Error Messages ... 44
3.13 Precautions ... 45

CHAPTER 4 ASSEMBLER ... 46
4.1 Functions ... 46
4.2 Input/Output Files ... 46
4.2.1 Input File ... 46
4.2.2 Output Files ... 47
4.3 Starting Method ... 48
4.4 Messages ... 49
4.5 Grammar of Assembly Source ... 50
4.5.1 Statements ... 50
4.5.2 Instructions (Mnemonics and Pseudo-instructions) ... 52
4.5.3 Symbols (Labels) ... 53
4.5.4 Comments ... 55
4.5.5 Blank Lines ... 55
4.5.6 Register Names ... 56
4.5.7 Numerical Notations ... 56
4.5.8 Operators ... 57
4.5.9 Location Counter Symbol "$" ... 59
4.5.10 Optimization Branch Instructions for Old Preprocessor ... 59
4.6 Section Management ... 60
4.6.1 Definition of Sections ... 60
4.6.2 Absolute and Relocatable Sections ... 60
4.6.3 Sample Definition of Sections ... 61
4.7 Assembler Pseudo-Instructions ... 62
4.7.1 Include Instruction (#include) ... 63
4.7.2 Define Instruction (#define) ... 64
4.7.3 Numeric Define Instruction (#defnum) ... 66
4.7.4 Macro Instructions (#macro ... #endm) ... 67
4.7.5 Conditional Assembly Instructions
(#ifdef ... #else ... #endif, #ifndef... #else ... #endif) ... 69
4.7.6 Section Defining Pseudo-Instructions (.code, .data, .bss) ... 71
4.7.7 Location Defining Pseudo-Instructions (.org, .align) ... 73
4.7.8 Absolute Assembling Pseudo-Instruction (.abs) ... 76
4.7.9 Symbol Defining Pseudo-Instruction (.set) ... 77

4.7.10 Data Defining Pseudo-Instructions (.codeword, .word) 78
4.7.11 Area Securing Pseudo-Instructions (.comm, .lcomm) 79
4.7.12 Global Declaration Pseudo-Instruction (.global) 80
4.7.13 List Control Pseudo-Instructions (.list, .nolist) 80
4.7.14 Source Debugging Information Pseudo-Instructions (.stabs, .stabn) 80
4.7.15 Comment Adding Function 81
4.7.16 Priority of Pseudo-Instructions 81

4.8 Relocatable List File 82

4.9 Sample Executions 83

4.10 Error/Warning Messages 86
4.10.1 Errors 86
4.10.2 Warning 87

4.11 Precautions 87

CHAPTER 5 LINKER 88

5.1 Functions 88

5.2 Input/Output Files 88
5.2.1 Input Files 88
5.2.2 Output Files 89

5.3 Starting Method 90

5.4 Messages 93

5.5 Linker Command File 94

5.6 Link Map File 95

5.7 Symbol File 96

5.8 Absolute List File 97

5.9 Cross Reference File 98

5.10 Linking 99

5.11 Branch Optimization Function 101

5.12 Error/Warning Messages 102
5.12.1 Errors 102
5.12.2 Warning 102

5.13 Precautions 103

CHAPTER 6 HEX CONVERTER 104

6.1 Functions 104

6.2 Input/Output Files 104
6.2.1 Input Files 104
6.2.2 Output Files 104

6.3 Starting Method 105

6.4 Messages 106

6.5 Output Hex Files 107
6.5.1 Hex File Configuration 107
6.5.2 Motorola-S Format 107
6.5.3 Intel-HEX Format 108
6.5.4 Conversion Range 108

6.6 Error/Warning Messages 109
6.6.1 Errors 109
6.6.2 Warning 109

6.7 Precautions 109

CHAPTER 7 *DISASSEMBLER* *110*

7.1 Functions 110

7.2 Input/Output Files 110

7.2.1 Input Files 110

7.2.2 Output Files 110

7.3 Starting Method 111

7.4 Messages 112

7.5 Disassembling Output 113

7.6 Error/Warning Messages 116

7.6.1 Errors 116

7.6.2 Warning 116

CHAPTER 8 *DEBUGGER* *117*

8.1 Features 117

8.2 Input/Output Files 117

8.2.1 Input Files 117

8.2.2 Output Files 118

8.3 Starting Method 119

8.3.1 Start-up Format 119

8.3.2 Start-up Options 119

8.3.3 Start-up Messages 120

8.3.4 Hardware Check at Start-up 120

8.3.5 Method of Termination 122

8.4 Windows 123

8.4.1 Basic Structure of Window 123

8.4.2 [Command] Window 125

8.4.3 [Source] Window 126

8.4.4 [Data] Window 128

8.4.5 [Register] Window 128

8.4.6 [Trace] Window 129

8.5 Tool Bar 130

8.5.1 Tool Bar Structure 130

8.5.2 [Key Break] Button 130

8.5.3 [Load File] and [Load Option] Buttons 130

8.5.4 [Source], [Mix], and [Unassemble] Buttons 130

8.5.5 [Go], [Go to Cursor], [Go from Reset], [Step], [Next], and [Reset] Buttons 130

8.5.6 [Break] Button 131

8.5.7 [Help] Button 131

8.6 Menu 132

8.6.1 Menu Structure 132

8.6.2 [File] Menu 132

8.6.3 [Run] Menu 132

8.6.4 [Break] Menu 133

8.6.5 [Trace] Menu 133

8.6.6 [View] Menu 134

8.6.7 [Option] Menu 134

8.6.8 [Windows] Menu 134

8.6.9 [Help] Menu 134

8.7 Method for Executing Commands 135

8.7.1 Entering Commands from Keyboard 135

8.7.2 Executing from Menu or Tool Bar 137

8.7.3 Executing from a Command File 138

8.7.4 Log File 139

8.8 Debug Functions 140
8.8.1 Loading Program and Data Files 140
8.8.2 Source Display and Symbolic Debugging Function 141
8.8.3 Displaying and Modifying Program, Data, Option Data and Register 143
8.8.4 Executing Program 145
8.8.5 Break Functions 148
8.8.6 Trace Functions 151
8.8.7 Operation of Flash Memory 154
8.8.8 Coverage 155
8.9 Command Reference 156
8.9.1 Command List 156
8.9.2 Reference for Each Command 157
8.9.3 Program Memory Operation 158
a / as (assemble mnemonic) 158
pe (program memory enter) 160
pf (program memory fill) 161
pm (program memory move) 162
8.9.4 Data Memory Operation 163
dd (data memory dump) 163
de (data memory enter) 165
df (data memory fill) 167
dm (data memory move) 168
dw (data memory watch) 169
8.9.5 Command to Display Option Information 171
od (option data dump) 171
8.9.6 Register Operation 173
rd (register display) 173
rs (register set) 174
8.9.7 Program Execution 176
g (go) 176
gr (go after reset CPU) 178
s (step) 179
n (next) 181
8.9.8 CPU Reset 182
rst (reset CPU) 182
8.9.9 Break 183
bp (break point set) 183
bc / bpc (break point clear) 185
bd (data break) 186
bdc (data break clear) 188
br (register break) 189
brc (register break clear) 191
bs (sequential break) 192
bsc (sequential break clear) 194
bsp (break stack pointer) 195
bl (break point list) 197
bac (break all clear) 198
8.9.10 Program Display 199
u (unassemble) 199
sc (source code) 201
m (mix) 203
8.9.11 Symbol Information 205
sy (symbol list) 205
8.9.12 Load File 206
lf (load file) 206
lo (load option) 207
8.9.13 Flash Memory Operation 208
lfl (load from flash memory) 208
sfl (save to flash memory) 210
efl (erase flash memory) 212

8.9.14 Trace *213*
tm (trace mode) *213*
td (trace data display) *215*
ts (trace search) *218*
tf (trace file) *220*
8.9.15 Coverage *221*
cv (coverage) *221*
cvc (coverage clear) *222*
8.9.16 Command File *223*
com (execute command file) *223*
cmw (execute command file with wait) *224*
rec (record commands to a file) *225*
8.9.17 log *226*
log (log) *226*
8.9.18 Map Information *227*
ma (map information) *227*
8.9.19 Mode Setting *228*
md (mode) *228*
8.9.20 Quit *231*
q (quit) *231*
8.9.21 Help *232*
? (help) *232*
8.10 Status/Error/Warning Messages *233*

CHAPTER 1 GENERAL

1.1 Features

The E0C63 Family Assembler Package contains software development tools that are common to all the models of the E0C63 Family. The package comes as an efficient working environment for development tasks, ranging from source program assembly to debugging.
Its principal features are as follows:

Simple composition
A task from assembly to debugging can be made with minimal tools.

Integrated working environment
A Windows-based integrated environment allows the tool chain to be used on its Windows GUI interface.

Modular programming
The relocatable assembler lets you develop a program which is made up of multiple sources. This makes it possible to keep a common part independently and to use it as a part or a basis for the next program.

Source debugging
A debugger can display an assembler source to show its execution status and allow debugging operations on it. This makes debugging much easier to perform.

Common to all E0C63 chips
The tools (workbench, assembler, linker, hex converter, disassembler, and debugger) are common to all E0C63 Family models except for several chip dependent masking tools ("Dev" tools). The chip dependent information is read from the ICE parameter file for each chip.

Complete compatibility with old syntax sources
By supporting old syntax, existing sources written for old 63 tools are available with these new tools.

1.2 Tool Composition

1.2.1 Composition of Package

The E0C63 Family Assembler Package contains the items listed below. When it is unpacked, make sure that all items are supplied.

1) CD-ROM .... One
2) Warranty card .... One each in English and Japanese

1.2.2 Outline of Software Tools

The following shows the outlines of the software tools included in the package:

Assembler (as63.exe)

Converts the mnemonic of the source files into object codes (machine language) of the E0C63000. The results are output in a relocatable object file. This assembler includes preprocessing functions such as macro definition/call, conditional assembly, and file-include functions.

Linker (lk63.exe)

Links the relocatable objects created by the assembler by fixing the memory locations, and creates executable absolute object codes. The linker also provides an auto EXT insertion/correction function allowing the programmer to create sources without having to know branch destination ranges.

Hex converter (hx63.exe)

Converts an absolute object in IEEE-695 format output from the linker into ROM-image data in Motorola-S format or Intel-HEX format. This conversion is needed when making the ROM or when creating mask data using the mask data checker.

Disassembler (ds63.exe)

Disassembles an absolute object file in IEEE-695 format or a hex file in Motorola-S format, and restores it to a source format file. The restored source file can be processed in the assembler/linker/hex converter to obtain the same object or hex file.

Debugger (db63.exe)

This software performs debugging by controlling the ICE63 hardware tool. Commands that are used frequently, such as break and step, are registered on the tool bar, minimizing the necessary keyboard operations. Moreover, sources, registers, and command execution results can be displayed in multiple windows, with resultant increased efficiency in the debugging tasks.

Work Bench (wb63.exe)

This software provides an integrated development environment with Windows GUI. Creating/editing source files, selecting files and major start-up options, and the start-up of each tool can be made with simple Windows operations.

1.3 Working Environment

To use the E0C63 Family Assembler Package, the following conditions are necessary:

Personal computer

An IBM PC/AT or a compatible machine which is equipped with a CPU equal to or better than a Pentium 75 MHz, and 32MB or more of memory is recommended.
To use the optional In-Circuit Emulator ICE63, the personal computer also requires a serial port (with a D-sub 9 pin).

Display

A display unit capable of displaying 800 × 600 dots or more is necessary.

Hard disk and CD-ROM drive

Since the installation is done from a CD-ROM to a hard disk, a CD-ROM drive and a hard disk drive are required.

Mouse

A mouse is necessary to operate the tools.

System software

The E0C63 Family Assembler Package supports Microsoft® Windows®95 (English or Japanese), Windows®98 (English or Japanese) and Windows NT®4.0 (English or Japanese).

Other development tools

To debug the target program, the optional In-Circuit Emulator ICE63 and a Peripheral Circuit Board PRC63xxx are needed as the hardware tools.
The PRC63xxx board is prepared for each E0C63 model.

1.4 Installation

The supplied CD-ROM contains the installer (Setup.exe) that installs the tools.
To install the tools, start up the "Setup.exe" and follow the instructions in the dialog boxes that will be appeared. For more information on the installation procedure, please refer to "setup_e.pdf" on the CD-ROM.

1.5 Directories and Files after Installation

The installer copies the following files in the specified directory (default is "C:\E0C63\"):

```
[Specified folder]
    README.TXT          ... ReadMe document

    /bin
        WB63.EXE        ... Work bench
        AS63.EXE        ... Assembler
        LK63.EXE        ... Linker
        HX63.EXE        ... Hex converter
        DS63.EXE        ... Disassembler
        DB63.EXE        ... Debugger
        E0C63.CNT       ... Help index
        E0C63.HLP       ... Help contents
        CORE63.DLL      ... Core class library for debugger
        ICE63.DLL       ... ICE control/communication module for debugger
        IEEE695.DLL     ... Object format library for debugger
        HEXLIB.DLL      ... Hex file library for debugger
        AS63.DLL        ... Inline assembler module for debugger
        SPAWNEX.EXE     ... Child task library for work bench
        OLEPRO32.DLL    ... OLE library for work bench
        MSVCRT.DLL      ... Run time library for work bench

    /doc
        MANUAL_E.PDF    ... E0C63 Family Assembler Package Manual in PDF format
        QUICK_E.PDF     ... Quick Reference in PDF format
        63xxx.PDF       ... E0C63xxx Development Tool Manual in PDF format
            :               (for the model selected at installation)

    /dev63
        /DEV63xxx       ... Selected E0C63 development tool for each chip type
            :
```

Online manual in PDF format

The online manuals are provided in PDF format, so Adobe Acrobat Reader Ver. 3.0 or later is needed to read it. The English version and Japanese version of Acrobat Readers are included in the CD-ROM (\Acrobat). To install it, run its set up program (\Acrobat\ar32eXXX.exe for English, \Acrobat \ar32jXXX.exe for Japanese). Acrobat Reader can be installed any time before or after the installation of the E0C63 tools.

CHAPTER 2 SOFTWARE DEVELOPMENT PROCEDURE

This chapter outlines a basic development procedure.

2.1 Software Development Flow

Figure 2.1.1 represents a flow of software development work.

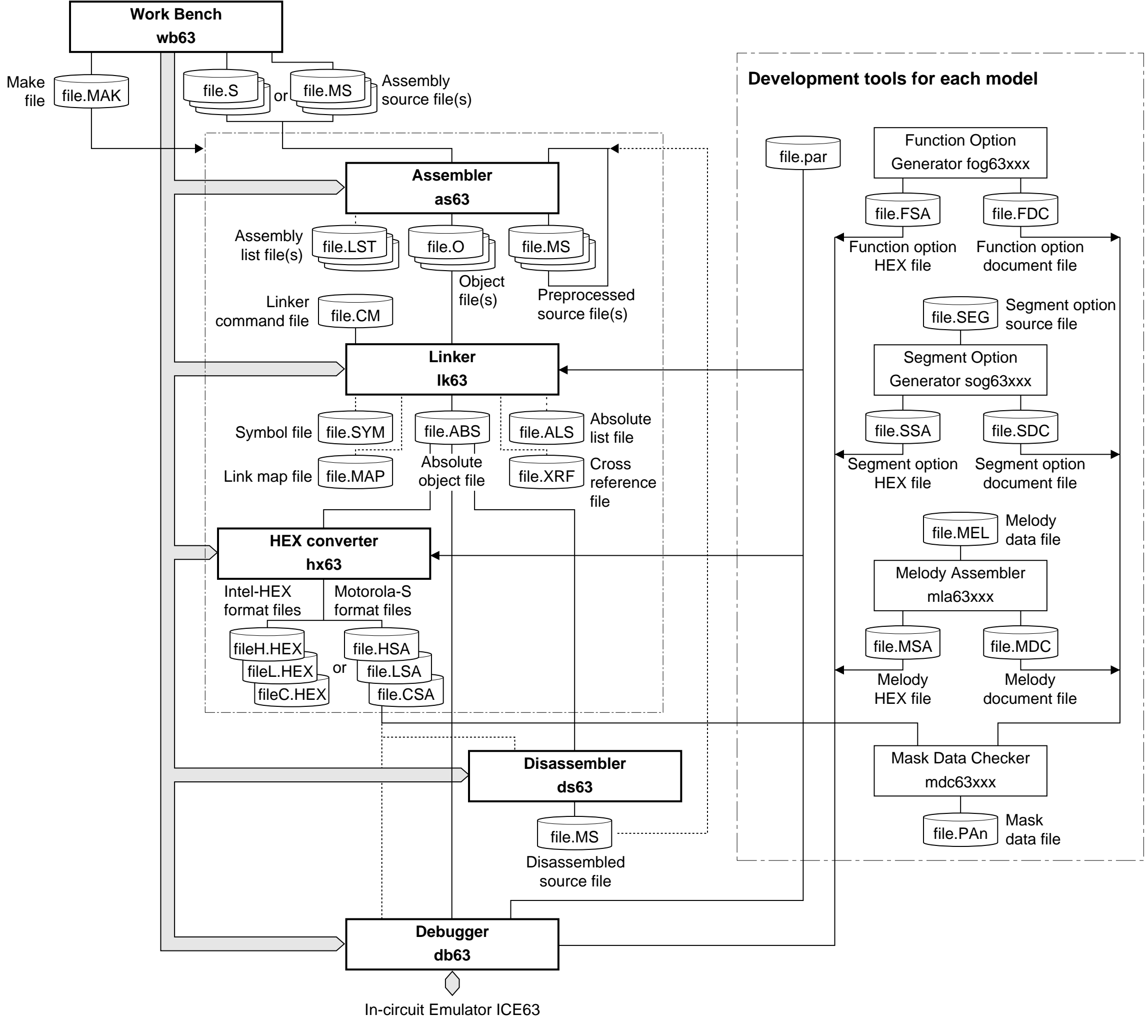


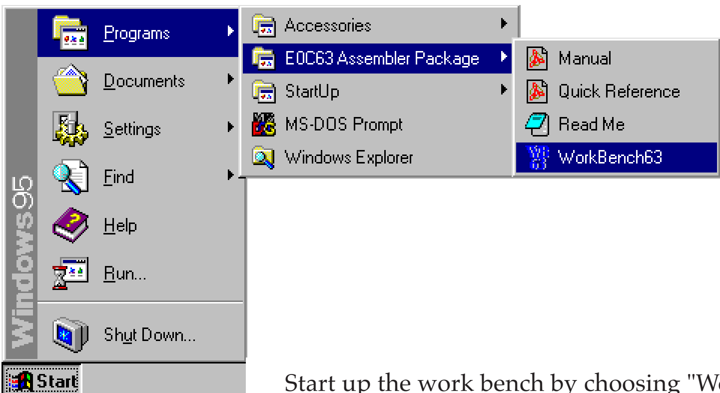
Fig. 2.1.1 Software development flow

The work bench provides an integrated development environment from source editing to debugging. Tools such as the assembler and linker can be invoked from the work bench. The tools can also be invoked individually from the DOS prompt.
Refer to the respective chapter for details of each tool.
The part indicated as "Development tools for each model" is not covered in this manual. For details, refer to the tool manual associated with each specific model.

2.2 Development Using Work Bench

This section shows a basic development procedure using the work bench wb63.
Refer to Chapter 3, "Work Bench", for operation details.

2.2.1 Starting Up the Work Bench



Start up the work bench by choosing "WorkBench63" from the program menu.

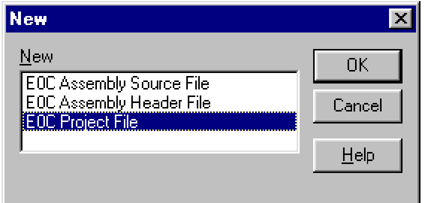
2.2.2 Creating a New Project

The work bench manages necessary file and tool setting information as a project.
First a new project file should be created.

1. Select [New] from the [File] menu (or click the [New] button).

[New] button

The [New] dialog box appears.



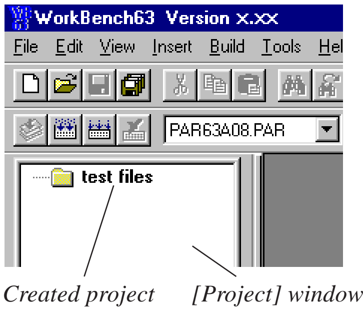
2. Select [E0C Project File] and click [OK].

The [Project] dialog box appears.

3. Enter a project name, select an ICE parameter file and select a directory, then click [OK].

* The [ICE parameter file:] box lists the parameter files that exist in the "dev63" directory.

The work bench creates a folder (directory) with the specified project name as a work space, and puts the project file (.epj) into the folder.
The specified project name will also be used for the absolute object and other files.



Created project *[Project] window*

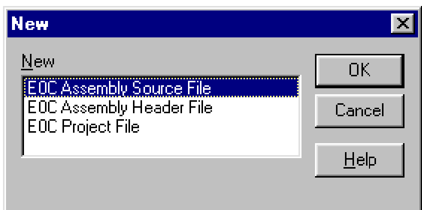
2.2.3 Editing Source Files

The work bench has an editor function. This makes it possible to edit source files without another editor.
To create a new source file:

1. Select [New] from the [File] menu (or click the [New] button).

[New] button

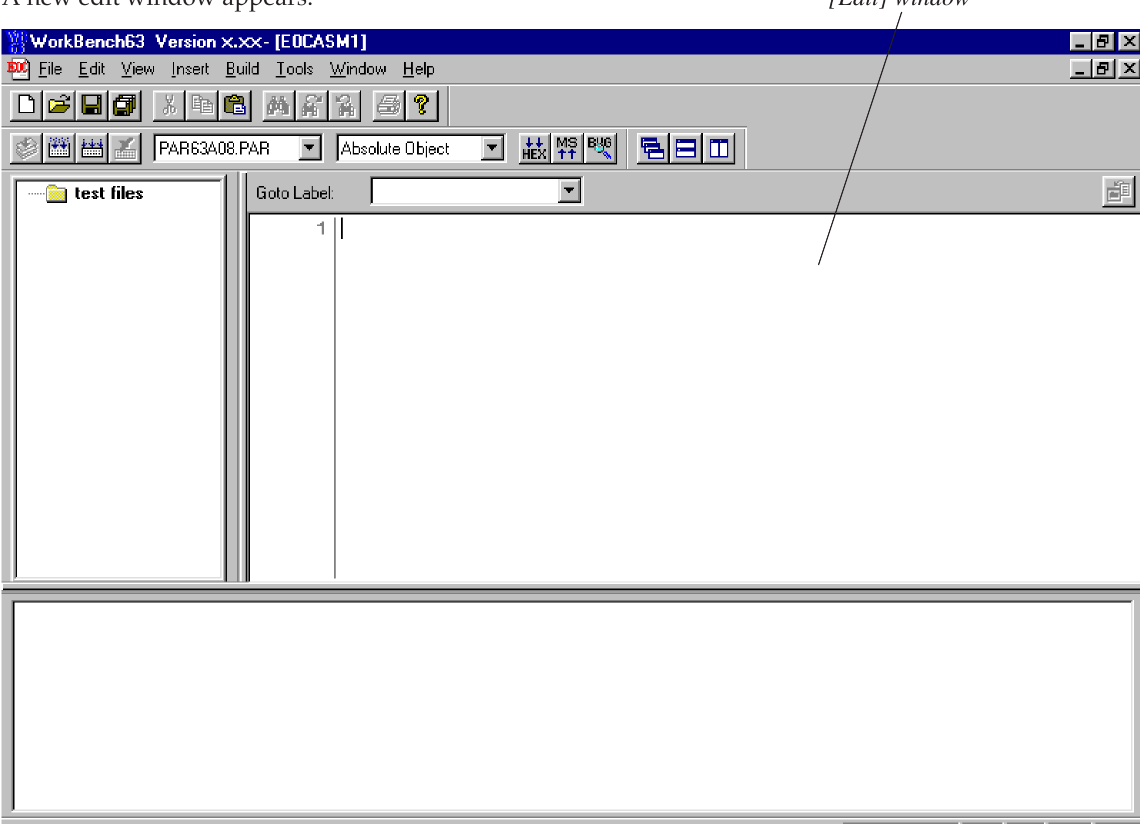
The [New] dialog box appears.



2. Select [E0C Assembly Source File] and click [OK].

A new edit window appears.

[Edit] window



3. Enter source codes in the [Edit] window.

Goto Label:

```
 1 ; main.s
 2 ; ASM63 test program (main routine)
 3 ;
 4
 5 ;***** INITIAL SP1 & SP2 ADDRESS DEFINITION *****
 6
 7 #ifdef SMALL_RAM
 8         .set SP1_INIT_ADDR 0xb          ;SP1 init addr = 0x2c
 9 #else
10         .set SP1_INIT_ADDR 0x4b         ;SP1 init addr = 0x12c
11 #endif
12
13         .set SP2_INIT_ADDR 0x1f         ;SP2 init addr = 0x1f
14
15
16 ;***** NMI & BOOT, LOOP *****
17
```

4. Save the source in a file by selecting [Save] from the [File] menu (or clicking the [Save] button).

[Save] button

5. Click the [Insert into project] button on the [Edit] window.

[Insert into project] button

The created source file is added in the project.

To add existing source files, use [Files into project...] in the [Insert] menu. It can also be done by dragging source files from Windows Explorer to the project window.
Create necessary source files and add them into the project.

Sample list in the [Project] window

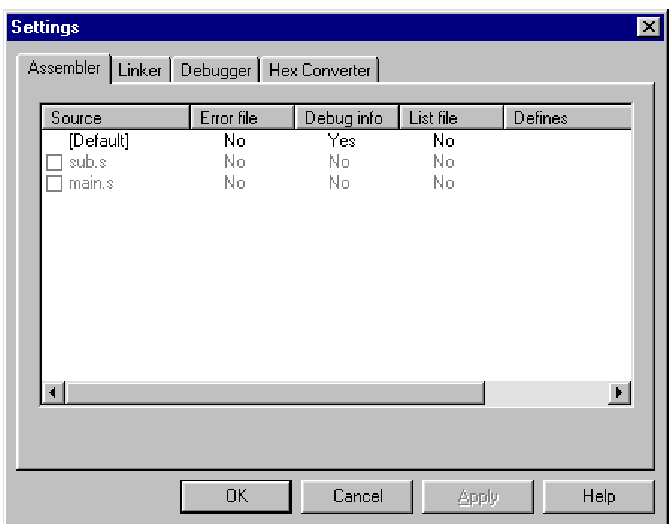
The added source files are listed in the project window. Double-clicking a listed source file name opens the edit window.

2.2.4 Configuration of Tool Options

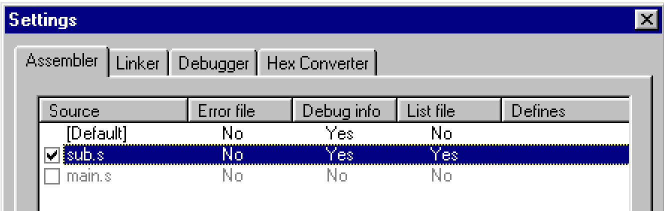
The work bench supports all the start up options of each tool and they can be selected in a dialog box. A make process for generating an executable object will be configured based on the settings.
In addition to option selection, command files for the linker and debugger can be configured here.
To set tool options:

1. Select [Setting...] from the [Build] menu.

   A dialog box appears.



2. Configure options if necessary.
   Check box items can be selected by clicking. Items in the list can be toggled or entered by double-clicking.



Refer to Chapter 3, "Work Bench", for details of the [Settings] dialog box.

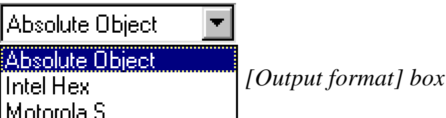
2.2.5 Building an Executable Object

To make an executable object file:

1. Select [Build] from the [Build] menu (or click the [Build] button).

[Build] button

This will invoke the assembler and linker to create an executable object file. If a HEX file format (Intel HEX or Motorola S) is selected by the [Output format] box, the HEX converter will be invoked after linking. By default, an absolute object file in IEEE-695 format will be created.



[Output format] box

Messages delivered from each executed tool are displayed in the [Output] window. The work bench has a tag-jump function that jumps to the source line in which an error has occurred by double-clicking a source syntax error message that appears in the [Output] window. It opens the corresponding source window if it is closed.

Linked with the corresponding source line

[Output] window

In the build task, a general make process is executed to update the least necessary files. To rebuild all the files without the make function, select [Rebuild All] from the [Build] menu (or click the [Rebuild All] button).

[Rebuild All] button

To invoke the assembler only to correct syntax errors, select [Assemble] in the [Built] menu (or click the [Assemble] button).

[Assemble] button

2.2.6 Debugging

To debug the executable object:

1. Select [Debug] from the [Build] menu (or click the [Debug] button).

[Debug] button

The debugger starts up with the specified ICE parameter file and then loads the executable object file.

Note: Make sure that the ICE63 is ready to debug before invoking the debugger. Refer to the "ICE Hardware Manual" for settings and startup method of the ICE63.

```
Db63 - Command
File  Run  Break  Trace  View  Option  Window  Help

Mix
Search Label:
     Addr  Code  Unassemble                       Source
 57  0110  094b  ldb  ba,4b             ldb    %ba,SP1_INIT_ADDR
 58  0111  1fc4  ldb  sp1,ba            ldb    %sp1,%ba
 59  0112  091f  ldb  ba,1f             ldb    %ba,SP2_INIT_ADDR
 60  0113  1fc6  ldb  sp2,ba            ldb    %sp2,%ba
 61  0114  08fe  ldb  ext,fe    (+)     ldb  ext,fe
 62  0115  02ea  calr ea                calr   INIT_RAM_BLK1
 63                                LOOP:
 64  0116  08fe  ldb  ext,fe    (+)     ldb  ext,fe
 65  0117  02ef  calr ef                calr   INC_RAM_BLK1
 66  0118  00fd  jr   fd                jr     LOOP
 67

Reg
PC    :0110
A     :A
B     :A
X     :[AAAA] = *
Y     :[AAAA] = *
EICZ  :0000
SP1   :AA
SP2   :AA
EXT   :AA
QUEUE:AAAA
0000000000 cycle
[0000] = AAAA
[0004] = AAAA
[0008] = AAAA

Command
LCD board version         : 1.0
EXT board version         : No use ..... warning : chec
ICE hardware version      : 1.0
ICE software version      : 1.0
DIAG test                 : omitted
Map ............................ done
Initialize ..................... done
>com "C:\E0C63\Test\test.cmd" 0
>lf "test.abs"
Loading file...OK!
>

Data
ADDR: 0 1 2 3 4 5 6 7  8 9 A B C D E F
0000  A A A A A A A A  A A A A A A A A
0010  A A A A A A A A  A A A A A A A A
0020  A A A A A A A A  A A A A A A A A
0030  A A A A A A A A  A A A A A A A A
0040  A A A A A A A A  A A A A A A A A
0050  A A A A A A A A  A A A A A A A A
0060  A A A A A A A A  A A A A A A A A
0070  A A A A A A A A  A A A A A A A A
0080  A A A A A A A A  A A A A A A A A
0090  A A A A A A A A  A A A A A A A A

Ready                                              NUM
```

For the debugging functions and operations, refer to Chapter 8, "Debugger".

CHAPTER 3 WORK BENCH

This chapter describes the functions and operating method of the Work Bench wb63.

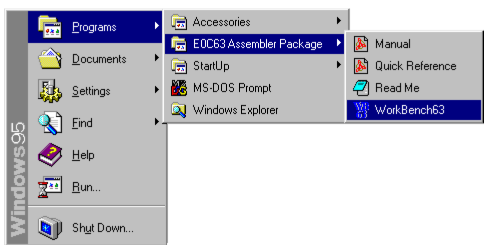
3.1 Features

The Work Bench wb63 provides an integrated operating environment ranging from editing source files to debugging. Its functions and features are summarized below:

- Source edit function that supports copy/paste, find/replace, print, label jump and tag jump from error messages.
- Allows simple management of all necessary files and information as a project.
- General make process to invoke necessary tools and to update the least necessary files.
- Supports all options of the assembler, linker, HEX converter, disassembler and debugger.
- Windows GUI interface for simple operation.

3.2 Starting Up and Terminating the Work Bench

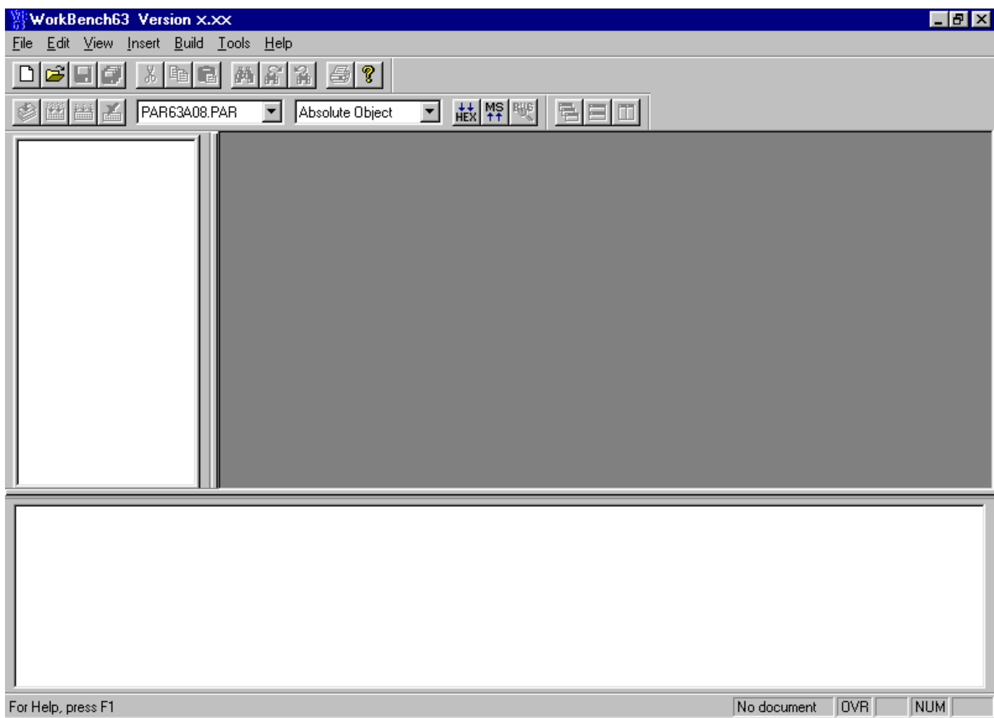
To start up the work bench



Choose "WorkBench63" from the [Program] menu to start up the work bench.

* If "WorkBench63" is not registered in the [Program] menu, it means that the installation was not successful. Therefore, reinstall the tools .

When the work bench starts up, the window shown below appears.

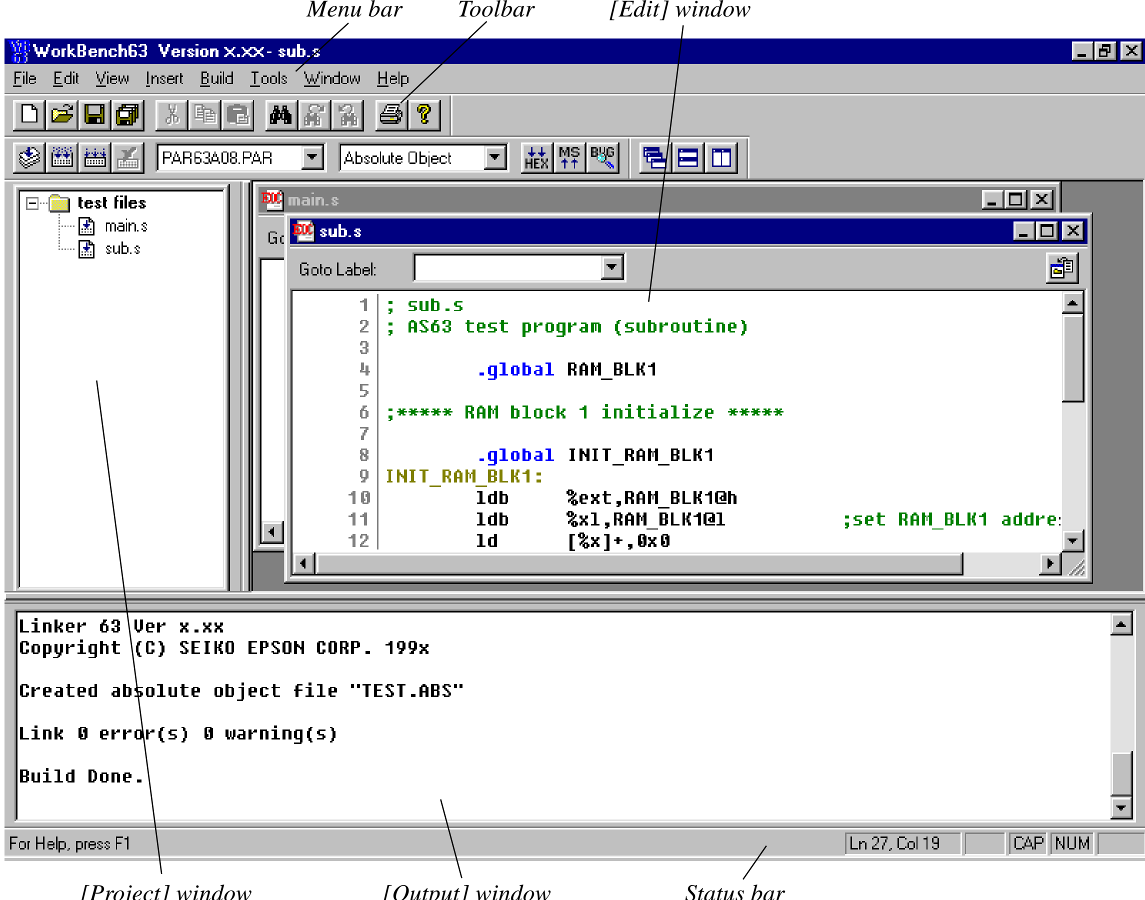


To terminate the work bench

Select [Exit] from the [File] menu.

3.3 Work Bench Windows

3.3.1 Window Configuration



The work bench has three types of windows: [Edit] window, [Project] window and [Output] window.

[Edit] window

This window is used for editing a source file. A standard text file can also be displayed in this window. Two or more windows can be opened in the edit window area.

When an E0C63 assembly source file is opened, the source is displayed with in colors according to the contents. The default colors are shown below.

E0C63 instructions:	Black
Preprocess (#) pseudo-instructions:	Dark brown
Assemble (.) pseudo-instructions:	Blue
Labels:	Light brown
Comments:	Green

These colors can be changed by the [Tools | Options] menu command (refer to Section 3.10).

[Project] window

This window shows the currently opened work space folder and lists all the source files in the project, with a structure similar to Windows Explorer.

Double-clicking a source file icon opens the source file in the [Edit] window.

[Output] window

This window displays the messages delivered from the executed tools in a build or assemble process. Double-clicking a syntax error message with a source line number displayed in this window activates or opens the [Edit] window of the corresponding source so that the source line in which the error has occurred can be viewed.

Menu bar

Refer to Section 3.5.

Toolbar

Refer to Section 3.3.

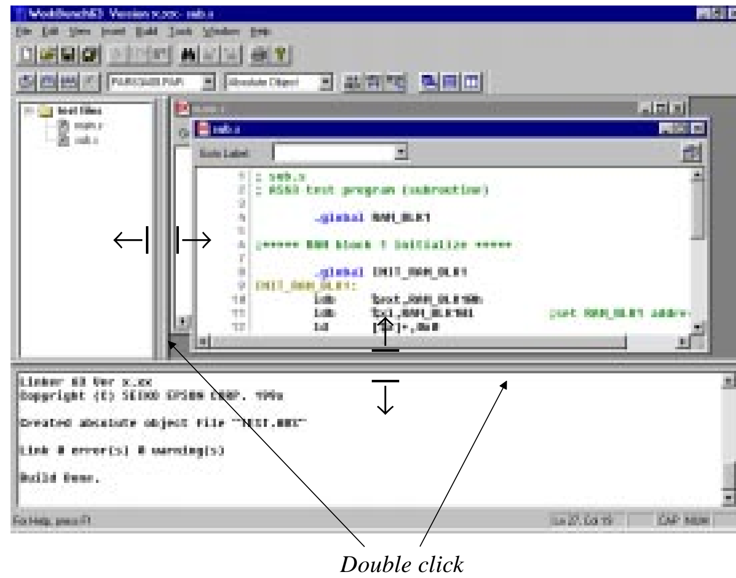
Status bar

Shows help messages when the mouse cursor is placed on a menu item or a button.
It also indicates the cursor position in the [Edit] window and Key lock status (Num lock, Caps lock, Scroll lock).

3.3.2 Window Manipulation

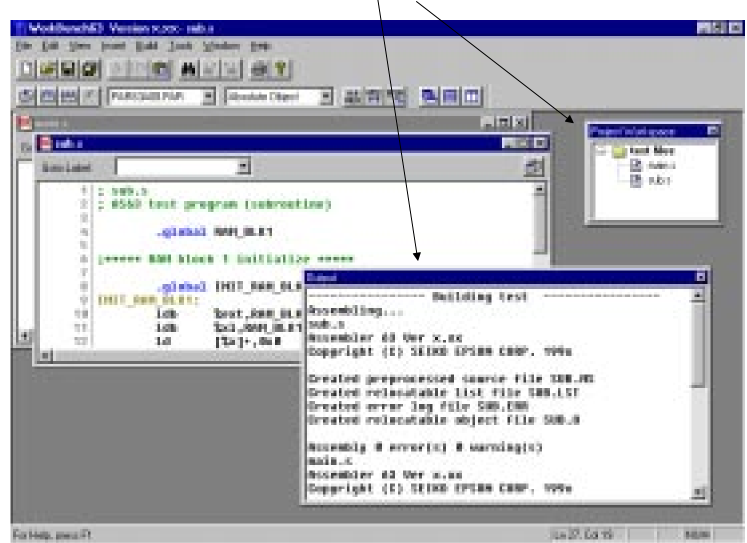
Resizing the windows

Each window area can be resized by dragging the window boundary. The size information is saved when the work bench is terminated. So the same window layout will appear at the next time the work bench starts up.



Floating and docking the [Project] and [Output] window

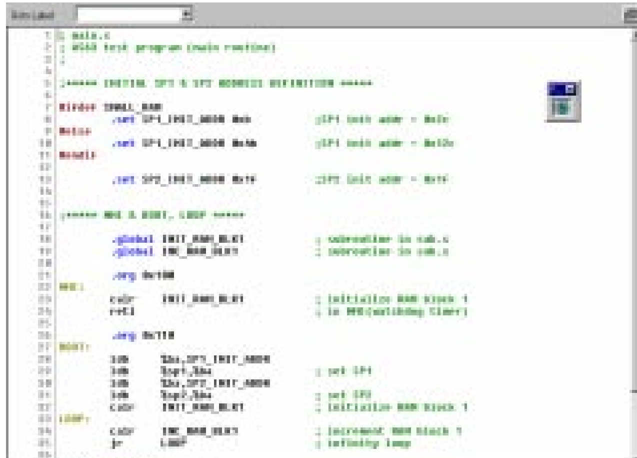
The [Project] window and the [Output] window can be made a floating window by double-clicking the window boundary and the floating window can be moved and resized in the work bench window. The floating window will be restored to a docking window by double clicking the window's title bar or dragging the title bar towards an edge of the work bench window.



Closing the [Project] and [Output] window

The [Project] window and the [Output] window can be closed by selecting [Project Window] and [Output Window] from the [View] menu, respectively. To open them, select the menu items again.

Maximizing the [Edit] window area



The [Edit] window area can be maximized to the full screen size by selecting [Full Screen] from the [View] menu. All other windows and toolbars are hidden behind the [Edit] window area.
To return it to the normal display, click the button that appears on the screen. This button can be moved anywhere in the screen by dragging its title bar. Pressing the [ESC] key also returns the window to the normal display.

Opening/Closing [Edit] windows

An [Edit] window opens when a source file (text file) is loaded using a menu, button or a file icon in the [Project] window, or when a new source is created.
[Edit] windows close by clicking the [Close] box of each window or selecting [Close] from the [File] menu.
When a project file is saved, the [Edit] window information (files opened, size and location) is also saved. So the next time the project opens, editing can begin in the saved condition.

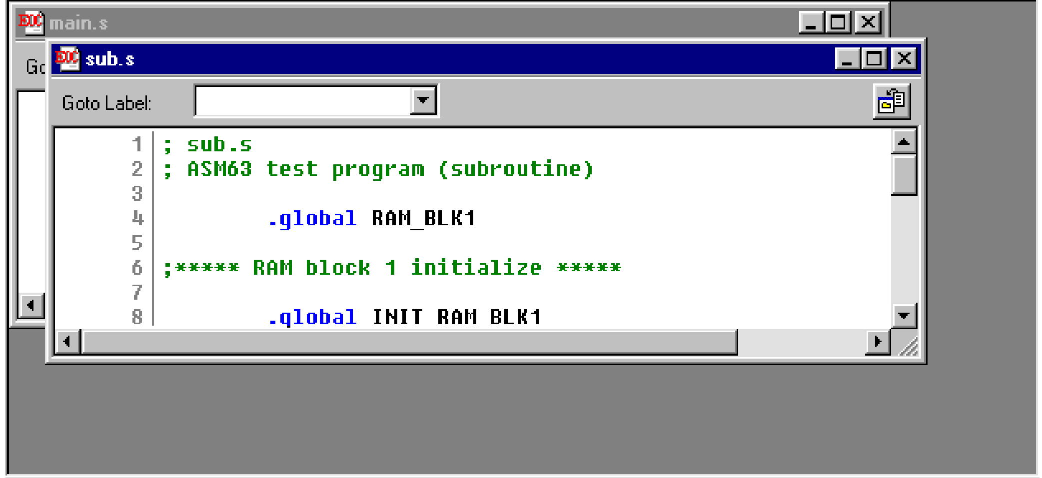
Arrangement of the [Edit] windows

The [Edit] windows being opened can be arranged similar to standard Windows applications.

1 Cascade windows

Select [Cascade] from the [Window] menu or click the [Cascade Windows] button.

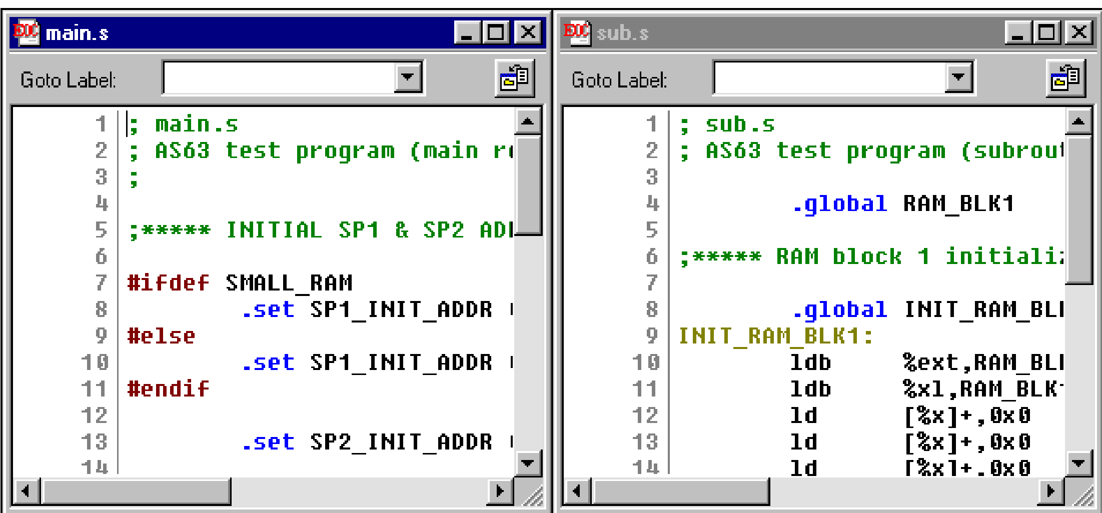
[Cascade Windows] button



2 *Tile windows*

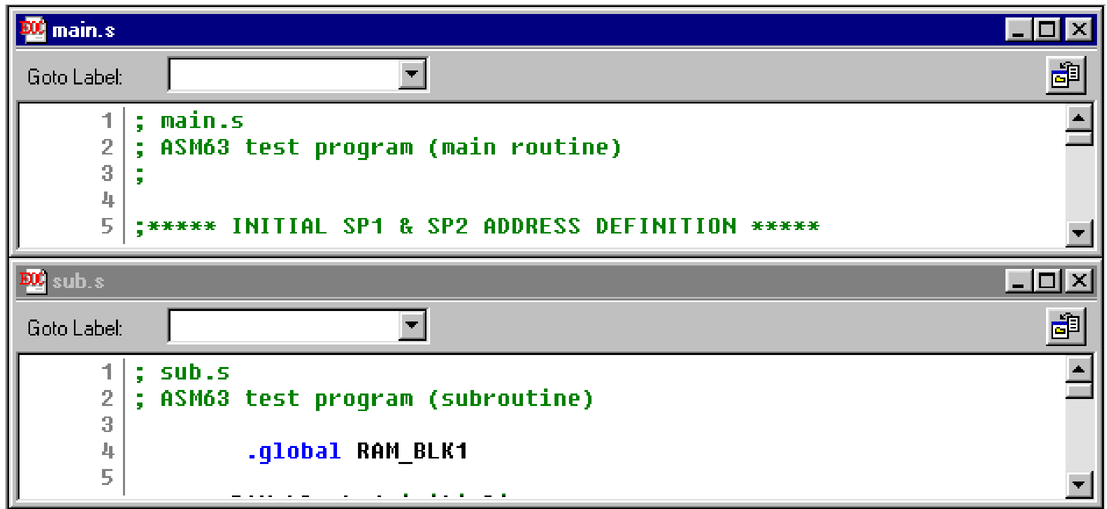
To tile windows vertically, select [Tile Vertically] from the [Window] menu or click the [Tile Vertically] button.

[Tile Vertically] button



To tile windows horizontally, select [Tile Horizontally] from the [Window] menu or click the [Tile Horizontally] button.

[Tile Horizontally] button



3 *Maximizing an [Edit] window*

Click the [Maximize] button on the window title bar. The window will be maximized to the [Edit] window area size and other [Edit] windows will be hidden behind the active window.

```
Goto Label:
 1  ; main.s
 2  ; ASM63 test program (main routine)
 3  ;
 4
 5  ;***** INITIAL SP1 & SP2 ADDRESS DEFINITION *****
 6
 7  #ifdef SMALL_RAM
 8          .set SP1_INIT_ADDR 0xb          ;SP1 init addr = 0x2c
 9  #else
10          .set SP1_INIT_ADDR 0x4b         ;SP1 init addr = 0x12c
11  #endif
12
13          .set SP2_INIT_ADDR 0x1f         ;SP2 init addr = 0x1f
14
15
16  ;***** NMI & BOOT, LOOP *****
17
```

4 *Minimizing an [Edit] window*

Click the [Minimize] button on the window title bar. The window will be minimized as a window icon. The minimized icons can be arranged at the bottom of the [Edit] window area by selecting [Arrange Icons] from the [Window] menu.

5 *Moving and resizing an [Edit] window*

The [Edit] window allows changing of its location and its size in the same way as the standard Windows applications if it is not maximized.

Switching active [Edit] window

Click the window to be activated if it can be viewed. Otherwise, select the window name (source file name) from the currently-opened window list in the [Window] menu.

Scrolling display contents

A standard scroll bar appears if the display contents exceed the display size of a window. Use it to scroll the display contents. The arrow keys can also be used.

Showing and hiding the status bar

The status bar can be shown or hidden by selecting [Status Bar] from the [View] menu.

3.4 Toolbar and Buttons

Tree types of toolbars have been implemented in the work bench: standard toolbar, build toolbar and window tool bar.

Standard toolbar

PAR63A08.PAR Absolute Object

Build toolbar *Window toolbar*

3.4.1 Standard Toolbar

This toolbar has the following standard buttons:

[New] button

Creates a new document. A dialog box will appear allowing selection from among three document types: E0C63 assembly source, E0C63 assembly header and project.

[Open] button

Opens a document. A dialog box will appear allowing selection of the file to be opened.

[Save] button

Saves the document in the active [Edit] window to the file. The file will be overwritten.
This button becomes inactive if no [Edit] window is opened.

[Save All] button

Saves the documents of all [Edit] windows and the project information to the respective files.

[Cut] button

Cuts the selected text in the [Edit] window to the clipboard.

[Copy] button

Copies the selected text in the [Edit] window to the clipboard.

[Paste] button

Pastes the text copied on the clipboard to the current cursor position in the [Edit] window or replaces the selected text with the copied text.

[Find] button

Finds the specified word in the active [Edit] window. A dialog box will appear allowing specification of the word to be found and a search condition.

[Find Next] button

Finds next target word towards the end of the file.

[Find Previous] button

Finds next target word towards the beginning of the file.

[Print] button

Prints the document in the active [Edit] window. A standard print dialog will appear allowing a specific print condition.

[Help] button

Displays the help window.

3.4.2 Build Toolbar

This tool bar has the following buttons and list boxes used to build a project:

[Assemble] button

Assembles the assembly source in the active [Edit] window. This button becomes active only when the active [Edit] window shows an assembly source file.

[Build] button

Builds the currently opened project using a general make process.

[Rebuild All] button

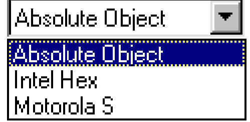
Builds the currently opened project. All the source files will be assembled regardless of whether they are updated or not.

[Stop Build] button

Stops the build process being executed. This button becomes active only while a build process is being executed.

[ICE Parameter] pull-down list box

Selects the ICE parameter file for the model being developed. In this box, all the ICE parameter files that exist in the "Dev63" directory are listed.



[Output Format] pull-down list box

Selects an executable object file format. Three types of formats are available: IEEE-695 absolute object format, Intel HEX format and Motorola S format. The build process will generate an executable object in the format selected here.

[HEX Convert] button

Invokes the HEX converter to convert an absolute object into an Intel HEX object or a Motorola S object. A dialog box will appear allowing selection of an absolute object and options of the HEX converter.

[Disassemble] button

Invokes the disassembler to disassemble an absolute object. A dialog box will appear allowing selection of an absolute object and options of the disassembler.

[Debug] button

Invokes the debugger with the specified ICE parameter file.

3.4.3 Window Toolbar

This tool bar has the following buttons used in window manipulation:

[Cascade] button

Cascades the opened [Edit] windows.

[Tile Horizontally] button

Tiles the opened [Edit] window horizontally.

[Tile Vertically] button

Tiles the opened [Edit] window vertically.

3.4.4 Toolbar Manipulation

Hiding and showing toolbars

Each toolbar can be hidden if not needed. Select the toolbar name from the [View] menu. This operation toggles between hiding and showing the toolbar.

Changing the toolbar location

Toolbars can be moved to another location in the toolbar area by dragging them. If a toolbar is moved out of the toolbar area, it will be changed to a window.

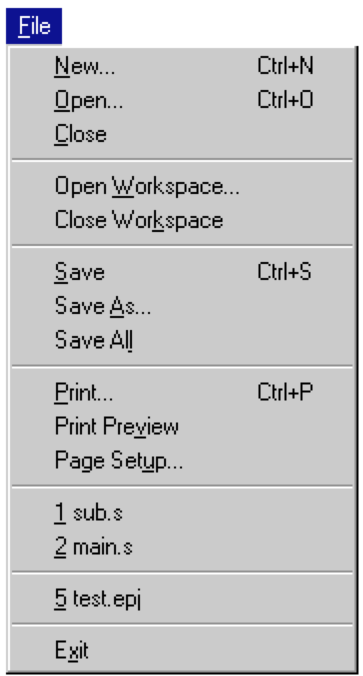
3.4.5 [Insert into project] Button on a [Edit] Window

[Insert into project] button

When a source file (.s or .ms) is opened, the [Insert into project] button appears on the [Edit] window. It can be used to insert the source file into the current opened project.
For other file types, the [Edit] window opens without the [Insert into project] button.

3.5 Menus

3.5.1 [File] Menu



The file names listed in this menu are recently used source and project files. Selecting one opens the file.
The number of files to be listed can be selected by the [Tools | Options] menu command.

[New...] ([Ctrl]+[N])

Creates a new document. A dialog box will appear allowing selection from among three document types: E0C63 assembly source, E0C63 assembly header and project.

[Open...] ([Ctrl]+[O])

Opens a document. A dialog box will appear allowing selection of the file to be opened.

[Close]

Closes the active [Edit] window. This menu item appears when an [Edit] window becomes active.

[Open Workspace...]

Opens a project. A dialog box will appear allowing selection of the project to be opened.

[Close Workspace]

Closes the currently opened project. This menu item becomes inactive if no project is opened.

[Save] ([Ctrl]+[S])

Saves the document in the active [Edit] window to the file. The file will be overwritten. This menu item appears when an [Edit] window becomes active.

[Save As...]

Saves the document in the active [Edit] window with another file name. A dialog box will appear allowing specification of a save location and a file name. This menu item appears when an [Edit] window becomes active.

[Save All]

Saves the documents of all [Edit] windows and the project information to the respective files.

[Print...] ([Ctrl]+[P])

Prints the document in the active [Edit] window. A standard [print] dialog box will appear allowing a specific print condition. This menu item appears when an [Edit] window becomes active.

[Print Preview]

Displays a print image of the document in the active [Edit] window. This menu item appears when an [Edit] window becomes active.

[Page Setup...]

Displays a dialog box for selecting paper and printer.

3.5.2 [Edit] Menu

[Undo] ([Ctrl]+[Z])

Undoes the previous executed operation in the [Edit] window.

[Cut] ([Ctrl]+[X])

Cuts the selected text in the [Edit] window to the clipboard.

[Copy] ([Ctrl]+[C])

Copies the selected text in the [Edit] window to the clipboard.

[Paste] ([Ctrl]+[V])

Pastes the text copied on the clipboard to the current cursor position in the [Edit] window or replaces the selected text with the copied text.

[Select All] ([Ctrl]+[A])

Selects all text in the active [Edit] window.

[Find...] ([Ctrl]+[F])

Finds the specified word in the active [Edit] window. A dialog box will appear allowing specification of the word to be found and a search condition.

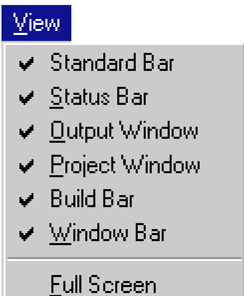
[Replace] ([Ctrl]+[H])

Replaces the specified words in the active [Edit] window with one another. A dialog box will appear allowing specification of the words.

[Go To] ([Ctrl]+[G])

Jumps to the specified line or label in the active [Edit] window. A dialog box will appear allowing specification of a line number or a label name.

3.5.3 [View] Menu



[Standard Bar]

Shows or hides the standard toolbar.

[Status Bar]

Shows or hides the status bar located at the bottom of the work bench window.

[Output Window]

Opens or closes the [Output] window.

[Project Window]

Opens or closes the [Project] window.

[Build Bar]

Shows or hides the build toolbar.

[Window Bar]

Shows or hides the window toolbar.

[Full Screen]

Maximizes the [Edit] window area to the full screen size.

3.5.4 [Insert] Menu

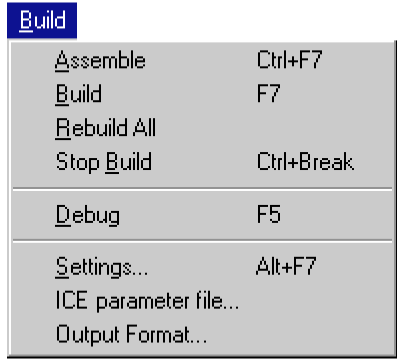
[File...]

Inserts the specified file to the current cursor position in the [Edit] window or replaces the selected text with the contents of the specified file. A dialog box will appear allowing selection of the file to be inserted.

[Files into project...]

Adds the specified source file in the currently opened project. A dialog box will appear allowing selection of the file to be added.

3.5.5 [Build] Menu



[Assemble] ([Ctrl]+[F7])

Assembles the assembly source in the active [Edit] window. This menu item becomes active only when the active [Edit] window shows an assembly source file.

[Build] ([F7])

Builds the currently opened project using a general make process.

[Rebuild All]

Builds the currently opened project. All the source files will be assembled regardless of whether they are updated or not.

[Stop Build] ([Ctrl]+[Break])

Stops the build process being executed. This button become active only while a build process is being executed.

[Debug] ([F5])

Invokes the debugger with the specified ICE parameter file.

[Settings...] ([Alt]+[F7])

Displays a dialog box for selecting tool options.

[ICE parameter file...]

Displays a dialog box for selecting an ICE parameter file.

[Output Format...]

Displays a dialog box for selecting an executable object file format. Three types of formats are available: IEEE-695 absolute object format, Intel HEX format and Motorola S format. The build process will generate an executable object in the format selected here.

3.5.6 [Tools] Menu

[HEX Converter...]

Invokes the HEX converter to convert an absolute object into an Intel HEX object or Motorola S object. A dialog box will appear allowing selection of an absolute object and options for the HEX converter.

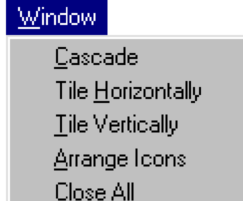
[Disassembler...]

Invokes the disassembler to disassemble an absolute object. A dialog box will appear allowing selection of an absolute object and options for the disassembler.

[Options...]

Displays a dialog box for selecting work bench options such as character colors in the [Edit] window and a printing font.

3.5.7 [Window] Menu



This menu appears when an [Edit] window is opened.

[Cascade]

Cascades the opened [Edit] windows.

[Tile Horizontally]

Tiles the opened [Edit] window horizontally.

[Tile Vertically]

Tiles the opened [Edit] window vertically.

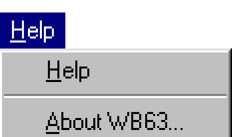
[Arrange Icons]

Arranges the minimized [Edit] window icons at the bottom of the [Edit] window area.

[Close All]

Closes all the [Edit] windows opened.

3.5.8 [Help] Menu



[Help]

Displays the [Help] window.

[About WB63...]

Displays a dialog box showing the version of the work bench.

3.6 Project and Work Space

The work bench manages a program development task using a work space folder and a project file that contains file and other information necessary for invoking the development tools.

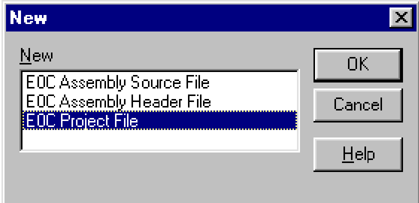
3.6.1 Creating a New Project

A new project file can be created by the following procedure:

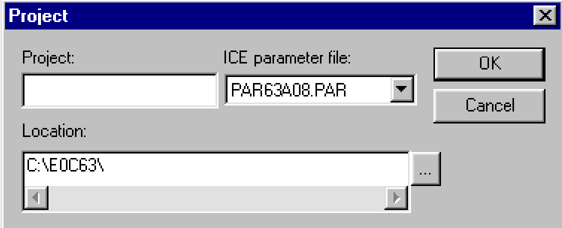
1. Select [New] from the [File] menu or click the [New] button.

   [New] button

   The [New] dialog box appears.



2. Select [E0C Project File] and click [OK].
   The [Project] dialog box appears.



3. Enter a project name, select an ICE parameter file and select a directory, then click [OK].

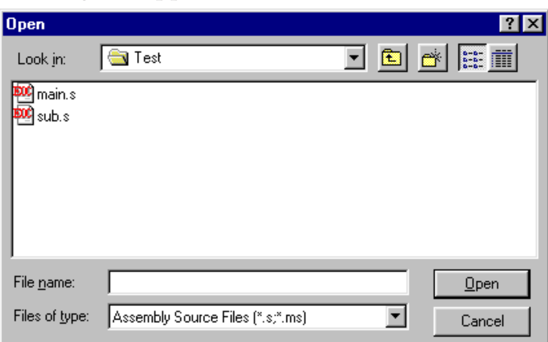
   ∗ The [ICE parameter file:] box lists the parameter files that exist in the "dev63" directory.

The work bench creates a folder (directory) with the specified project name as a work space, and puts the project file (.epj) into the folder.
If a folder which has the same name as that of a specified one already exists in the specified location, the work bench uses the folder as the work space. Thus you can specify a folder in which sources are created.
The specified project name will also be used for the absolute object and other files.

3.6.2 Inserting Sources into a Project

The sources created must be inserted into the project.
To insert a source into a project, use one of the four methods shown below:

1. [Insert | Files into project...] menu item
   A dialog box appears when this menu item is selected.

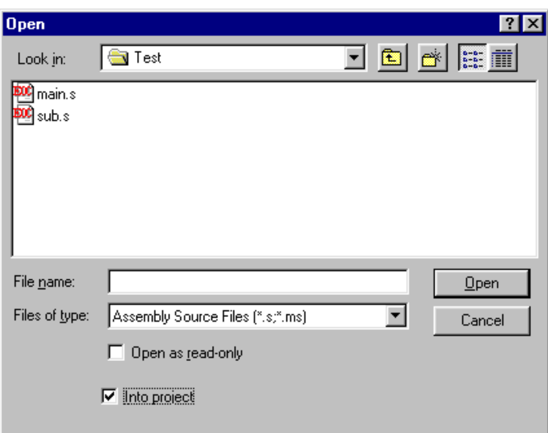


   Choose a source file from the list box and then click [Open].

2. [File | Open...] menu item or [Open] button

[Open] button

   A dialog box appears when this menu item or button is selected.



   Choose a source file from the list box and select the [Into project] button, then click [Open].

3. [Insert into project] button on the [Edit] window

[Insert into project] button

   When the source file has been opened, click the [Insert into project] button on the [Edit] window. Do not forget to save the source to the file before inserting into the project.

4. Dragging source files on the [Project] window
   Drag source files from Windows Explorer to the [Project] window. These files will be added to the current project.

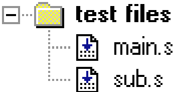
When a source file is inserted into the project, the source file name appears in the [Project] window.

Removing a source from the project

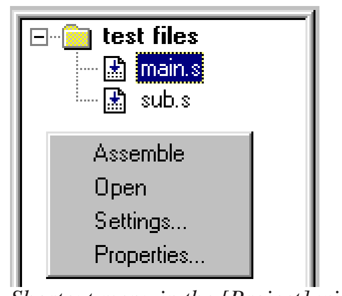
To remove a source file from the project, select the source in the [Project] window and then press the [Delete] key. This removes only the source information, and does not delete the actual source file.

3.6.3 [Project] Window

The [Project] window shows the work space folder and the source files included in the project that has been opened.

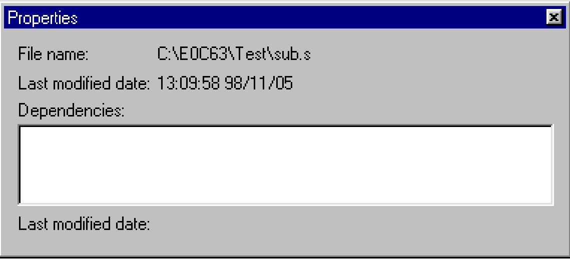


When a source file icon is double-clicked, the source file will be opened or the corresponding [Edit] window will be activated.



Shortcut menu in the [Project] window

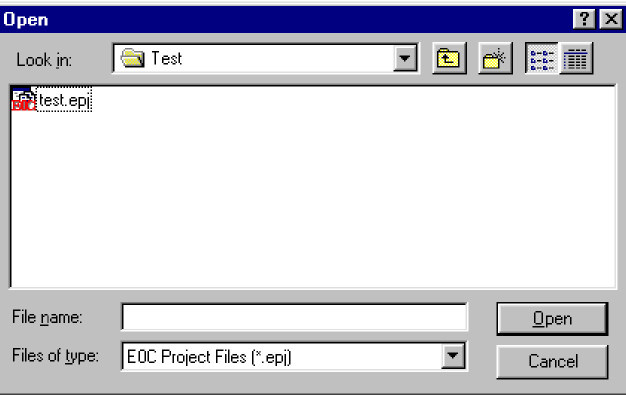
When the folder icon or a source file icon is clicked with the right mouse button, a shortcut menu including the available build menu items appears.
[Properties...] shows the source file information as follows:



Note: Note that the list in the [project] window is not the actual directory structure.
Sources of the project in other folders than the work space folder are also listed as they exist in the work space folder.

3.6.4 Opening and Closing a Project

To open a project, select [Open WorkSpace...] from the [File] menu.
A dialog box appears allowing selection of a project file.



The work bench allows only one project to be opened at a time. So if a project has been opened, it will be closed when another project is opened. At this time, a dialog box appears to select whether the current project file is to be saved or not if it has not already been saved after a modification.

The project file can also be opened by selecting [Open] from the [File] menu or clicking the [Open] button. In this case, choose the file type as E0C Project Files (*.epj) in the file open dialog box.

To close the currently opened project file, select [Close WorkSpace] from the [File] menu. At this time, a dialog box appears to select whether the current project file is to be saved or not if it has not already been saved after a modification. If [Yes] (save) is selected in this dialog box, all the modification items including sources, tool settings and window configuration will be saved.

3.6.5 Files in the Work Space Folder

The work bench generates the following files in the work space folder:

<file>.epj Project file

This file contains the project information.

<file>.cm Linker command file

This file is generated when a build task is started, and is used by the linker to generate an absolute object file.

Example:

```
; E0C WorkBench Generated
; Thursday, November 05, 1998

"C:\E0C63\dev63\DEV63A08\PAR63A08.PAR"         ;ICE parameter file

-o "test.abs"        ;output file : absolute object

; linked object file(s)
"sub.o"
"main.o"
```

The contents vary according to the source files included in the project and the linker option setting.

<file>.cmd Debugger startup command file

This file is generated when a build task is started, and is used by the debugger to execute the command in this file when it is started up.

Example:

```
lf "test.abs"
```

The work bench generates this file so that the executable file according to the format selection is loaded when the debugger starts up.

<file>.mak "make" file for build task

This file is generated when a build task is started, and is used for the build process in the work bench.

Example:

```
# E0C WorkBench Generated
# Thursday, November 05, 1998

ASM = as63.exe
LINK = lk63.exe
HEX = hx63.exe
ASM_FLG = -g
LINK_FLG = -g
HEX_FLG =

ALL : test.abs

test.abs : test.cm sub.o main.o
     $(LINK) $(LINK_FLG) test.cm

sub.o : C:\E0C63\Test\sub.s
     $(ASM) $(ASM_FLG) C:\E0C63\Test\sub.s

main.o : C:\E0C63\Test\main.s
     $(ASM) $(ASM_FLG) C:\E0C63\Test\main.s
```

This is a generic make file that contains macro setting and dependency list.

The following files are generated by the development tools during a build process:

<file>.o	Relocatable object files (generated by the assembler)
<file>.abs	Absolute object file (generated by the linker)
<file>.hsa, <file>.lsa, <file>.csa	Motorola S files (generated by the HEX converter when this format is specified in the work bench)
<file>h.hex, <file>l.hex, <file>c.hex	Intel HEX files (generated by the Hex converter when this format is specified in the work bench)

3.7 Source Editor

The work bench has a source editor function. Sources can be created and modified in the [Edit] window.

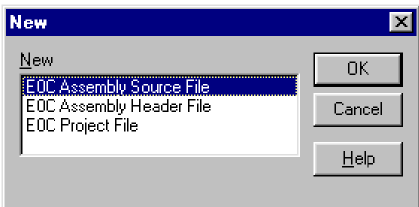
3.7.1 Creating a New Source or Header File

To create a new source file:

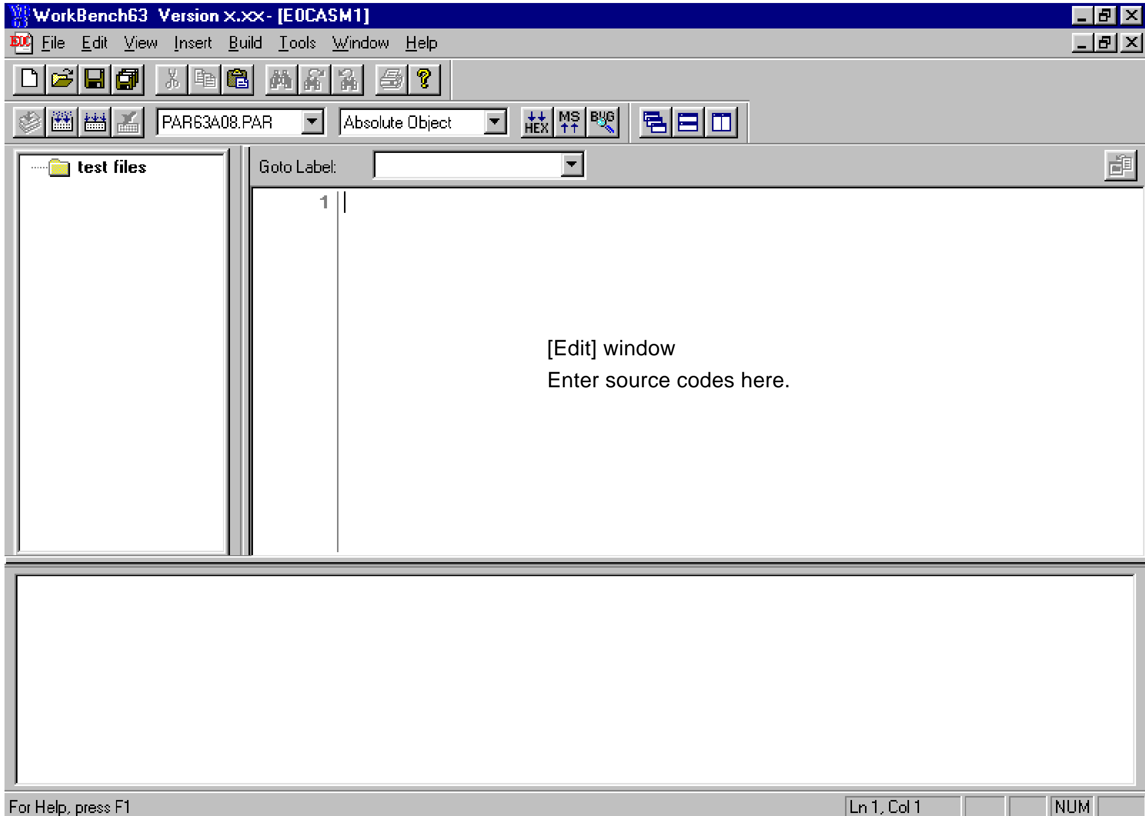
1. Select [New] from the [File] menu or click the [New] button.

   [New] button

   The [New] dialog box appears.



2. Select [E0C Assembly Source File] and click [OK].
   An [Edit] window appears.



Enter source codes in this window.

The [New] dialog box allows selection of the [E0C Header File]. Select it when creating a header file for constant definitions.

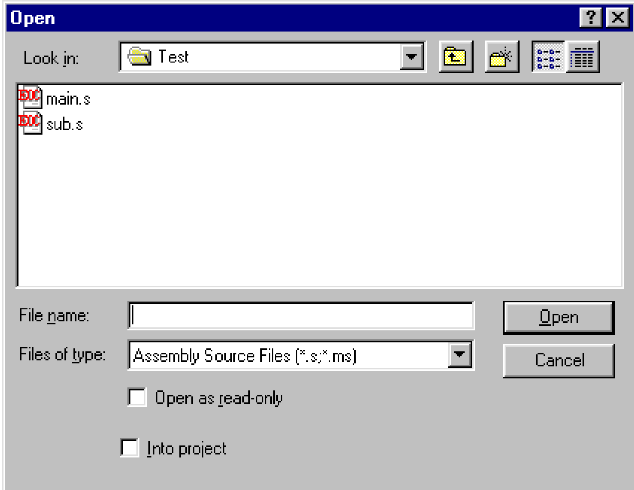
3.7.2 Loading and Saving Files

To load a source file:

1. Select [Open...] from the [File] menu or click the [Open] button.

   [Open] button

   The [Open] dialog box appears.

2. Choose a source file to be opened after selecting the file type (*.s, *.ms) and click [OK].
   An [Edit] window opens and shows the contents of the source file.

WorkBench63 Version x.xx- [main.s]

File Edit View Insert Build Tools Window Help

PAR63A08.PAR Absolute Object

test files

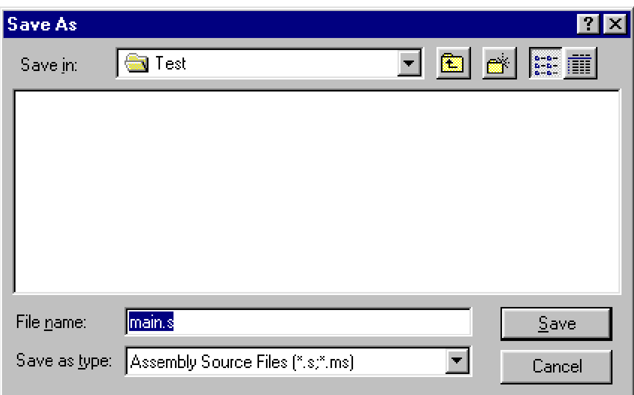
Goto Label:

```
1  ; main.s
2  ; ASM63 test program (main routine)
3  ;
4
5  ;***** INITIAL SP1 & SP2 ADDRESS DEFINITION *****
6
7  #ifdef SMALL_RAM
8          .set SP1_INIT_ADDR 0xb          ;SP1 init addr = 0x2c
9  #else
10         .set SP1_INIT_ADDR 0x4b         ;SP1 init addr = 0x12c
11 #endif
12
13         .set SP2_INIT_ADDR 0x1f         ;SP2 init addr = 0x1f
14
15
16 ;***** NMI & BOOT, LOOP *****
17
```

For Help, press F1 Ln 1, Col 1 NUM

To save the source:

1. Activate the [Edit] window of the source to be saved.
2. Select [Save as...] from the [File] menu.
   The [Save As] dialog box appears.



3. Enter the file name and then click [OK].

When overwriting the source on the existing file, select [Save] from the [File] menu or click the [Save] button.

[Save] button

To save all the source files opened and the project file, use the [File | Save All] menu item or the [Save All] button.

[Save All] button

3.7.3 Edit Function

The source editor has general text editing functions similar to standard Windows applications.

Editing text

Basic text editing function is the same as general Windows applications.
Cut, copy and paste are supported in the [Edit] menu and with the toolbar buttons. These commands are available only in the [Edit] window.
Undo can be selected from the [Edit] menu.

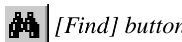
The tab stops are set at every 8 characters.

Find, replace and go to

Any words can be searched in the active [Edit] window.

Find

To find a word, select [Find...] from the [Edit] menu or click the [Find] button.

 *[Find] button*

The [Find] dialog box appears.

The controls in the dialog are as follows:

[Find what:] text box

Enter the word to be found in this text box. The specified word is maintained as the finding word even if this dialog box is closed.

[Match whole word only] check box

If this option is selected, the work bench searches only the words that are completely matched with the specified word. If not, only the part of word that matches the specified word will be searched.

[Match case] check box

If this option is specified, a case-sensitive search is performed. If not, a case-insensitive search is performed.

[Direction] option

If the [Up] radio button is selected, the specified word is searched toward to the beginning of the file. If the [Down] radio button is selected, a search is performed toward to the end of the file.

[Find Next] button

Clicking this button starts searching the specified word. If the specified word is found, the [Edit] window refreshes the display and highlights the word found.

[Cancel] button

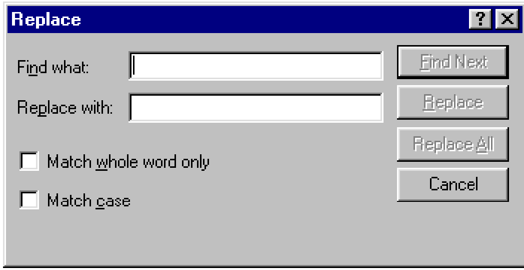
Clicking this button closes the dialog box.

Once a word to be found is specified in the [Find] dialog box, the [Find Next] and [Find Previous] buttons on the toolbar can be used for a forward or backward search.

[Find Next] button *[Find Previous] button*

Replace

To replace a word with another one, select [Replace] from the [Edit] menu.
The [Replace] dialog box appears.



The controls in the dialog are as follows:

[Find what:] text box

Enter the word to be found in this text box. If a word has been specified in the [Find] dialog box, it appears in this box.

[Replace with:] text box

Enter the substitute word in this box.

[Match whole word only] check box

If this option is selected, the work bench searches only the words that are completely matched with the specified word. If not, only the part of word that matches the specified word will be searched.

[Match case] check box

If this option is specified, a case-sensitive search is performed. If not, a case-insensitive search is performed.

[Find Next] button

Clicking this button starts searching the specified word. If the specified word is found, the [Edit] window refreshes the display and highlights the word found.

[Replace] button

By clicking this button after the specified word is found, it is replaced with the substitute word. Then the work bench searches the next.

[Replace All] button

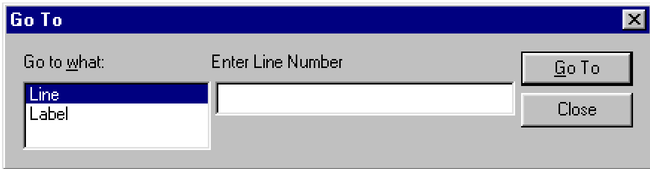
Replaces all the specified found words with the substitute word. Note that undo function cannot be performed for this operation except for the last replaced word.

[Cancel] button

Clicking this button closes the dialog box.

Go to

You can go to any source line or any label position quickly.
To do this, select [Go To] from the [Edit] menu.
The [Go To] dialog box appears.

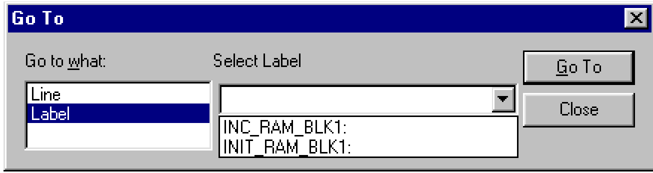


Going to a source line

1. Select "Line" in the [Go to what:] list box.
2. Type a line number in the [Enter Line Number] box and then click the [Go To] button.

Going to a label position

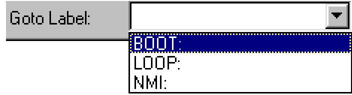
1. Select "Label" in the [Go to what:] list box.
   The [Enter Line Number] box changes to the [Select Label] list box.



2. Select a label from the [Select Label] box and then click the [Go To] button.

The [Select Label] list box has a pull-down menu that contains the list of labels defined in the current source file.

The [Edit] windows for source files (*.s, *.ms) have the [Go To Label] list box similar to the [Select Label] list box in the [Go To] dialog box. You can also go to a label position using this box.

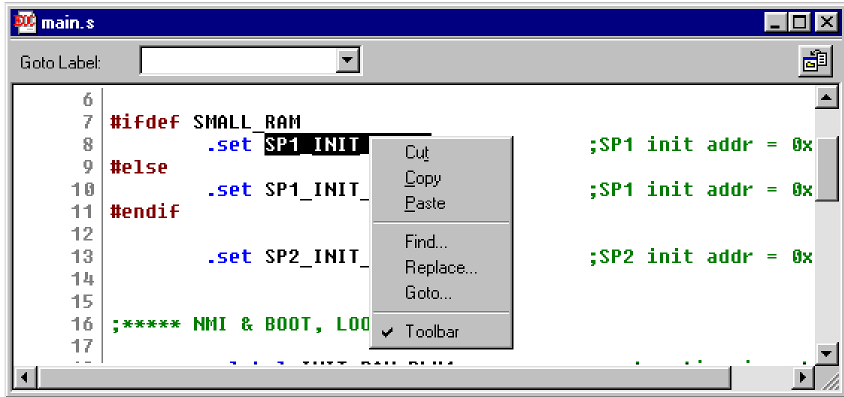


Inserting a file

To insert a file such as a header file and another source at the cursor position of the current source, select [File...] from the [Insert] menu.
A dialog box will appears allowing selection of the file to be inserted.

Shortcut menu

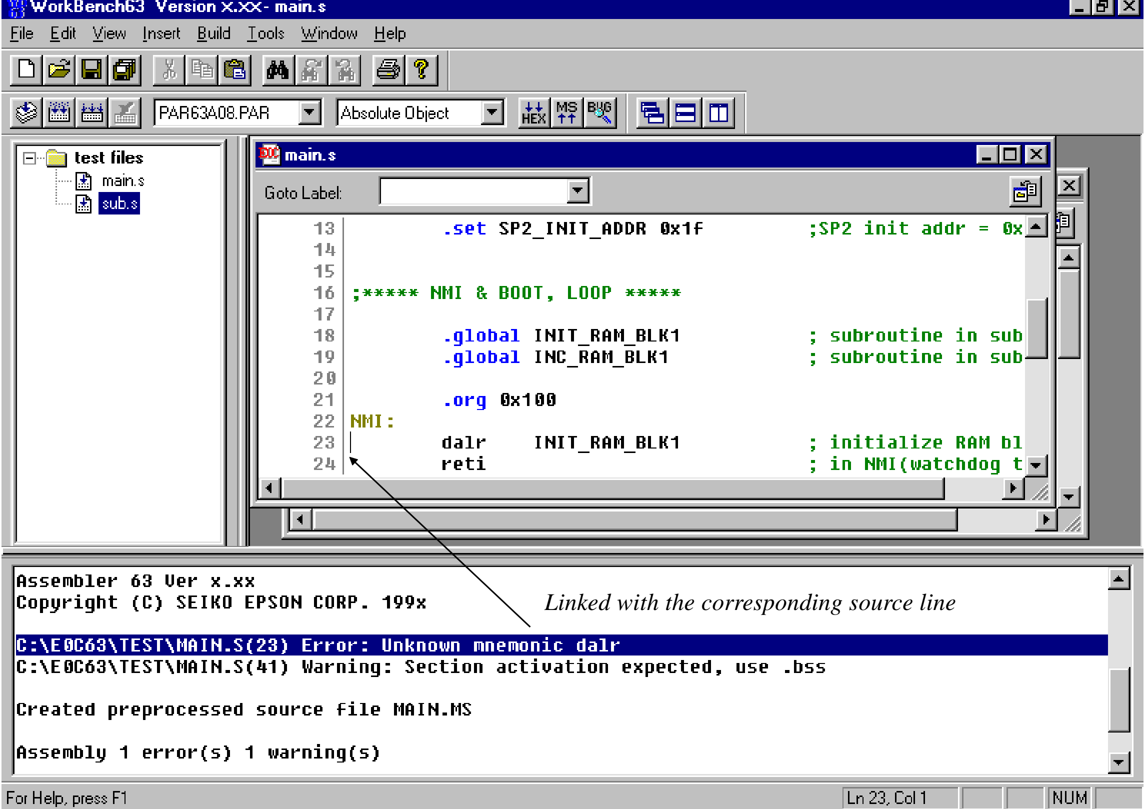
The [Edit] window supports a short cut menu that appears by clicking the right mouse button on the [Edit] window. It can also be done by pressing the [Short cut menu] key while the [Edit] window is active if the key is available on the keyboard. It contains the editing menu items descried above, so you can select an edit command using this menu.



3.7.4 Tag Jump Function

When assembler syntax errors occur during assembling, their error messages are displayed in the [Output] window. In this case, you can go to the source line in which an error has occurred by double-clicking the error message in the [Output] window.

However, this function is available only when the error message contains a source line number.



3.7.5 Printing

The document in the [Edit] window can be printed out.
The [Print...], [Print Preview] and [Page Setup...] commands are provided in the [File] menu. The [Print] button can also be used. They have the same function as those of standard Windows application.
Select one after activating the [Edit] window of the document to be printed.

3.8 Build Task

By using the [Build] menu or [Build] toolbar, the assembler, linker, debugger, HEX converter and disassembler can be executed from the work bench.
In the work bench, process to generate an executable object from the source files is called a build task.

For details of each development tool, refer to the respective chapter.

3.8.1 Preparing a Build Task

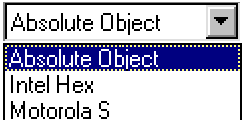
Before starting a build task, necessary source files should be prepared and tool options should be configured.
1. Create a new project. (Refer to Section 3.6.1.)
2. Select an ICE parameter file. (Refer to Section 3.6.1.)
3. Create source files and add them into the project. (Refer to Sections 3.7 and 3.6.2.)
4. Select tool options (Refer to Section 3.9.)

3.8.2 Building an Executable Object

To generate an executable object:

1. Open the project file.

2. Select an output format (absolute, Intel HEX or Motorola S) using the [Output Format] list box.



3. Select [Build] from the [Build] menu or click the [Build] button.

[Build] button

The work bench generates a make file according to the source files in the project and the tool options set by the user. This file is used to control invocation of tools.
First, the make process invokes the assembler for each source file to be assembled. If the latest relocatable object file exists in the work space, the corresponding source file is not assembled to reduce process time.
Next, the linker is invoked to generate an absolute object file. The linker command file used in this phase is automatically generated.
If absolute object has been selected as the output format, the build task is completed at this phase. If Intel HEX or Motorola S has been selected, the HEX converter will be invoked to generate an object in the specified format.

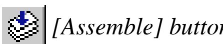
To rebuild all files including the latest relocatable object files, select [Rebuild All] from the [Build] menu or click the [Rebuild All] button.

[Rebuild All] button

The build task can be suspended by selecting [Stop Build] from the [Build] menu or clicking the [Stop Build] button.

[Stop Build] button

To invoke only the assembler, select [Assemble] from the [Build] menu or click the [Assemble] button after activating the [Edit] window of the source to be assembled.


[Assemble] button

3.8.3 Debugging

To debug the generated executable file, select [Debug] from the [Build] menu or click the [Debug] button.

[Debug] button

The debugger starts up with the specified ICE parameter file and then loads the executable object by the command file generated from the work bench.
This command file contains the command to load the specified type of an executable object to the debugger. The contents of the command file can be edited in the [Settings] dialog box explained in Section 3.9.

* When the building process is performed again after invoking the debugger, the debugger will reload the object file if its window can be activated.

```
Db63 - Command
File  Run  Break  Trace  View  Option  Window  Help

Mix
Search Label:
    Addr  Code   Unassemble                 Source
 57 0110  094b   ldb  ba,4b            ldb     %ba,SP1_INIT_ADDR
 58 0111  1fc4   ldb  sp1,ba           ldb     %sp1,%ba
 59 0112  091f   ldb  ba,1f            ldb     %ba,SP2_INIT_ADDR
 60 0113  1fc6   ldb  sp2,ba           ldb     %sp2,%ba
 61 0114  08fe   ldb  ext,fe     (+)   ldb  ext,fe
 62 0115  02ea   calr ea               calr     INIT_RAM_BLK1
 63                               LOOP:
 64 0116  08fe   ldb  ext,fe     (+)   ldb  ext,fe
 65 0117  02ef   calr ef               calr     INC_RAM_BLK1
 66 0118  00fd   jr    fd              jr       LOOP
 67

Reg
PC    :0110
A     :A
B     :A
X     :[AAAA] = *
Y     :[AAAA] = *
EICZ  :0000
SP1   :AA
SP2   :AA
EXT   :AA
QUEUE:AAAA
0000000000 cycle
[0000] = AAAA
[0004] = AAAA
[0008] = AAAA

Command
LCD board version          : 1.0
EXT board version          : No use ..... warning : check
ICE hardware version       : 1.0
ICE software version       : 1.0
DIAG test                  : omitted
Map ......................... done
Initialize .................. done
>com "C:\E0C63\Test\test.cmd" 0
>lf "test.abs"
Loading file...OK!
>

Data
ADDR: 0 1 2 3 4 5 6 7  8 9 A B C D E F
0000  A A A A A A A A  A A A A A A A A
0010  A A A A A A A A  A A A A A A A A
0020  A A A A A A A A  A A A A A A A A
0030  A A A A A A A A  A A A A A A A A
0040  A A A A A A A A  A A A A A A A A
0050  A A A A A A A A  A A A A A A A A
0060  A A A A A A A A  A A A A A A A A
0070  A A A A A A A A  A A A A A A A A
0080  A A A A A A A A  A A A A A A A A
0090  A A A A A A A A  A A A A A A A A

Ready                                      NUM
```

Refer to Chapter 8, "Debugger", for operating the debugger.

3.8.4 Executing Other Tools

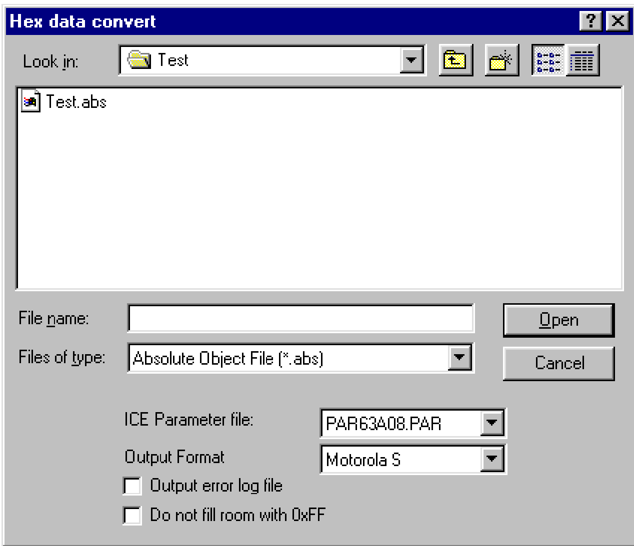
The HEX converter and disassembler can be invoked independently.

HEX converter

To invoke the HEX converter, select [HEX converter...] from the [Tools] menu or click the [HEX convert] button.

[HEX convert] button

Then select an absolute object file to be converted in the [Hex data convert] dialog box.



This dialog box allows selection of the HEX converter options.

[ICE Parameter file:] list box

Select an ICE parameter file from the pull-down list.

[Output Format:] list box

Select an output format from between Intel HEX and Motorola S.

[Output error log file] check box

Select this option to generate the error log file of the HEX converter.

[Do not fill room with 0xFF] check box

Select this option when not filling the unused program area with 0xFF.

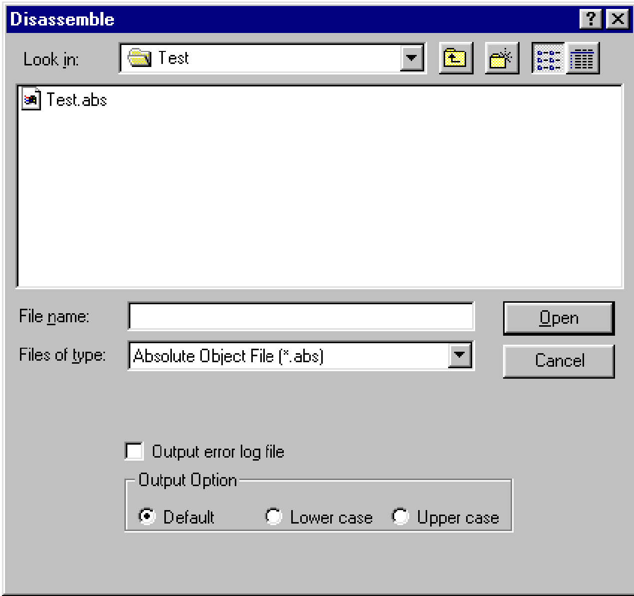
After selecting an absolute object and options, click the [Open] button. The HEX converter starts up and converts the selected object into the specified format. The messages delivered from the HEX converter are displayed in the [Output] window.

Disassembler

To invoke the disassembler, select [Disassembler...] from the [Tools] menu or click the [Disassemble] button.

[Disassemble] button

Then select the executable object file to be disassembled in the [Disassemble] dialog box.



This dialog box allows selection of the disassembler options.

[Output error log file] check box

Select this option to generate the error log file of the disassembler.

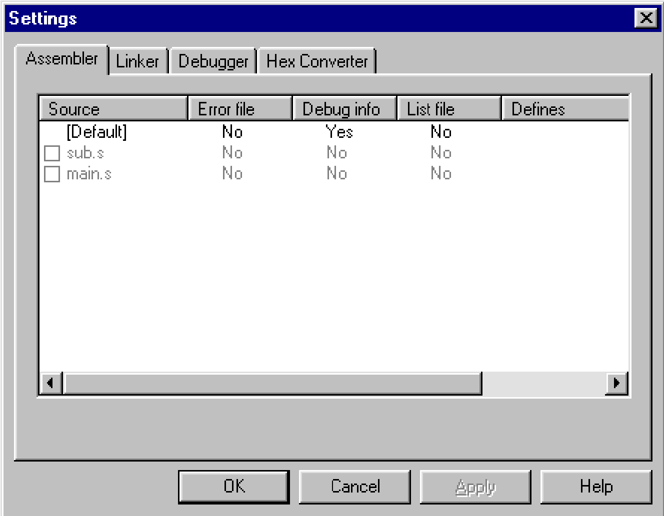
[Output Option]

Select a character case option using the radio buttons.
When [Default] is selected, the disassembled source will be made with all labels in upper-case characters and instructions in lower-case characters.
When [Upper case] is selected, the source will be made with upper-case characters only.
When [Lower case] is selected, the source will be made with lower-case characters only.

After selecting an executable object and options, click the [Open] button. The disassembler starts up and converts the selected object into the source file. The messages delivered from the disassembler are displayed in the [Output] window.

3.9 Tool Option Settings

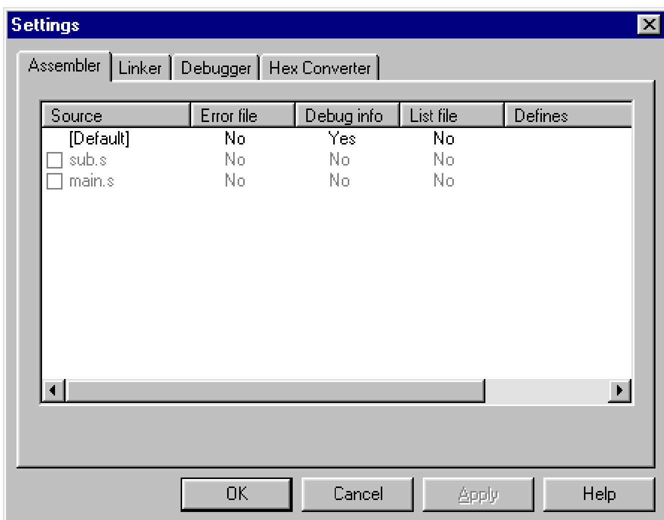
The development tools have startup options that can be specified when invoking them.
These settings can be made in the [Settings] dialog box that appears by selecting [Settings...] from the [Build] menu.



Click the tool name tab to view option settings of each tool.

Clicking the [OK] button updates option setting information in the project and then closes the dialog box. To continue to select other tool options, click the [Apply] button. This does not close the dialog box. Clicking the [Cancel] button closes the dialog box.

3.9.1 Assembler Options



In this dialog, the following four assembler options can be selected.
[Error file] Output of an error file (No: Not output, Yes: Output)
[Debug info] Addition of debugging information to the relocatable object (No: Not added, Yes: Added)
[List file] Output of the relocatable list file (No: Not output, Yes: Output)
[Defines] Name definition for conditional assembly (Enter a define name.)

The edit box shows the default setting ([Default]) and the list of source files in the project.
The default setting applies to all the sources excluding ones that are specified independently.
To select options of a specific source, select the check box at the front of the source file name.

Check here → ☐ sub.s No No No

Each of the [Error file], [Debug info] and [List file] options is set to either "No" or "Yes" and it toggles by double-clicking. For example, to change the default [List file] option from "No" to "Yes", double click "No" in the [Default] line. It changes to "Yes".

Source	Error file	Debug info	List file	Defines
[Default]	No	Yes	No ← *Double-click here. It will be changed to Yes.*	

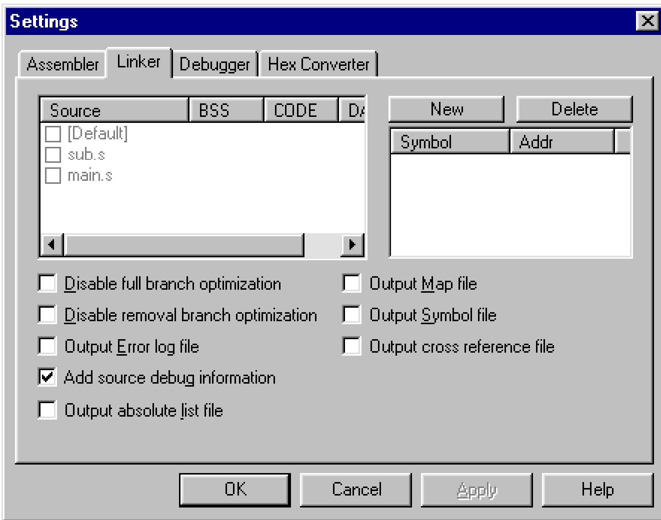
To define a name for conditional assembly, double-clicking the [Defines] part.

Source	Error file	Debug info	List file	Defines
[Default]	No	Yes	No	← *Double-click here, then type a define name.*

An text box appears. Type a name in the box. If two or more names are to be entered, separate each name with a comma (,).

Refer to Chapter 4, "Assembler", for details of the assembler options.

3.9.2 Linker Options



In this dialog, section allocation, symbol definition and other linker options can be specified.
The work bench generates a linker command file including these specifications, and specifies it when invoking the linker.

Specifying section allocation

This option is set by default as all the sections will be allocated from the memory start address. To specify a section start address, double click the cell and then enter the address.

Source	BSS	CODE	DATA
☐ [Default]		← *Double-click here to change default CODE section start address, then type an address.*	

Source	BSS	CODE	DATA
☑ [Default]		0x100	

The edit box shows the default setting ([Default]) and the list of source files in the project.
The default setting applies to all the sections excluding those of the source specified.
To set a specific source independently, select the check box at the front of the source file name.

Check here → ☑ sub.s 0x200

Symbol definition

To define a symbol, click the [New] button and then enter the symbol name and address in the edit box.

```
Symbol          Addr
[          ]    [          ]   ← Enter a symbol name and the address.
```

To modify a symbol name or address, double click the name or the address in the edit box and then enter a new name or address.

```
Symbol          Addr
TEST            0x0000   ← Double-click to modify.
```

To delete a symbol, highlight the symbol line by clicking and then click the [Delete] button.

Other option selections

[Disable full branch optimization] check box
Select this option if extension code insertions, deletions and corrections are not necessary.

[Disable removal branch optimization] check box
Select this option if extension code deletions are not necessary.

[Output Error log file] check box
Select this option to generate the error log file of the linker.

[Add source debug information] check box
Select this option to add the debugging information. If this option is not specified, the sources cannot be displayed in debugging.

[Output absolute list file] check box
Select this option to generate the absolute list file.

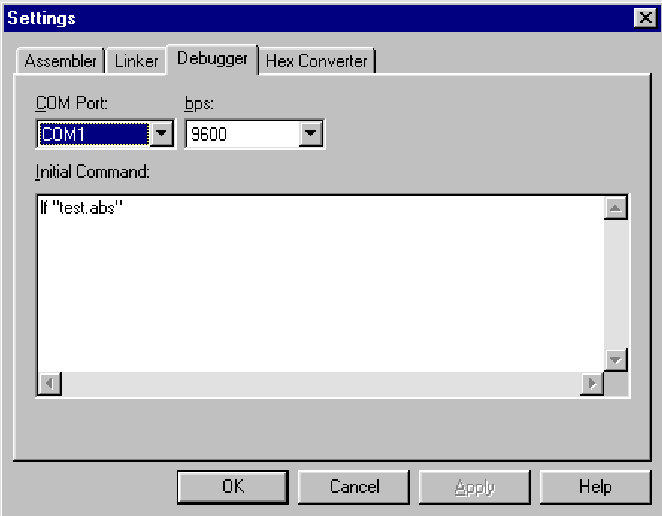
[Output Map file] check box
Select this option to generate the link map file.

[Output Symbol file] check box
Select this option to generate the symbol file.

[Output cross reference file] check box
Select this option to generate the cross reference file.

Refer to Chapter 5, "Linker", for details of the linker options.

3.9.3 Debugger Options



[COM Port:] list box

Select a COM port of the personal computer used to communicate with the ICE. COM1 is set by default.

[bps:] list box

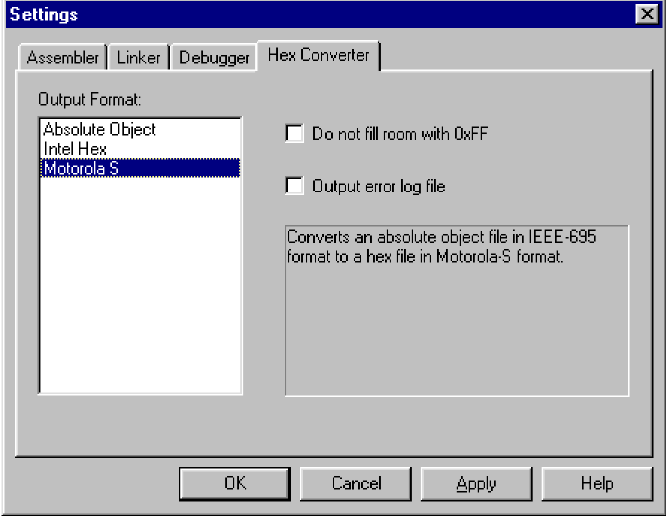
Select a baud rate to communicate with the ICE. 9600 bps is set by default.

[Initial Command:] edit box

This box is used to edit the debugger commands to be executed when the debugger starts up. The work bench generates a command file with the commands entered in this box and specifies it when invoking the debugger. A load command is initially set so that the debugger can load the object at start up.

Refer to Chapter 8, "Debugger", for details of the debugger options.

3.9.4 HEX Converter Options



[Output Format:] list box

An output format of the executable object to be generated by the build task can be selected.
When "Absolute Object" is selected, the build task will be terminated after linking has completed. The HEX converter will not be invoked.
When "Intel Hex" or "Motorola S" is selected, the HEX converter will be invoked after linking has completed.
Other HEX converter options become selectable when one of them is selected.

[Do not fill room with 0xFF] check box

Select this option when not filling the unused program area with 0xFF.

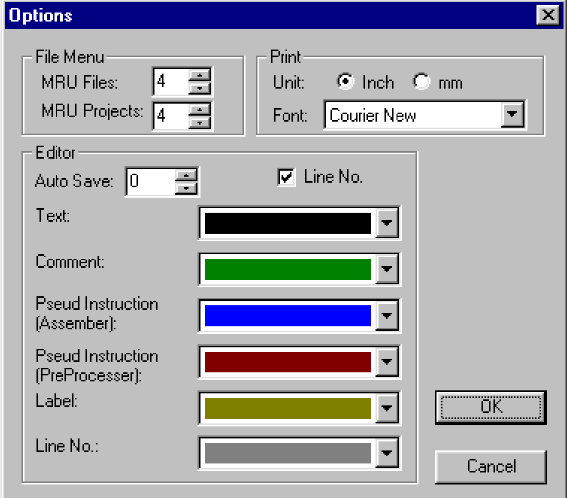
[Output error log file] check box

Select this option to generate the error log file of the HEX converter.

Refer to Chapter 6, "HEX Converter", for details of the HEX converter options.

3.10 Work Bench Options

[Options...] in the [Tools] menu allows selection of some options for customizing the work bench. When this menu item is selected, a dialog box appears.



File menu options

[MRU Files:] box

This option allows selection of a number of recently used files to be listed in the [File] menu. The selectable range is 0 to 9.

[MRU Projects:] box

This option allows selection of a number of recently used project files to be listed in the [File] menu. The selectable range is 0 to 9.

Print options

[Unit:] radio button

This option allows selection of a unit used for specifying the margins of the printing sheet. Either "inch" or "mm" can be selected. This selection affects the margin setup field in the [Page Setup...] dialog box.

[Font:] list box

This option allows selection of a font used for printing the document in the [Edit] window.

Editor options

[Auto Save:] box

This option sets an auto-save interval for the document to be edited in the [Edit] window. The selectable range is 0 to 999 minutes. When 0 is selected, the document being edited will not be automatically saved.

[Line No.] check box

This option enables or disables the line number display in the [Edit] window.

Color selection list box

These list boxes allow selection of colors used to display the document in the [Edit] window. Text (mnemonics), comments, assembler pseudo-instructions, preprocessor pseudo-instructions, labels and line numbers are displayed with different colors selected here.

Note: The contents selected in this dialog box will be effective after restarting the work bench.

3.11 Short-Cut Key List

Key operation	Function
Ctrl + N	Creates a new document
Ctrl + O	Opens an existing document
Ctrl + F12	Opens an existing document
Ctrl + S	Saves the document
Ctrl + P	Print the active document
Ctrl + Shift + F12	Print the active document
Ctrl + Z	Undoes the last action
Alt + BackSpace	Undoes the last action
Ctrl + X	Cuts the selection and puts it on the clipboard
Shift + Delete	Cuts the selection and puts it on the clipboard
Ctrl + C	Copies the selection to the clipboard
Ctrl + Insert	Copies the selection to the clipboard
Ctrl + V	Inserts the clipboard contents at the insertion point
Shift + Insert	Inserts the clipboard contents at the insertion point
Ctrl + A	Selects the entire document
Ctrl + F	Finds the specified text
F3	Finds next
Shift + F3	Finds previous
Ctrl + H	Replaces the specified text with different text
Ctrl + G	Moves to the specified location
Ctrl + F7	Assembles the file
F7	Builds the project
Ctrl + Break	Stops the build
F5	Debugs the project
Alt + F7	Edits the project build and debug settings
Ctrl + Tab	Next MDI Window
Short-cut-key	Opens the popup menu
Shift + F10	Opens the popup menu

3.12 Error Messages

The work bench error messages are given below.

Error message	Description
<filename> is changed by another editor. Reopen this file?	The currently opened file is modified by another editor.
Cannot create file: <filename>	The file (linker command file, debugger command file, etc.) cannot be created.
Cannot find file: <filename>	The source file cannot be found.
Cannot find ICE parameter file	The ICE parameter file cannot be found.
Cannot open file: <filename>	The source file cannot be opened.
You cannot close workspace while a build is in progress. Select the Stop Build command before closing.	The project close command or work bench terminate command is specified while the build task is being processed.
Would you like to build it?	The debugger invoke command is specified when the build task has not already been completed.

3.13 Precautions

(1) The source file that can be displayed and edited in the work bench is limited to 16M byte size.

(2) The label search and coloring function of the work bench does not support labels that have not ended with a colon.

(3) The work bench can create a make, linker command and debugger command files, note, however, that these files or settings created with another editor cannot be input into the work bench.

CHAPTER 4 ASSEMBLER

This chapter describes the functions of the assembler as63 and grammar involved with the creation of assembly source files.

4.1 Functions

The assembler as63 is a tool that constitutes the core of this software package. It assembles (translates) assembly source files and creates object files in the machine language.
The functions and features of the assembler are summarized below:
- Allows absolute and relocatable sections mixed in one source.
- Allows to develop programs in multiple sources by creating relocatable object files that can be combined by the linker.
- Can add source debugging information for source debugging on the debugger.
- Upper compatible with the old E0C63 preprocessor and assembler.

The assembler provides the following additional functions as well as the basic assembly functions:
- Macro definition and macro invocation
- Definition of Define name
- Operators
- Insertion of other file
- Conditional assembly

The assembler processes source files in two stages: preprocessing stage and assembling stage. The preprocessing stage expands the additional function part described in the source file to mnemonics that can be assembled, and delivers them to a temporary file (preprocessed file). The assembling stage assemble the preprocessed file to convert the source codes into the machine codes.

4.2 Input/Output Files

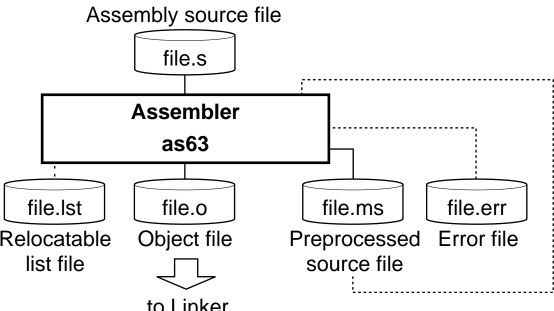


Fig. 4.2.1 Flow chart

4.2.1 Input File

Assembly source file

File format: Text file
File name: <File name>.s
<File name>.ms (A preprocessed source file created by the assembler or disassembler.)
Description: File in which a source program is described. If the file extension is omitted, the assembler finds a source file that has the specified file name and an extension ".s".
Note: When a ".s" source file is specified, it will be processed in the preprocessing stage and then the assembling stage. When a ".ms" source file is specified, it will be processed only in the assembling stage. Therefore, ".ms" files cannot include preprocessor instructions.

4.2.2 Output Files

Object file

File format: Binary file in relocatable IEEE-695 format
File name: <file name>.o (The <file name> is the same as that of the input file, unless otherwise specified with the -o option.)
Output destination: Current directory
Description: File in which machine language codes are stored in a relocatable form available for the linker to link with other modules and to generate an executable absolute object.

Relocatable list file

File format: Text file
File name: <file name>.lst (The <file name> is the same as that of the input file, unless otherwise specified with the -o option.)
Output destination: Current directory
Description: File in which offset locations, machine language codes and source codes are stored in plain text.

Preprocessed file

File format: Text file
File name: <file name>.ms (The <file name> is the same as that of the input file, unless otherwise specified with the -o option.)
Output destination: Current directory
Description: File in which instructions for preprocessing (e.g. conditional assembly and macro instructions) are expanded into an assembling format.

Error file

File format: Text file
File name: <file name>.err (The <file name> is the same as that of the input file, unless otherwise specified with the -o option.)
Output destination: Current directory
Description: The file is created if the -e option is specified. It records error messages and other information which the assembler delivers via the Standard Output (stdout).

4.3 Starting Method

General form of command line

as63 ∧ [options] ∧ [<source file name>]

∧ denotes a space.
[ ] indicates the possibility to omit.

Source file name

In the command line, only one assembly source file can be specified at a time. Therefore, you will have to process multiple files by executing the assembler the number of times equal to the number of files to be processed.
A long file name supported in Windows and a path name can be specified. When including spaces in the file name, enclose the file name with double quotation marks (").

Options

The assembler comes provided with the following six start-up options:

-d <define name>

Function: Definition of Define name
Explanation:
- Works in the same manner as you describe "#define <define name>" at top of the source. It is an option to control the conditional assembly at the start-up.
- One or more spaces are necessary between -d and the <define name>.
- To define two or more Define names, repeat the specification of "-d <define name>".

-g

Function: Addition of debugging information
Explanation:
- Creates an output file containing symbolic/source debugging information.
- Always specify this function when you perform symbolic/source debugging.

Default: If this option is not specified, no debugging information will be added to the relocatable object file.

-o <file name>

Function: Specification of output path/file name
Explanation:
- Specifies an output path/file name without extension or with an extension ".o". If no extension is specified, ".o" will be supplemented at the end of the specified output path/file name.

Default: The input file name is used for the output file names.

-c

Function: Ignore character case of symbols
Explanation:
- Allows description of symbols in case insensitive.

Default: If this option is not specified, symbol names will be case sensitive.

-l

Function: Output of relocatable list file
Explanation:
- Outputs a relocatable list file.

Default: If this option is not specified, no relocatable list file will be output.

-e

Function: Output of error file
Explanation:
- Creates an .err file which contains the information that the assembler outputs to the Standard Output (stdout), such as error messages.

Default: If this option is not specified, no error file will be created.

When entering an option in the command line, you need to place one or more spaces before and after the option. The options can be specified in any order. It is also possible to enter options after the source file name.

Example: `c:\e0c63\bin\as63 -g -e -l -d TEST1 -d TEST2 test.s`

4.4 Messages

The assembler delivers all its messages through the Standard Output (stdout).

Start-up message

The assembler outputs only the following message when it starts up.

```
Assembler 63 Ver x.xx
Copyright (C) SEIKO EPSON CORP. 199x
```

End message

The assembler outputs the following messages to indicate which files have been created when it ends normally.

```
Created preprocessed source file <FILENAME.MS>
Created relocatable object file <FILENAME.O>
Created relocatable list file <FILENAME.LST>
Created error log file <FILENAME.ERR>

Assembly 0 error(s) 0 warning(s)
```

Usage output

If no file name was specified or the option was not specified correctly, the assembler ends after delivering the following message concerning the usage:

```
Usage: as63 [options] <file name>
Options: -d <symbol>    Add preprocess definition
         -e             Output error log file (.ERR)
         -g             Add source debug information in object
         -l             Output relocatable list file (.LST)
         -c             Ignore character case of symbols
         -o <file name> Specify output file name
File name: Source file name (.S or .MS)
```

When error/warning occurs

If an error is produced, an error message will appear before the end message shows up.

Example:

```
TEST.S(5) Error: Illegal syntax
Assembly 1 error(s) 0 warning(s)
```

In the case of an error, the assembler ends without creating an output file. If an error occurs at the preprocessing stage in the assembler, the assembler stops processing and outputs preprocess-level errors only.

If a warning is issued, a warning message will appear before the end message shows up.

Example:

```
TEST.S(6) Warning: Expression out of range
Assembly 0 error(s) 1 warning(s)
```

In the case of a warning, the assembler ends after creating an output file.

The source file name that was specified in the command line will appear at the beginning of the error and warning messages.

For details on errors and warnings, refer to Section 4.10, "Error/Warning Messages".

4.5 Grammar of Assembly Source

Assembly source files should be created on a general-purpose editor or the source editor of the work bench. Save sources as standard text files. For the file name, a long file name supported in Windows can be specified.

This section explains the rules and grammar involved with the creation of assembly source files.

4.5.1 Statements

Each individual instruction or definition of an assembly source is called a statement. The basic composition of a statement is as follows:

Syntax pattern

(1) Mnemonic	Operand	(;comment)
(2) Assembler pseudo-instruction	Parameter	(;comment)
(3) Label:		(;comment)
(4) ;comment		

```
Example:   <Statement>                                   <Syntax Pattern>
#include   "define.h"                                        (2)
           .set  IO1, 0xfff1                                 (2)

; TEXT SECTION (ROM, 13bit width)                            (4)

           .org  0x100                                       (2)
NMI:                                                         (3)
           reti                                              (1)
           nop                                               (1)
           nop                                               (1)
           jr    NMI                                         (1)

           .org  0x110                                       (2)
BOOT:                                                        (3)
           ld    %f,0x4                                      (1)
           ld    %a,0                                        (1)
           ld    %a,0                                        (1)
           ldb   %ext,0   ; clear memory 0 to 3              (1)
               :              :
```

The example given above is an ordinary source description method. For increased visibility, the elements composing each statement are aligned with tabs and spaces.

Restrictions

- Only one statement can be described in one line. A description containing more than two instructions in one line will result in an error. However, a comment or a label may be described in the same line with an instruction.

  Example:

  ```
  ;OK
  BOOT:      ld    %f,0x4

  ;Error
  BOOT:      ld    %f,0x4          ld    %a,0x0
  ```

- One statement cannot be described in more than one line. A statement that cannot complete in one line will result in an error.

  Example:

  ```
  .word      0x0,0x1,0x2,0x3 ... OK
  .word      0xa,0xb,0xc,0xd ... OK

  .word      0x0,0x1,0x2,0x3
             0xa,0xb,0xc,0xd ... Error
  ```

- The maximum describable number of characters in one line is 259 (ASCII characters). If this number is exceeded, an error will result.

- The usable characters are limited to ASCII characters (alphanumeric symbols), except for use in comments. Also, the usable symbols have certain limitations (details below).

- The reserved words such as mnemonics and pseudo-instructions are all not case sensitive, while the user defined items such as labels and symbols are all case sensitive if the -c option is not specified. Therefore, mnemonics and pseudo-instructions can be written in uppercase (A–Z) characters, lowercase (a–z) characters, or both. For example, "ld", "LD", and "Ld" are all accepted as "ld" instructions. For purposes of discrimination from symbols, this manual uses lowercase characters for the reserved words.

4.5.2 Instructions (Mnemonics and Pseudo-instructions)

The assembler supports all the mnemonics of the E0C63000 instruction set and the assembler pseudo-instructions. The following shows how to describe the instructions.

Mnemonics

An instruction is generally composed of [mnemonic] + [operand]. Some instructions do not contain an operand.

General notation forms of instructions

General forms: **<Mnemonic>**
<Mnemonic> tab or space **<Operand>**
<Mnemonic> tab or space **<Operand1>, <Operand2>**
<Mnemonic> tab or space **<Operand1>, <Operand2>, <Operand3>**

Examples:
```
nop
jr   NMI
ld   %f,0x4
```

There is no restriction as to where the description of a mnemonic should begin in a line. A tab or space preceding a mnemonic is ignored.

An instruction containing an operand needs to be separated into the mnemonic and the operand with one or more tabs or spaces. If an instruction requires multiple operands, the operands must be separated from each other with one comma (,). Space between operands is ignored.

The elements of operands will be described further below.

Types of mnemonics

The following 39 types of mnemonics can be used in the E0C63 Family:

```
add adc and bit calr calz clr cmp dec ex halt inc int jp jr jrc jrnc jrnz
jrz ld ldb nop or pop push ret retd reti rets rl rr sbc set sll slp srl sub
tst xor
```

For details on instructions, refer to the "E0C63000 Core CPU Manual".

Note

The assembler is commonly used for all the E0C63 Family models, so all the instructions can be accepted. Be aware that no error will occur in the assembler even if instructions or operands unavailable for the model are described. They will be checked in the linker.

Assembler pseudo-instructions

The assembler pseudo-instructions are not converted to execution codes, but they are designed to control the assembler or to set data.
For discrimination from other instructions, all the assembler pseudo-instructions begin with a sharp (#) or a period (.).

General notation forms of pseudo-instructions

General forms: **<Pseudo-instruction>**
<Pseudo-instruction> tab or space **<Parameter>**
<Pseudo-instruction> tab or space **<Parameter1>** tab, space or comma **<Parameter2>** ...

Examples:
```
#define    SW1   1
.org       0x100
.comm      BUF   4
```

There is no restriction as to where the description of an instruction may begin in a line.

An instruction containing a parameter needs to be separated into the instruction and the parameter with one or more tabs or spaces. If an instruction requires multiple parameters, they are separated from each other with an appropriate delimiter.

Types of pseudo-instructions

The following 25 types of pseudo-instructions are available:

```
#include #define #macro #endm #ifdef #ifndef #else #endif #defnum
.abs .align .org .code .data .bss .codeword .word .comm .lcomm
.global .set .list .nolist .stabs .stabn
```

For details of each pseudo-instruction and its functionality, refer to Section 4.7, "Assembler Pseudo-Instructions".

Restriction

The mnemonics and pseudo-instructions are all not case sensitive. Therefore, they can be written in uppercase (A–Z) characters, lowercase (a–z) characters, or both. For example, "ld", "LD", and "Ld" are all accepted as "ld" instructions. However, the user defined symbols used in the operands or parameters are case sensitive. They must be the same with the defined characters. When assembling with the "-c" option, all symbols are case insensitive.

4.5.3 Symbols (Labels)

A symbol (label) is an identifier designed to refer to an arbitrary address in the program. It is possible to refer to a branch destination of a program or a data memory address using the defined symbol.

Definition of a symbol

Usable symbols are defined as 16-bit values by any of the following methods:

1. <Symbol>:
   Example: `LABEL1:`
   ... LABEL1 is a label that indicates the address of a described location.

   Preceding spaces and tabs are ignored. It is a general practice to describe from the top of a line.

2. Definition using the ***.set*** pseudo-instruction
   Example: `.set ADDR1 0xff00`
   ... ADDR1 is a symbol that represents absolute address 0xff00.

3. Definition using the ***.comm*** or ***.lcomm*** pseudo-instruction
   Example: `.comm BUF1 4`
   ... BUF1 is a label that represents a RAM address.

   The ***.comm*** and ***.lcomm*** pseudo instructions can define labels only in bss sections (data memory such as RAM). Program memory addresses cannot be defined.

Reference with symbols

A defined symbol denotes an address.
The actual address value should be determined in the linking process, except in the case of absolute sections.
Examples:

```
LABEL1:
        :
      jr      LABEL1           ... jumps to the LABEL1 location.

    .set   IO_M   0xfff0
    .org   0x0000
    .bss
    .comm  COUNT1 1

    .code
    ldb    %ext,IO_M@h
    ldb    %xl,IO_M@l    ... 0xfff0 is loaded to X-register. (@h and @l are symbol masks.)
    inc    [COUNT1]      ... Regarded as inc [0x0000].
```

Scope

The scope is a reference range of a symbol (label). It is called local if the symbol is to be referenced within the same file, and it is called global if the symbol is to be referenced from other files.
Any defined symbol's scope is local in default. To make a symbol's scope global, use the ***.global*** pseudo-instruction both in the file in which the symbol is defined and in the file that references the symbol.
A double definition of local symbols will be an error at the assembly stage, while a double definition of global symbols will be an error at the link stage.
Example:

File in which global symbol is defined (file1)

```
        .global    SYMBOL      ... Global declaration of a symbol which is to be defined in this file.
SYMBOL:
           :
LABEL:                         ... Local symbol
           :                           (Can be referenced to only in this file)
```

File in which a global symbol is referenced to (file2)

```
        .global    SYMBOL      ... Global declaration of a symbol defined in other source file.
        call       SYMBOL      ... Symbol externally referenced to.
           :
LABEL:                         ... Local symbol
           :                           (Treated as a different symbol from LABEL of file1)
```

The assembler regards those symbols as those of undefined addresses in the assembling, and includes that information in the object file it delivers. Those addresses are finally determined by the processing of the linker.

* When a symbol is defined by the ***.comm*** pseudo-instruction, that symbol will be a global symbol. Therefore, in a defined file, no global declaration needs to be made using the ***.global*** pseudo-instruction. On the contrary, in a file to be referenced, the global declaration is necessary prior to the reference.

Symbol masks

Symbol masks are designed to acquire the upper 8-bit address and the lower 8-bit address from a symbol representing a 16-bit address.
The following 5 types of symbol masks can be used:

@l or @L	Acquires the lower 8 bits of an absolute address.
@h or @H	Acquires the upper 8 bits of an absolute address.
@rl or @RL	Acquires the lower 8 bits of a relative address.
@rh or @RH	Acquires the upper 8 bits of a relative address.
@xh or @XH	Acquires the upper 8 bits of an absolute address by inverting them (Used exclusively for the "ldb" instruction combined with the "cmp" instruction).

Sample uses:

```
ldb     %ext,ADDR@h
ldb     %xl,ADDR@l             ... Functions as "ld  %x, ADDR (16-bit)"

ldb     %ext,NUM@h
add     %x,NUM@l               ... Functions as "add  %x, NUM (16-bit)"

ldb     %ext,LABEL@rh
calr    LABEL@rl               ... Functions as "calr  LABEL (16-bit)"

ldb     %ext,DATA@xh
cmp     %x,DATA@l              ... Functions as "cmp  %x, DATA (16-bit)"

.set    IO_ADDR   0xff12
ldb     %ext,IO_ADDR@l
ld      %a,[%y]                ... Functions as "ld  %a, [IO_ADDR]"
```

Restrictions

- The maximum number of characters of a symbol is 259 (not including colon). If this number is exceeded, an error will result.
- Only the following characters can be used:
  A–Z a–z _ 0–9 ?
- A symbol cannot begin with a numeral.
  Examples:

```
;OK                         ;Error
FOO:                        1lable:
L1:                         L 1:
.set   IO    0xfff0         .set    #IO      0xfff0
.comm  BUF   4              .lcomm  1st_BUF  2
```

- Since symbols are case sensitive by default, uppercase and lowercase are discriminated. When referencing a defined symbol, use the characters exactly the same as the defined symbol.
  Examples:

```
_Abcd:
           :
        jr    _ABCD         ... Does not jump to _Abcd
```

  However, symbols will be case insensitive if the -c option is specified.
- The symbol masks are effective only on the defined symbols. If a symbol mask is applied to a numeric value, an error will result.
- If a symbol mask is omitted, the lower bits effective for that instruction will be used. However, if the bit value does not fall within the instruction range, an error or warning will be issued.
- Symbols and symbol masks cannot be used on 4-bit immediate values.

4.5.4 Comments

Comments are used to describe a series of routines, or the meaning of each statement. Comments cannot comprise part of coding.

Definition of comment

A character string beginning with a semicolon (;) and ending with a line feed code (LF) is interpreted as a comment. Not only ASCII characters, but also other non-ASCII characters can be used to describe a comment.

Examples:

```
;This line is a comment line.
LABEL:       ;This is the comment for LABEL.
  ld  %a,%b  ;This is the comment for the instruction on the left.
```

Restrictions

- A comment is allowed up to 259 characters, including a semicolon (;), spaces before, after and inside the comment, and a return/line feed code.
- When a comment extends to several lines, each line must begin with a semicolon.
  Examples:

```
;These are
  comment lines.     ... The second line will not be regarded as a comment. An error will
                         result.
;These are
; comment lines.     ... Both lines will be regarded as comments.
```

4.5.5 Blank Lines

This assembler also allows a blank line containing only a return/line feed code. It need not be made into a comment line using a semicolon.

4.5.6 Register Names

The CPU register names may be written in either uppercase or lowercase letters.

Table 4.5.6.1 Notations of register names

Register		Notation
A	Data register A	%a, %A, a or A
B	Data register B	%b, %B, b or B
BA	BA-register pair	%ba, %BA, ba or BA
X	Index register X	%x, %X, x or X
XH	Upper 8 bits of X-register	%xh, %XH, xh or XH
XL	Lower 8 bits of X-register	%xl, %XL, xl or XL
Y	Index register Y	%y, %Y, y or Y
YH	Upper 8 bits of Y-register	%yh, %YH, yh or YH
YL	Lower 8 bits of Y-register	%yl, %YL, yl or YL
F	Flag register F	%f, %F, f or F
EXT	Extension register EXT	%ext, %EXT, ext or EXT
SP1	Stack pointer SP1	%sp1, %SP1, sp1 or SP1
SP2	Stack pointer SP2	%sp2, %SP2, sp2 or SP2

Note: "%" can be omitted. These symbols are reserved words, therefore they cannot be used as user-defined symbol names.

4.5.7 Numerical Notations

This Assembler supports three kinds of numerical notations: decimal, hexadecimal, and binary.

Decimal notations of values

Notations represented with 0–9 only will be regarded as decimal numbers. To specify a negative value, put a minus sign (-) before the value.

Examples: `1   255   -3`

Characters other than 0–9 and the sign (-) cannot be used.

Hexadecimal notations of values

To specify a hexadecimal number, place "0x" before the value.

Examples: `0x1a   0xff00`

"0x" cannot be followed by characters other than 0–9, a–f, and A–F.

Binary notations of values

To specify a binary number, place "0b" before the value.

Examples: `0b1001   0b1001100`

"0b" cannot be followed by characters other than 0 or 1.

Specified ranges of values

The size (specified range) of immediate data varies with each instruction.
The specifiable ranges of different immediate data are given below.

Table 4.5.7.1 Types of immediate data and their specifiable ranges

Symbol	Type	Decimal	Hexadecimal	Binary
imm2	2-bit immediate data	0–3	0x0–0x3	0b0–0b11
imm4	4-bit immediate data	0–15	0x0–0xf	0b0–0b1111
imm6	Software vectored interrupt address	0–64	0x0–0x3f	0b0–0b111111
imm8	8-bit immediate data	0–255	0x0–0xff	0b0–0b11111111
n4	4-bit n-ary specified data	1–16	0x1–0x10	0b0–0b10000
sign8	Signed 8-bit immediate data	-128–127	0x0–0xff	0b0–0b11111111
add6	6-bit address	0–64	0x0–0x3f	0b0–0b111111

Other numerical notations

The following numerical notations can also be used:

nnnnB: Binary numbers
nnnnO: Octal numbers
nnnnQ: Octal numbers
nnnnH: Hexadecimal numbers

"nnnnB" (binary numbers) and "nnnnH" (hexadecimal numbers) are converted into the new format ("0bnnnn" and "0xnnnn") in the preprocessing stage.
"nnnnO" and "nnnnQ" (octal numbers) are converted into hexadecimal numbers ("0xnnnn") in the preprocessing stage.

ASCII to HEX conversion

One or two ASCII characters (enclosed with ' ') can be described in source files unless converting into numbers. The numeric operators can also be used. The described characters are converted into ASCII codes and delivered to the output relocatable object file.

Examples:
```
retd  '1'     →  (retd  0x31)
retd  '23'    →  (retd  0x3233)
retd  '4'+1   →  (retd  0x35)
```

Note: Three or more characters and the following characters cannot be described:
Control codes (0x0 to 0x1f) space @ [ ] ; ,

4.5.8 Operators

An expression that consists of operators, numbers and/or defined symbols (including labels) can be used for specifying a number or defining a Define name (only for number definition).
The preprocess in the assembler handles expressions in signed 16-bit data and expands them as hexadecimal numbers.

Types of operators

Arithmetic operators		Examples	
+	Addition, Plus sign	`+0xff, 1+2`	
-	Subtraction, Minus sign	`-1+2, 0xff-0b111`	
*	Multiplication	`0xf*5`	
/	Division	`0x123/0x56`	
%	Residue	`0x123%0x56`	(%% is also be supported.)
>>	Shifting to right	`1>>2`	
<<	Shifting to left	`0x113<<3`	
^H	Acquires upper 8 bits	`0x1234^H`	
^L	Acquires lower 8 bits	`0x1234^L`	
( )	Parenthesis	`1+(1+2*5)`	

The arithmetic operator returns the result of arithmetic operation on the specified terms.

Logical operators		Examples
&	Bit AND	`0b1101&0b111`
|	Bit OR	`0b123|0xff`
^	Bit XOR	`12^35`
~	Bit inversion	`~0x1234`

The logical operator returns the result of logic operation on the specified terms.

Relational operators		Examples
==	Equal	`SW==0`
!=	Not equal	`SW!=0`
<	Less than	`ABC<5`
<=	Less than or equal	`ABC<=5`
>	Greater than	`ABC>5`
>=	Greater than or equal	`ABC>=5`
&&	AND	`ABC&&0xf`
||	OR	`ABC||0b1010`

The relational operator returns 1 if the expression is true, otherwise it returns 0.

Priority

The operators have the priority shown below. If there are two or more operators with the same priority in an expression, the assembler calculates the expression from the left.

1. ( ) High priority
2. + (plus sign), - (minus sign), ~ ↑
3. ^H, ^L
4. *, /, % (%%)
5. + (addition), - (subtraction)
6. <<, >>
7. ==, !=, <, <=, >, >=
8. &
9. ^
10. |
11. && ↓
12. || Low priority

Examples

```
#defnum BLK_HEADER_SIZE  4
#defnum BLK_START  0x30+BLK_HEADER_SIZE*2
#defnum BLK_END    BLK_START+4*2

#macro  ADD_X   ADDR
        ldb     %ext,(ADDR*2)^H          ... Can be used in macros.
        add     %x,(ADDR*2)^L
#endm

        ldb     %ext,BLK_START^H   ; %x=BLK_START
        ldb     %xl, BLK_START^L
        ld      [%x],0b11&0x110
        ldb     %ext, ~BLK_END^H   ; cmp %x, BLK_END
        cmp     %x,  BLK_END^L
        ADD_X   (0x1200+0x34)*2    ; %x+=0x1234*2
```

Precautions

- Minus numbers -1 to -32768 are handled as 0xffff to 0x8000.
- The assembler handles expressions as 16-bit data. Pay attention to the data size when using it as 4-bit immediate data, especially when it has a minus value.
  Example:
  ```
  ld   %a,-2+1          ... NG. It will be expanded as "ld  a,0xffff".
  ld   %a,(-2+1)&0xf    ... OK. It will be expanded as "ld  a,0xf".
  ```
- Expressions are calculated with a sign (like a signed short in C language).
  Pay attention to the calculation results of the >>, / and % operators using hexadecimal numbers.
  Example:
  ```
  .set NUM1 0xfffe/2    ... -2/2 = -1 (0xffff)
                        The / and % operators can only be used within the range of +32767 to -32768.
  .set NUM2 0xfffe>>1   ... -2>>1 = -1 (0xffff)
                        Mask as (0xfffe>>1)&0x7fff.
  ```
- When using an expression in a #define statement, it will be expanded as is. Pay attention when a number is defined using the #define pseudo-instruction.
  Example:
  ```
  #define NUM1 1+1
  ld      %a,NUM1*2     ... This will be expanded as "ld  %a, 1+1*2" (=3).
  #define NUM2 (1+1)
  ld      %a,NUM2*2     ... This will be expanded as "ld  %a, (1+1)*2" (=4).
  ```
- Do not insert a space or a tab between an operator and a term.

4.5.9 Location Counter Symbol "$"

The address of each instruction code is set in the 16-bit location counter when a statement is assembled. It can be referred using a symbol "$" as well as labels. "$" indicates the current location, thus it can be used for relative branch operation. The operators can be used with this symbol similar to labels.

Example:
```
jr  $                    ... Jumps to this address (means endless loop).
jr  $+2                  ... Jumps to two words after this address.
jr  $-10                 ... Jumps to 10 words before this address.
jr  $+16+(16*(BLK>16))   ... Operators and defined symbols can be used.
```

Precaution

When the address referred to relatively with "$" is in another section, it should be noted if the intended section resides at the addressed place, because if the section is relocatable, the absolute address is not fixed until the linking is completed.

4.5.10 Optimization Branch Instructions for Old Preprocessor

The old version of the E0C63 preprocessor has optimization branch instructions for optimizing the extension code. Since this function is supported by the linker in the current version, they are expanded without an extension code in the assembler. The relative distance to the label does not affect this expansion.

Optimization Branch Instruction		Mnemonic after Expansion
xjr LABEL	→	jr LABEL
xjrc LABEL	→	jrc LABEL
xjrnc LABEL	→	jrnc LABEL
xjrz LABEL	→	jrz LABEL
xjrnz LABEL	→	jrnz LABEL
xcalr LABEL	→	calr LABEL

4.6 Section Management

4.6.1 Definition of Sections

The memory configuration of the E0C63 Family microcomputer is divided into a code ROM that contains programs written, and data memories such as data RAM and I/O memory. Moreover, some models carry a data ROM that holds static data written.
A section refers to an area where codes are written (or to be mapped), and there are three types of sections in correspondence with the memories:

1. CODE section Area located within a code ROM.
2. DATA section Area located within a data ROM.
3. BSS section Denotes a RAM area.

To allow to specify these sections in a source file, the assembler comes provided with pseudo-instructions.

CODE section

The ***.code*** pseudo-instruction defines a CODE section. Statements from this instruction to another section defining instruction will be regarded as program codes, and will be so processed as to be mapped in the code ROM. The source file will be regarded as a CODE section by default. Therefore, the part that goes from top of the file, to another section will be processed as a CODE section. Because this section is of 13 bits/word, 4-bit data cannot be defined.

DATA section

The ***.data*** pseudo-instruction defines a DATA section. Statements from this instruction to another section defining instruction will be regarded as 4-bit data, and will be so processed as to be mapped in the data ROM. Therefore, nothing else can be described in this area other than the symbols for referring to the address of the data ROM, the 4-bit data defining pseudo-instruction (***.word***), and comments. This section is applied only to models having a data ROM.

BSS section

The ***.bss*** pseudo-instruction defines a BSS section. Statements from this instruction to another section defining instruction will be regarded as 4-bit data, and will be so processed as to be mapped in the data memory (RAM). Therefore, nothing else can be described in this area other than the symbols for referring to the address of the data memory, the area securing pseudo-instructions (***.comm*** and ***.lcomm***).
The ***.comm*** pseudo-instruction and the ***.lcomm*** pseudo-instruction are designed to define the symbol and size of a data area. Although the BSS section basically consists in a RAM area, it can as well be used as a data memory area, such as display memory and I/O memory. Since code definition in this area is meaningless in embedded type microcomputers, such as those of the E0C63 Family, nothing else can be described other than the two instructions and comments.

4.6.2 Absolute and Relocatable Sections

The assembler is a relocatable assembler that always generates an relocatable object and needs the linker to make it into an executable absolute object. However, each section in one source can be absolute or relocatable depending on how they are described. The section whose absolute address is specified with the ***.org*** pseudo-instruction in the source is an absolute section, while the section whose absolute address is not specified is an relocatable section. Absolute addresses of relocatable sections will be fixed by the linker. Both types of sections can be included in one source.

4.6.3 Sample Definition of Sections

```
            :
        CODE1 (Relocatable program)
            :
.data
            :
        DATA1 (Relocatable data definition)
            :
.bss
            :
        BSS1 (Relocatable RAM area definition)
            :
.code
.org    0x0         ... If this specification is omitted, a CODE section begins from the address following CODE1.
            :
        CODE2 (Absolute program)
            :
.bss
.org    0x0         ... If this specification is omitted, a BSS section begins from the address following BSS1.
            :
        BSS2 (Absolute RAM area definition)
            :
.code
            :
        CODE3 (Relocatable program)
            :
.data
.org    0x8000      ... If this specification is omitted, a DATA section begins from the address following DATA1.
            :
        DATA2 (Absolute data definition)
            :
```

In the section definition shown above, absolute sections and relocatable sections are mixed in one source. Absolute sections are sections whose absolute addresses are specified with the ***.org*** pseudo-instructions. CODE2, BSS2 and DATA2 are absolute sections. Absolute sections will be located at the place specified.

Other sections are relocatable in the sense that the absolute location addresses are not fixed at the assembly stage and will be fixed later at the linking stage.

Precautions

When there appears in a section a statement which is designed for other section, a warning will be issued and a new section will be started according to the statement.

Examples:
```
.code
.comm BUF 16   ... Warning; A new bss section begins
.bss
ld    %a,%b    ... Warning; A new code section begins
```

4.7 Assembler Pseudo-Instructions

The assembler pseudo-instructions are not converted to execution codes, but they are designed to control the assembler or to set data.
For discrimination from other instructions, all the assembler pseudo-instructions begin with a character "#" or ".". The instructions that begin with "#" are preprocessed pseudo-instructions and they are expanded into forms that can be assembled. The expanded results are delivered in the preprocessed file (.ms). The original statements of the pseudo-instructions (#) are changed as comments by attaching a ";" before delivering to the file. The instruction that begins with "." are used for section and data definitions. They are not converted at the preprocessing stage.
All the pseudo-instruction characters are not case sensitive.
The following pseudo-instructions are available in the assembler:

Pseudo-instruction	Function
#include	Includes another source.
#define	Defines a constant string.
#defnum	Defines a constant number. (∗1)
#macro–#endm	Defines a macro.
#ifdef–#else–#endif	Defines an assemble condition.
#ifndef–#else–#endif	Defines an assemble condition.
.abs	Specifies absolute assembling. (∗1)
.align	Sets alignment of a section.
.org	Sets an absolute address.
.code	Declares a CODE section (mapping to the built-in code ROM).
.data	Declares a DATA section (mapping to the built-in data ROM).
.bss	Declares a BSS section (mapping to the built-in RAM).
.codeword	Defines data in the CODE section.
.word	Defines data in the DATA section.
.comm	Secures a global area in the BSS section.
.lcomm	Secures a local area in the BSS section.
.global	Defines an external reference symbol.
.set	Defines an absolute address symbol.
.list	Controls assembly list output.
.nolist	Controls assembly list output.
.stabs	Debugging information (source name).
.stabn	Debugging information (line number).

∗1: Maintained only for compatibility with the older assembler.

4.7.1 Include Instruction (#include)

The include instruction inserts the contents of a file in any location of a source file. It is useful when the same source is shared in common among several source files.

Instruction format

#include "<File name>"

- A drive name or path name can as well be specified as the file name.
- One or more spaces are necessary between the instruction and the "<File name>".
- Character case is ignored for both ***#include*** itself and "<File name>".

Sample descriptions:

```
#include     "sample.def"
#include     "c:\E0C63\header\common.h"
```

Expansion rule

The specified file is inserted in the location where ***#include*** was described.

Precautions

- Only files created in text file format can be inserted.
- The ***#include*** instruction can be used in the including files. However, nesting is limited up to 10 levels. If this limit is surpassed, an error will result.

4.7.2 Define Instruction (#define)

Any substitute character string can be left defined as a Define name by the define instruction (***#define***), and the details of that definition can be referred to from various parts of the program using the Define name.

Instruction format

#define <Define name> [<Substitute character string>]

<Define name>:
- The first character is limited to a–z, A–Z, ? and _.
- The second and the subsequent characters can use a–z, A–Z, 0–9, ? and _.
- Uppercase and lowercase characters are discriminated. (***#define*** itself is case insensitive.) When assembling with the "-c" option, all symbols are case insensitive.
- One or more spaces or tabs are necessary between the instruction and the Define name.

<Substitute character string>:
- When writing all characters can be used, but a semicolon (;) is interpreted as the start of a comment.
- Uppercase and lowercase characters are discriminated.
- One or more spaces or tabs are necessary between the Define name and the substitute character string.
- The substitute character string can be omitted. In that case, NULL is defined in lieu of the substitute character string. It can be used for the conditional assembly instruction.

Sample definitions:

```
#define   TYPE1
#define   L1      LABEL_01
#define   Xreg    %x
#define   CONST   (DATA1+DATA2)*2
```

Expansion rule

If a Define name defined appears in the source, the assembler substitutes a defined character string for that Define name.

Sample expansion:

```
#define INT_F1     0xfff0
#define INT_F1_1   0
           :
        set   [INT_F1], INT_F1_1     ... Expanded to "set  [0xfff0],0".
           :
```

Precautions

- The assembler only permits backward reference of a Define name. Therefore the name definition must precede the use of it.
- Once a Define name is defined, it cannot be canceled. However, redefinition can be made using another Define name.
  Example:
  ```
  #define    XL    %xl
  #define    Xlow  XL
             ldb   [Xlow],%ba         ... Expanded to "ldb  [%xl],%ba".
  ```
- When the same Define name is defined duplicatedly, a warning message will appear. Until it is redefined, it is expanded with the original content, and once it is redefined, it is expanded with the new content.
- No other characters than delimiters (space, tab, line feed, and comma) can be added before and after a Define name in the source, unless they are enclosed in [ ] or [ ]+. However, an operator or a symbol mask (@..) can be added to a Define name string without delimiters.
  Examples:
  ```
  #define    INT_F  0xfff
             tst    [INT_F1],0 ;tst [0xfff1],0? ... Specification like this is invalid.

  #define    L      LABEL
             ldb    %ext,L@h          ... Replaced with "ldb  %ext,LABEL@h".
             ldb    %xl,L@l           ... Replaced with "ldb  %xl,LABEL@l".
  ```
- When using an expression in a #define statement, it will be expanded as is. Pay attention when a number is defined using the #define pseudo-instruction.
  Examples:
  ```
  #define    NUM1   1+1
             ld     %a,NUM1*2         ... Expanded as "ld  %a, 1+1*2" (=3).

  #define    NUM2   (1+1)
             ld     %a,NUM2*2         ... Expanded as "ld  %a, (1+1)*2" (=4).
  ```
- The internal preprocess part of the assembler does not check the validity of a statement as the result of the replacement of the character string.

4.7.3 Numeric Define Instruction (#defnum)

Instruction format

#defnum <Numeric Define name> <Number>

Function

The ***#defnum*** pseudo-instruction is provided for compatibility with the older assembler. In the older assembler, ***#defnum*** is required to define a numeric constant, while ***#define*** is for defining a string. In the new assembler, there is no need to differentiate between a numeric constant and a string. Therefore the new assembler should use the ***#define*** instruction.

4.7.4 Macro Instructions (#macro ... #endm)

Any statement string can be left defined as a macro using the macro instruction (***#macro***), and the content of that definition can be invoked from different parts of the program with the macro name. Unlike a subroutine, the part that is invoking a macro is replaced with the content of the definition.

Instruction format

```
#macro   <Macro name>     [<Dummy parameter>] [,<Dummy parameter>] ...
         <Statement string>
#endm
```

<Macro name>:
- The first character is limited to a–z, A–Z, ? and _.
- The second and the subsequent characters can use a–z, A–Z, 0–9, ? and _.
- Uppercase and lowercase characters are discriminated. (***#macro*** itself is case insensitive.) When assembling with the "-c" option, all symbols are case insensitive.
- One or more spaces or tabs are necessary between the instruction and the macro name.

<Dummy parameter>:
- Dummy parameter symbols for macro definition. They are described when a macro to be defined needs parameters.
- One or more spaces or tabs are necessary between the macro name and the first parameter symbol. When describing multiple parameters, a comma (,) is necessary between one parameter and another.
- The same symbols as for a macro name are available.
- The number of parameters are limited according to the free memory space.

<Statement string>:
- The following statements can be described:
  - Basic instruction (mnemonic and operand)
  - Conditional assembly instruction
  - Internal branch label*
  - Comments
- The following statements cannot be described:
  - Assembler pseudo-instructions (excluding conditional assembly instruction)
  - Other labels than internal branch labels
  - Macro invocation

* ***Internal branch label***

A macro is spread over to several locations in the source. Therefore, if you describe a label in a macro, a double definition will result, with an error issued. So, use internal branch labels which are only valid within a macro.
- The number of internal-branch labels are limited according to the free memory space.
- The same symbols as for a macro name are available.

Sample definition:

```
#define  C_RESET  0b1101
#macro   WAIT     COUNT
         ld       %a,COUNT
         and      %f,C_RESET
LOOP:
         nop
         jr       LOOP
#endm
```

Expansion rules

When a defined macro name appears in the source, the assembler inserts a statement string defined in that location.
If there are actual parameters described in that process, the dummy parameters will be replaced with the actual parameters in the same order as the latter are arranged.
The internal branch labels are replaced, respectively, with __L0001 ... from top of the source in the same order as they appear.

Sample expansion:
When the macro WAIT shown above is defined:
Macro invocation

```
                :
              WAIT 15
                :
```

After expansion

```
                :
              ;WAIT    15
              ld    %a,15
              and   %f,0b1101
    __L0001:
              nop
              jr    __L0001
```

("__L0001" denotes the case where an internal branch label is expanded for the first time in the source.)

Precautions

- The assembler only permits backward reference of a macro invocation. Therefore the macro definition must precede the use of it.
- Once a defined macro name is defined, it cannot be canceled. If the same macro name is defined duplicatedly, a warning message will appear. Until it is redefined, it is expanded with the original content, and once it is redefined, it is expanded with the new content. Definition should be done with distinct names, although the program operation will not be affected.
- No other characters than delimiters (space, tab, line feed, and commas) can be added before and after a dummy parameter in a statement.
- The same character string as that of the define instruction cannot be used as a macro name.
- When the number of dummy parameters differs from that of actual parameters, an error will result.
- The maximum number of parameters and internal branch labels are limited according to the free memory space.
- "__Lnnnn" used for the internal branch labels should not be employed as other label or symbol.

4.7.5 Conditional Assembly Instructions (#ifdef ... #else ... #endif, #ifndef... #else ... #endif)

A conditional assembly instruction determines whether assembling should be performed within the specified range, dependent on whether the specified name (Define name) is defined or not.

Instruction formats

Format 1)

```
#ifdef      <Name>
             <Statement string 1>
[#else
            <Statement string 2> ]
#endif
```

If the name is defined, <Statement string 1> will be subjected to the assembling.
If the name is not defined, and #else ... <Statement string 2> is described, then <Statement string 2> will be subjected to the assembling. #else ... <Statement string 2> can be omitted.

Format 2)

```
#ifndef     <Name>
             <Statement string 1>
[#else
            <Statement string 2> ]
#endif
```

If the name is not defined, <Statement string 1> will be subjected to the assembling.
If the name is defined, and #else ... <Statement string 2> is described, <Statement string 2> will be subjected to the assembling. #else ... <Statement string 2> can be omitted.

<Name>:
Conforms to the restrictions on Define name. (See ***#define***.)

<Statement string>:
All statements, excluding conditional assembly instructions, can be described.

Sample description:

```
#ifdef   TYPE1
         ld     %x,0x12
#else
         ld     %x,0x13
#endif

#ifndef  SMALL
#define  STACK1 0x31
#endif
```

Name definition

Name definition needs to have been completed by either of the following methods, prior to the execution of a conditional assembly instruction:

(1) Definition using the start-up option (-d) of the assembler.
Example: `as63 -d TYPE1 sample.s`

(2) Definition in the source file using the ***#define*** instruction.
Example: `#define TYPE1`

The ***#define*** statement is valid even in a file to be included, provided that it goes before the conditional assembly instruction that uses its Define name. A name defined after a conditional assembly instruction will be regarded as undefined.

When a name is going to be used only in conditional assembly, no substitute character string needs to be specified.

Expansion rule

A statement string subjected to the assembling is expanded according to the expansion rule of the other preprocessing pseudo-instructions. (If no preprocessing pseudo-instruction is contained, the statement will be output in a file as is.)

Precaution

A name specified in the condition is evaluated with discrimination between uppercase and lowercase. When assembling with the "-c" option, all symbols are case insensitive.
The condition is deemed to be satisfied only when there is the same Define name defined.

4.7.6 Section Defining Pseudo-Instructions (.code, .data, .bss)

The section defining pseudo-instructions define one related group of codes or data and make it possible to relocate by the groups at the later linking stage. Even if these section defining pseudo-instructions are not used, the section kind will be automatically judged by its contents (however, a warning occurs). If the new codes or data without section definition are different from the previous code or data kind, they will be taken as another new section.

.code pseudo-instruction

Instruction format

.code

Function

Declares the start of a CODE section. Statements following this instruction are assembled as those to be mapped in the code ROM, until another section is declared.
The CODE section is set by default in the assembler. Therefore, the ***.code*** pseudo-instruction can be omitted at top of a source file. Always describe it when you change a section to a CODE section.

Precautions

- A CODE section can be divided among multiple locations of a source file for purpose of definition (describing the ***.code*** pseudo-instruction in the respective start positions).
- A CODE section is relocatable by default unless its location is specified with the ***.org*** pseudo-instruction or more loosely with the ***.align*** pseudo-instruction.

.data pseudo-instruction

Instruction format

.data

Function

Declares the start of a DATA section. Statements following this instruction are assembled as those to be mapped in the data ROM, until another section is declared.

Precautions

- The DATA section is a static data area, and effective only for models with data ROM installed.
- In a DATA section, nothing other than the ***.org*** and ***.word*** pseudo-instructions, symbols, and comments can be described.
- A DATA section can be divided among multiple locations of a source file for purpose of definition (describing the ***.data*** pseudo-instruction in the respective start positions).
- A DATA section is relocatable by default unless its location is specified with the ***.org*** pseudo-instruction or more loosely with the ***.align*** pseudo-instruction.

.bss pseudo-instruction

Instruction format

.bss

Function

Declares the start of a BSS section. Statements following this instruction are assembled as those to be mapped in the RAM, until another section is declared.

Precautions

- In a BSS section, nothing else other than the ***.comm***, ***.lcomm***, and ***.org*** pseudo-instructions, symbols, and comments can be described.
- A BSS section can be divided among multiple locations of a source file for purpose of definition (describing the ***.bss*** pseudo-instruction in the respective start positions).
- A BSS section is relocatable by default unless its location is specified with the ***.org*** pseudo-instruction or more loosely with the ***.align*** pseudo-instruction.

4.7.7 Location Defining Pseudo-Instructions (.org, .align)

The absolute addressing pseudo-instructions (***.align*** and ***.org***) work to specify absolute location of a section in different precision such as 2^n words alignment level and complete absolute address level.

.org pseudo-instruction

Instruction format

.org <Address>

<Address>:

Absolute address specification

- Only decimal, binary and hexadecimal numbers can be described.
- The addresses that can be specified are from 0 to 65,535 (0xffff).
- One or more spaces or tabs are necessary between the instruction and the address.

Sample description:

```
.code
.org    0x0100
```

Function

Specifies an absolute address location of a CODE, DATA or BSS section in an assembly source file. The section with the ***.org*** pseudo-instruction is taken as an absolute section.

Precautions

- If an overlap occurs as the result of specifying absolute locations with the ***.org*** pseudo-instruction, an error will result.

  Examples:

  ```
  .bss
  .org   0x00
  .comm  RAM0   4     ... RAM secured area (0x00–0x03)
  .org   0x01
  .comm  RAM1   4     ... Error (because the area of 0x01–0x03 is overlapped)
  ```

- When the ***.org*** pseudo-instruction appears in a section, a new absolute section starts at that point. The section type does not change. The ***.org*** pseudo-instruction keeps its effect only in that section until the next section definer (***.code***, ***.data*** or ***.bss***) or the next location definer (***.org*** or ***.align)*** appears.

  Example:

  ```
        :
  .code              ... The latest relocatable section definition.
        :
  .org  0x100        ... Starts new absolute CODE section from address 0x100.
        :
  .bss               ... This section is relocatable not affected by the ".org" pseudo-instruction.
        :
  .code              ... This section is also relocatable not affected by the ".org" pseudo-instruction.
        :
  ```

- If the ***.org*** pseudo-instruction is defined immediately after a section definer (***.code***, ***.data*** or ***.bss***), the section definer does not start a new section. But ***.org*** starts a new section with the attribute of the section definer.

  Example:

  ```
  .code              ... This does not start a new CODE section.
  .org  0x100        ... This starts an absolute CODE section.
        :
  ```

- If the ***.org*** pseudo-instruction is defined immediately before a section definer (***.code***, ***.data*** or ***.bss***), it does not start a new section and makes no effect to the following sections.

  Example:

```
.code             ... The latest relocatable section definition.
     :
.org 0x100        ... This does not start a new absolute section and makes no effect.
.bss              ... The another kind (BSS) of section which is not affected by the
     :                previous ".org" pseudo-instruction in the CODE section.
.code             ... This will be an relocatable CODE section not affected by the
     :                previous ".org" pseudo-instruction.
```

.align pseudo-instruction

Instruction format

.align <Alignment number>

<Alignment number>:

Word alignment in 2^n value
- Only decimal, binary and hexadecimal numbers can be described.
- The alignment that can be specified is a 2^n value.
- One or more spaces or tabs are necessary between the instruction and the alignment number.

Sample description:

```
.code
.align  32      ... Sets the location to the next 32-word boundary address.
```

Function

Specifies location alignment in words of a CODE, DATA or BSS section in an assembly source file. The section with the ***.align*** pseudo-instruction can be taken as a loosely absolute section in the sense that its location is partially defined.

Precautions

- When the ***.align*** pseudo-instruction appears in a section, a new absolute section starts at that point. The section type does not change. The ***.align*** pseudo-instruction keeps its effect only in that section until the next section definer (***.code***, ***.data*** or ***.bss***) or the next location definer (***.org*** or ***.align)*** appears.

  Example:

```
         :
.code          ... The latest relocatable section definition.
         :
.align 32      ... Starts new loosely absolute CODE section from the next 32-word boundary address.
         :
.bss           ... This section is relocatable not affected by the ".align" pseudo-instruction.
         :
.code          ... This section is also relocatable not affected by the ".align" pseudo-instruction.
         :
```

- If the ***.align*** pseudo-instruction is defined immediately after a section definer (***.code***, ***.data*** or ***.bss***), the section definer does not start a new section. But ***.align*** starts a new section with the attribute of the section definer.

  Example:

```
.code          ... This does not start a new CODE section.
.align 32      ... This starts a loosely absolute CODE section.
         :
```

- If the ***.align*** pseudo-instruction is defined immediately before a section definer (***.code***, ***.data*** or ***.bss***), it does not start a new section and makes no effect to the following sections.

  Example:

```
.code          ... The latest relocatable section definition.
         :
.align 32      ... This does not start a new absolute section and makes no effect.
.bss           ... The another kind (BSS) of section which is not affected by the
         :         previous ".align" pseudo-instruction in the CODE section.
.code          ... This will be an relocatable CODE section not affected by the
         :         previous ".align" pseudo-instruction.
```

4.7.8 Absolute Assembling Pseudo-Instruction (.abs)

Instruction format

.abs

Function

The ***.abs*** pseudo-instruction is provided for compatibility with the older assembler. In the older assembler, this pseudo-instruction is required to specify that a source file uses absolute sections as opposed to relocatable sections. It is not necessary to use this instruction in the new assembler, because the new assembler allows the use of absolute and relocatable sections in one source file. Use the ***.org*** or ***.align*** pseudo-instruction for defining absolute sections.

4.7.9 Symbol Defining Pseudo-Instruction (.set)

Instruction format

.set <Symbol>[,] <Value>

<Symbol>:

Symbols for value reference

- The 1st character is limited to a–z, A–Z, ? and _.
- The 2nd and the subsequent character can use a–z, A–Z, 0–9, ? and _.
- Uppercase and lowercase are discriminated.
  When assembling with the "-c" option, all symbols are case insensitive.
- One or more spaces, or tabs are necessary between the instruction and the symbol.

<Value>:

Value specification

- Only decimal, binary, and hexadecimal numbers can be described.
- The values that can grammatically be specified are from 0 to 65,535 (0xffff).
- One or more spaces, tabs, or a comma (,) are necessary between the instruction and the value.

Sample description:

```
.set    DATA1  0x20
.set    STACK1 0x100
```

Function

Defines a symbol for a value such as an absolute address.

Precaution

When the defined symbol is used as an operand, the defined value is referred as is. Therefore, if the value exceeds the valid range of the operand, a warning will result.

Example:

```
.set DATA1  0xff00
ldb  %ext,DATA1@h    ... OK
ldb  %xl,DATA1@l     ... OK
ld   %a,DATA1        ... Warning
```

4.7.10 Data Defining Pseudo-Instructions (.codeword, .word)

.codeword pseudo-instruction

Instruction format

.codeword <Data>[,<Data> ...,<Data>]

<Data>:

13-bit data

- Only decimal, binary and hexadecimal numbers can be described.
- The data that can be specified are from 0 to 8,191 (0x1fff).
- One or more spaces or tabs are necessary between the instruction and the first data.
- A comma (,) is necessary between one data and another.

Sample description:

```
.code
.codeword    0xa,0xa40,0xff3
```

Function

Defines 13-bit data to be written to the code ROM.

Precaution

The ***.codeword*** pseudo-instruction can be used only in CODE sections.

.word pseudo-instruction

Instruction format

.word <Data>[,<Data> ...,<Data>]

<Data>:

4-bit data

- Only decimal, binary and hexadecimal numbers can be described.
- The data that can be specified are from 0 to 15 (0xf).
- One or more spaces or tabs are necessary between the instruction and the first data.
- A comma (,) is necessary between one data and another.

Sample description:

```
.data
.word   0xa,0xb,0xc,0xd
```

Function

Defines 4-bit data to be written to the data ROM.

Precaution

The ***.word*** pseudo-instruction can be used only in DATA sections.

4.7.11 Area Securing Pseudo-Instructions (.comm, .lcomm)

Instruction format

.comm <Symbol>[,] <Size>
.lcomm <Symbol>[,] <Size>

<Symbol>:

Symbols for data memory access (address reference)

- The 1st character is limited to a–z, A–Z, ? and _.
- The 2nd and the subsequent character can use a–z, A–Z, 0–9, ? and _.
- Uppercase and lowercase are discriminated.
  When assembling with the "-c" option, all symbols are case insensitive.
- One or more spaces or tabs are necessary between instruction and symbol.

<Size>:

Number of words of the area to be secured (4 bits/word)

- Only decimal, binary and hexadecimal numbers can be described.
- The size that can grammatically be specified is from 0 to 65,534.
- One or more spaces, tabs or a comma (,) are necessary between symbol and size.

Sample description:

```
.bss
.comm   RAM0  4
.lcomm  BUF,1
```

Function

Sets an area of the specified size in the BSS section (RAM and other data memory), and creates a symbol indicating its top address with the specified name. By using this symbol, you can describe an instruction to access the RAM.

Difference between .comm and .lcomm

The ***.comm*** pseudo-instruction and the ***.lcomm*** pseudo-instruction are exactly the same in function, but they do differ from each other in the scope of the symbols they create. The symbols created by the ***.comm*** pseudo-instruction become global symbols, which can be referred to externally from other modules (however, the file to be referred to needs to be specified by the ***.global*** pseudo-instruction.) The symbols created by the ***.lcomm*** pseudo-instruction are local symbols, which cannot be referred to from other modules.

Precaution

The ***.comm*** and ***.lcomm*** pseudo-instructions can only be described in BSS sections.

4.7.12 Global Declaration Pseudo-Instruction (.global)

Instruction format

.global <Symbol>

<Symbol>:

Symbol to be defined in the current file, or symbol already defined in other module

• One or more spaces or tabs are necessary between the instruction and the symbol.

Sample description:

```
.global   GENERAL_SUB1
```

Function

Makes global declaration of a symbol. The declaration made in a file with a symbol defined converts that symbol to a global symbol which can be referred to from other modules. Prior to making reference, declaration has to be made by this instruction on the side of the file that is going to make the reference.

4.7.13 List Control Pseudo-Instructions (.list, .nolist)

Instruction format

.list
.nolist

Function

Controls output to the relocatable list file.
The ***.nolist*** pseudo-instruction stops output to the relocatable list file after it is issued.
The ***.list*** pseudo-instruction resumes from there the output which was stopped by the ***.nolist*** pseudo-instruction.

Precaution

The assembler delivers relocatable list files only when it is started up with the -l option specified. Therefore, these instructions are invalid, if the -l option was not specified.

4.7.14 Source Debugging Information Pseudo-Instructions (.stabs, .stabn)

Instruction formats

(1) .stabs "<File name>", FileName
(2) .stabn 0, FileEnd
(3) .stabn <Line number>, LineInfo

Function

The assembler outputs object files in IEEE-695 format, including source debugging information conforming to these instructions. This debugging information is necessary to perform debugging by Debugger db63, with the assembly source displayed.

Format (1) delivers information on the start position of a file.
Format (2) delivers information on the end position of a file.
Format (3) delivers information on the line No. of an instruction in a source file.

Insertion of debugging information

When the -g option is specified as a start option, the preprocess stage of the assembler will insert debugging pseudo-instructions in the preprocessed file. Therefore, you do not have to describe these pseudo-instructions in creating source files.

4.7.15 Comment Adding Function

The preprocessing pseudo-instructions that begin with "#" are all expanded to codes that can be assembled, and delivered in the preprocessed file. Even after that, those instructions are rewritten with comments beginning with a semicolon (;), so that the original instructions can be identified. However, note that the replacements of Define names will not subsist as comments.

The comment is added to the first line following the expansion. In case the original statement is accompanied by a comment, that comment is also added.
A macro definition should have a semicolon (;) placed at top of the line.

Example:

- Before expansion

```
#define    Areg       %a

#macro     ADDX2Y     VALUE
           ld         Areg, VALUE
           add        Areg, [%x]
           ld         [%y], Areg
#endm

           ADDX2Y     10h   ; MX + 10h -> MY
```

- After expansion (no debugging information)

```
;#define   Areg       %a

;#macro    ADDX2Y     VALUE
;          ld         Areg, VALUE
;          add        Areg, [%x]
;          ld         [%y], Areg
;#endm

           ;ADDX2Y    10h   ; MX + 10h -> MY
           ld         %a, 0x10
           add        %a, [%x]
           ld         [%y], %a
```

4.7.16 Priority of Pseudo-Instructions

Some remarks concerning the priority among the preprocessing pseudo-instructions will be given below:

1. The conditional assembly instructions (***#ifdef***, ***#ifndef***) have the first priority. Nesting cannot be made of those instructions.
2. Define instruction (***#define***), include instruction (***#include***), or macro instruction (***#macro***) can be described within a conditional assembly instruction.
3. Define instruction (***#define***), include instruction (***#include***), and macro instruction (***#macro***) cannot be described within a macro definition.
4. Define name definitions are expanded with priority over macro definitions.

4.8 Relocatable List File

The relocatable list file is an assembly source file that carries assembled results (offset addresses and object codes) added to the first half of each line. It is delivered only when the start-up option (-l) is specified.
Its file format is a text file, and the file name, <File name>.lst. (The <File name> is the same as that of the input source file.)
The format of each line of the assembly list file is as follows:

Line No.: Address Code Source statement

Example

```
Assembler 63 ver x.xx Relocatable List File MAIN.LST Sat Nov 07 12:40:41 1998

     1:                  ; main.s
     2:                  ; AS63 test program (main routine)
     3:                  ;
                                      :
    25:
    26:                          .org    0x110
    27:                  BOOT:
    28:   0110   0900            ldb     %ba,SP1_INIT_ADDR
    29:   0111   1fc4            ldb     %sp1,%ba              ; set SP1
    30:   0112   0900            ldb     %ba,SP2_INIT_ADDR
    31:   0113   1fc6            ldb     %sp2,%ba              ; set SP2
    32:   0114   0200            calr    INIT_RAM_BLK1         ; initialize RAM block 1
    33:                  LOOP:
    34:   0115   0200            calr    INC_RAM_BLK1          ; increment RAM block 1
    35:   0116   0000            jr      LOOP                  ; infinity loop
    36:
    37:
    38:                  ;***** RAM block *****
    39:
    40:                          .org 0x0
    41:                          .bss
    42:   0000   00              .comm RAM_BLK0, 4
    43:   0004   00              .comm RAM_BLK1, 4
```

Content of line No.

The source line number from top of the file will be delivered.

Content of address

In the case of an absolute section, an absolute address will be delivered in hexadecimal number.
In the case of a relocatable section, a relative address will be delivered in hexadecimal number from top of the file.

Content of code

CODE section: The instruction (machine language) codes are delivered in hexadecimal numbers. One address corresponds with one instruction. The assembler sets the operand (immediate data) of the code that refers to unresolved address to 0. The immediate data will be decided by the linker.

DATA section: The 4-bit data defined by the ***.word*** pseudo-instruction are delivered. One address corresponds with one data.

BSS section: Irrespective of the size of the secured area, 00 is always delivered here.
Only the address defined for a symbol (top address of the secured area) is delivered as the address of the BSS section.

4.9 Sample Executions

Command line

```
C:\E0C63\bin\as63 -g -e -l main.s
```

Assembly source file

```
; main.s
; AS63 test program (main routine)
;

;***** INITIAL SP1 & SP2 ADDRESS DEFINITION *****

#ifdef SMALL_RAM
       .set SP1_INIT_ADDR 0xb           ;SP1 init addr = 0x2c
#else
       .set SP1_INIT_ADDR 0x4b          ;SP1 init addr = 0x12c
#endif

       .set SP2_INIT_ADDR 0x1f          ;SP2 init addr = 0x1f


;***** NMI & BOOT, LOOP *****

       .global INIT_RAM_BLK1            ; subroutine in sub.s
       .global INC_RAM_BLK1             ; subroutine in sub.s

       .org    0x100
NMI:
       calr    INIT_RAM_BLK1            ; initialize RAM block 1
       reti                             ; in NMI(watchdog timer)

       .org    0x110
BOOT:
       ldb     %ba,SP1_INIT_ADDR
       ldb     %sp1,%ba                 ; set SP1
       ldb     %ba,SP2_INIT_ADDR
       ldb     %sp2,%ba                 ; set SP2
       calr    INIT_RAM_BLK1            ; initialize RAM block 1
LOOP:
       calr    INC_RAM_BLK1             ; increment RAM block 1
       jr      LOOP                     ; infinity loop


;***** RAM block *****

       .org 0x0
       .bss
       .comm RAM_BLK0, 4
       .comm RAM_BLK1, 4
```

Preprocessed file

```
.stabs "C:\E0C63\Test\main.s", FileName
; main.s
; AS63 test program (main routine)
;

;***** INITIAL SP1 & SP2 ADDRESS DEFINITION *****

;#ifdef SMALL_RAM
;      .set SP1_INIT_ADDR 0xb           ;SP1 init addr = 0x2c
;#else
       .set SP1_INIT_ADDR 0x4b          ;SP1 init addr = 0x12c
;#endif

       .set SP2_INIT_ADDR 0x1f          ;SP2 init addr = 0x1f


;***** NMI & BOOT, LOOP *****

       .global INIT_RAM_BLK1            ; subroutine in sub.s
       .global INC_RAM_BLK1             ; subroutine in sub.s

       .org    0x100
NMI:
.stabn 23, LineInfo
       calr    INIT_RAM_BLK1            ; initialize RAM block 1
.stabn 24, LineInfo
       reti                             ; in NMI(watchdog timer)

       .org    0x110
BOOT:
.stabn 28, LineInfo
       ldb     %ba,SP1_INIT_ADDR
.stabn 29, LineInfo
       ldb     %sp1,%ba                 ; set SP1
.stabn 30, LineInfo
       ldb     %ba,SP2_INIT_ADDR
.stabn 31, LineInfo
       ldb     %sp2,%ba                 ; set SP2
.stabn 32, LineInfo
       calr    INIT_RAM_BLK1            ; initialize RAM block 1
LOOP:
.stabn 34, LineInfo
       calr    INC_RAM_BLK1             ; increment RAM block 1
.stabn 35, LineInfo
       jr      LOOP                     ; infinity loop


;***** RAM block *****

       .org    0x0
       .bss
       .comm    RAM_BLK0, 4
       .comm    RAM_BLK1, 4
.stabn 0, FileEnd
```

Assembly list file

```
Assembler 63 ver x.xx Relocatable List File MAIN.LST Sat Nov 07 12:40:41 1998

        1:              ; main.s
        2:              ; ASM63 test program (main routine)
        3:              ;
        4:
        5:              ;***** INITIAL SP1 & SP2 ADDRESS DEFINITION *****
        6:
        7:              #ifdef SMALL_RAM
        8:                      .set SP1_INIT_ADDR 0xb          ;SP1 init addr = 0x2c
        9:              #else
       10:                      .set SP1_INIT_ADDR 0x4b         ;SP1 init addr = 0x12c
       11:              #endif
       12:
       13:                      .set SP2_INIT_ADDR 0x1f         ;SP2 init addr = 0x1f
       14:
       15:
       16:              ;***** NMI & BOOT, LOOP *****
       17:
       18:                      .global INIT_RAM_BLK1           ; subroutine in sub.s
       19:                      .global INC_RAM_BLK1            ; subroutine in sub.s
       20:
       21:                      .org    0x100
       22:              NMI:
       23:   0100   0200        calr    INIT_RAM_BLK1           ; initialize RAM block 1
       24:   0101   1ff9        reti                            ; in NMI(watchdog timer)
       25:
       26:                      .org    0x110
       27:              BOOT:
       28:   0110   0900        ldb     %ba,SP1_INIT_ADDR
       29:   0111   1fc4        ldb     %sp1,%ba                ; set SP1
       30:   0112   0900        ldb     %ba,SP2_INIT_ADDR
       31:   0113   1fc6        ldb     %sp2,%ba                ; set SP2
       32:   0114   0200        calr    INIT_RAM_BLK1           ; initialize RAM block 1
       33:              LOOP:
       34:   0115   0200        calr    INC_RAM_BLK1            ; increment RAM block 1
       35:   0116   0000        jr      LOOP                    ; infinity loop
       36:
       37:
       38:              ;***** RAM block *****
       39:
       40:                      .org 0x0
       41:                      .bss
       42:   0000   00          .comm RAM_BLK0, 4
       43:   0004   00          .comm RAM_BLK1, 4
```

Error file

```
Assembler 63 Ver x.xx Error log file MAIN.ERR Sat Nov 07 12:40:41 1998

Assembler 63 Ver x.xx
Copyright (C) SEIKO EPSON CORP. 1998

Created preprocessed source file MAIN.MS
Created relocatable list file MAIN.LST
Created error log file MAIN.ERR
Created relocatable object file MAIN.O

Assembly 0 error(s) 0 warning(s)
```

4.10 Error/Warning Messages

4.10.1 Errors

When an error occurs, no object file will be generated.
The assembler error messages are delivered/displayed in the following format:

<Source file name> (<Line number>) Error : <Error message>

Example: `TEST.S(431) Error: Illegal syntax`

∗ Some error messages are displayed without a line number.

The assembler error messages are given below:

Error message	Description
Address out of range	The specified address is out of range.
Cannot open <file kind> file <FILE NAME>	The specified file cannot be opened.
Cannot read <file kind> file <FILE NAME>	The specified file cannot be read.
Cannot write <file kind> file <FILE NAME>	Data cannot be written to the file.
Directory path length limit <directory path length limit> exceeded	The path name length has exceeded the limit.
Division by zero	The divisor in the expression is 0.
File name length limit <file name length limit> exceeded	The file name length has exceeded the limit.
Illegal macro label <label>	The internal branch label in macro definition is incorrect.
Illegal macro parameter <parameter>	The macro parameter is illegal.
Illegal syntax	The statement has a syntax error.
Line length limit <line length limit> exceeded	The number of characters in one line has exceeded the limit.
Macro parameter range <macro parameter range> exceeded	The number of macro parameters has exceeded the limit.
Memory mapping conflict	The address is already used.
Multiple statements on the same line	Two or more statements were described on one line.
Nesting level limit <nesting level limit> exceeded	Nesting of #include has exceeded the limit.
Number of macro labels limit <number of macro label limit> exceeded	The number of internal branch labels has exceeded the limit.
Out of memory	Cannot secure memory space.
Second definition of label <label>	The label is already defined.
Second definition of symbol <symbol>	The symbol is already defined.
Symbol name length limit <symbol name length limit> exceeded	The symbol name length has exceeded the limit.
Token length limit <token length limit> exceeded	The token length has exceeded the limit.
Unexpected character <name>	An invalid character has been used.
Unknown label <label>	Reference was made to an undefined label.
Unknown mnemonic <name>	A nonexistent instruction was used.
Unknown register <name>	A nonexistent register name was used.
Unknown symbol <name>	A reference to an undefined symbol was made.
Unknown symbol mask <name>	The symbol mask has a description error.
Unsupported directive <directive>	A nonexistent pseudo-instruction was used.

4.10.2 Warning

When a warning occurs, the assembler will keep on processing, and terminates the processing after displaying a warning message, unless an error is produced.
The warning message is delivered/displayed in the following formats:

<Source file name> (<Line number>) Warning : <Warning message>

Example: `TEST.S(41) : Warning : Expression out of range`

The warning messages are given below:

Warning message	Description
Expression out of range	The result of the expression is out of the effective range.
Invalid symbol mask	The symbol mask is not defined correctly.
Second definition of define symbol <symbol>	The symbol is already defined.
Section activation expected, use <.code/.bss>	There is no section definition.

4.11 Precautions

(1) Nesting of the ***#include*** pseudo instruction is limited to a maximum 10 levels. If this limit is surpassed, an error will result.

(2) A maximum of 64 internal branch labels can be specified per macro and maximum 9999 internal branch labels can be expanded within one source file. If these limits are exceeded, an error will result.

(3) Other limitations such as the number of sections depend on the free memory space.

CHAPTER 5 LINKER

This chapter describes the functions of the linker, lk63.

5.1 Functions

The linker lk63 is a software that generates executable object files. It provides the following functions:

- Puts together multiple object modules to create one executable object file.
- Resolves external reference from one module to another.
- Relocates relative addresses to absolute addresses.
- Delivers debugging information, such as line numbers and symbol information, in the object file created after linking.
- Capable of outputting a link map file, symbol file, absolute list file and a cross reference file.
- Automatic page correction function (insertion/removal/correction of the "ldb %ext, imm8" branch extension instruction) for branch instructions.

5.2 Input/Output Files

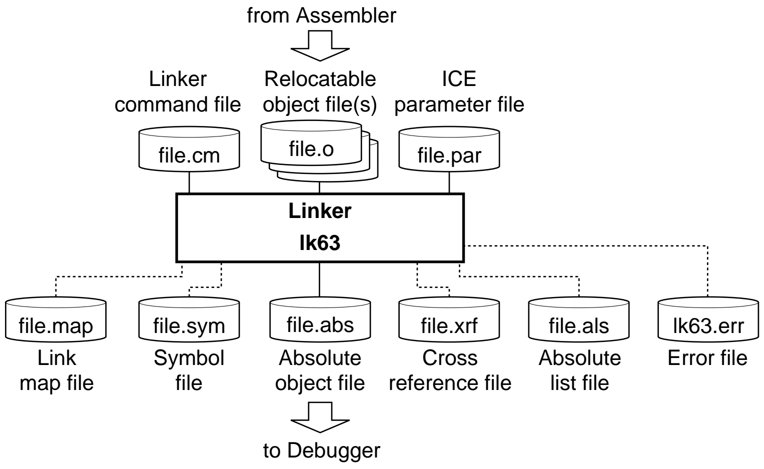


Fig. 5.2.1 Flow chart

5.2.1 Input Files

Relocatable object file ∗ **This file must always be specified in either a command line or a link command file.**

File format: Binary file in IEEE-695 format
File name: <File name>.o (A path can also be specified.)
Description: Object file of individual modules created by the assembler.

Linker command file

File format: Text file
File name: <File name>.cm (A path can also be specified.)
Description: File to specify the linker options. This makes it possible to reduce typing in a command line. This file is dispensable if all start-up options can be input in a command line.

ICE parameter file ∗ **This file must always be specified in either a command line or a link command file.**

File format: Binary file
File name: <File name>.par (A path can also be specified.)
Description: File to specify the memory mapping and unsupported instruction information of each E0C63 Family model. This file is supplied in the development tools for each model and commonly used with the debugger and HEX converter.

5.2.2 Output Files

An output file name can be specified in the command line or command file using the -o start-up option. If no output file name is specified, the same name as that of the relocatable object file to be linked first is used.

Absolute object file

File format: Binary file in IEEE-695 format
File name: <File name>.abs
Output destination: Current directory
Description: Object file in executable format that can be input to the debugger. All the modules comprising one program are linked together in the file, and the absolute addresses that all the codes will map are determined. It also contains the necessary debugging information in IEEE-695 format.

Link map file

File format: Text file
File name: <File name>.map
Output destination: Current directory
Description: Mapping information file showing from which address of a section each input file was mapped. This file is output when the -m start-up option is specified.

Symbol file

File format: Text file
File name: <File name>.sym
Output destination: Current directory
Description: Symbols defined in all the modules and their address information are delivered to this file. This file is delivered when the -s start-up option is specified.

Cross reference file

File format: Text file
File name: <File name>.xrf
Output destination: Current directory
Description: Labels defined in all the modules and their defined and referred addresses are delivered in this file. This file is delivered when the -x start-up option is specified.

Absolute list file

File format: Text file
File name: <File name>.als
Output destination: Current directory
Description: File delivered when the -l start-up option is specified. The file contents are similar to the relocatable list file output by the assembler except that the location addresses are absolute and takes the form of an integrated single file.

Error file

File format: Text file
File name: lk63.err
Output destination: Current directory
Description: The file is created if the -e start-up option is specified. It records the information which the linker outputs to the Standard Output (stdout), such as error messages. The file name is "lk63.err" by default, but it can be changed using the -o start-up option.

5.3 Starting Method

General form of command line

lk63 ^ [Options] ^ [<Relocatable object files>] ^ [<Linker command file>] ^ <ICE parameter file>

^ denotes a space.
[ ] indicates the possibility to omit.
The order of options and file names can be arbitrary.

File names

Files are identified with their extensions. Therefore, an appropriate extension should be included in each file name. However, the extension ".o" of the relocatable object file can be omitted.

Relocatable object files: <File name.o>
Linker command file: <File name.cm>
ICE parameter file: <File name.par>

When using a linker command file, options, relocatable object file names, an ICE parameter file name and an output file name can be described in the linker command file. If all the items to be specified are entered in a command line, the linker command file is not necessary.

When linking multiple relocatable object files from a command line, one or more spaces should be placed between the file names.

For the output file name, specify an absolute object file name (.abs). The file name will be used for other output files. If no absolute object file name is specified, the same name as that of the relocatable object file to be linked first is used as the output file name.

The ICE parameter file cannot be omitted.

A long file name supported in Windows and a path name can be specified. When including spaces in the file name, enclose the file name with double quotation marks (").

Options

The linker comes provided with the following options:

-d

Function: Disable full branch optimization
Explanation: Disables automatic insertion/deletion/correction of the extension codes (ldb %ext, imm8) for branch instructions (jumps and calls).
Default: If this option is not specified, the branch optimization function will be enabled.

-dr

Function: Disable removal branch optimization
Explanation: Disables extension code deletion only among full branch optimization (insertion/deletion/correction). This will be needed when at least the existing extension codes should not be removed.
Default: If this option is not specified, unnecessary extension codes will be removed when the full branch optimization function is specified.

-e

Function: Output of error file
Explanation: Creates an .err file which contains the information that the linker outputs to the Standard Output (stdout), such as error messages.
Default: If this option is not specified, no error file will be created.

-g

Function: Addition of debugging information

Explanation:
- Creates an absolute object file containing debugging information.
- Always specify this function when you perform source display or use the symbolic debugging facility of the debugger.

Default: If this option is not specified, no debugging information will be added to the absolute object file.

-l

Function: Output of absolute list file

Explanation: Outputs an absolute list file.

Default: If this option is not specified, no absolute list file will be output.

-m

Function: Output of link map file

Explanation: Outputs a link map file.

Default: If this option is not specified, no link map file will be output.

-o <file name>

Function: Specification of output path/file name

Explanation: Specifies an output path/file name without extension or with an extension ".abs". If no extension is specified, ".abs" will be supplemented at the end of the specified output path/file name.

Default: The 1st input file name is used for the output file names.

-s

Function: Output of symbol file

Explanation: Outputs a symbol file.

Default: If this option is not specified, no symbol file will be output.

-x

Function: Output of cross reference file

Explanation: Outputs a cross reference file.

Default: If this option is not specified, no cross reference file will be output.

-code <address>

Function: Set up of a relocatable CODE section start address

Explanation:
- Sets the absolute start address of a relocatable CODE section. Absolute sections remain unaffected.
- CODE sections are mapped from this address, unless otherwise specified.
- One or more spaces or tabs are necessary between -code and <address>.
- The address should be described in hexadecimal format (0xnnnn).

Default: If this option is not specified, the CODE section will begin from the code ROM physical start address specified with the ICE parameter file.

Sample description: `-code 0x100`

-data <address>

Function: Set up of a relocatable DATA section start address

Explanation:
- Sets the absolute start address of a relocatable DATA section. Absolute sections remain unaffected.
- DATA sections are mapped from this address, unless otherwise specified.
- One or more spaces or tabs are necessary between -data and <address>.
- The address should be described in hexadecimal format (0xnnnn).

Default: If this option is not specified, the DATA section will begin from the data ROM physical start address specified with the ICE parameter file.

Sample description: `-data 0x8000`

-bss <address>

Function: Set up of a relocatable BSS section start address

Explanation:
- Sets the absolute start address of a relocatable BSS section. Absolute sections remain unaffected.
- BSS sections are mapped from this address, unless otherwise specified.
- One or more spaces or tabs are necessary between -bss and <address>.
- The address should be described in hexadecimal format (0xnnnn).

Default: If this option is not specified, the BSS section will begin from the RAM physical start address specified with the ICE parameter file.

Sample description: `-bss 0x000`

-rcode <file name>=<address>

Function: Set up of the file-specific CODE section start address

Explanation:
- Sets the absolute address to map the CODE section of the specified module. This command serves to specify a module having a code to be fixed at a specific address, such as the interrupt vector. Absolute sections in the specified file remain unaffected.
- One or more spaces or tabs are necessary between -rcode and <file name>.
- The address should be described in hexadecimal format (0xnnnn).

Default: If this option is not specified, the CODE section of each module is mapped continuously from the address that was set by the -code option.

Sample description: `-rcode test1.o = 0x0110`

-rdata <file name>=<address>

Function: Set up of the file-specific DATA section start address

Explanation:
- Sets the absolute address to map the DATA section of the specified module. This command serves to specify a module having data to be fixed at a specific address of the data ROM. Absolute sections in the specified file remain unaffected.
- One or more spaces or tabs are necessary between -rdata and <file name>.
- The address should be described in hexadecimal format (0xnnnn).

Default: If this option is not specified, the DATA section of each module is mapped continuously from the address that was set by the -data option.

Sample description: `-rdata test1.o = 0x8100`

-rbss <file name>=<address>

Function: Set up of the file-specific BSS section start address

Explanation:
- Sets the absolute address to map the BSS section of the specified module. This command serves to specify a module having a symbol to be fixed at a specific address of the RAM. Absolute sections in the specified file remain unaffected.
- One or more spaces or tabs are necessary between -rbss and <file name>.
- The address should be described in hexadecimal format (0xnnnn).

Default: If this option is not specified, the BSS section of each module is mapped continuously from the address that was set by the -bss command.

Sample description: `-rbss test1.o = 0x100`

-defsym <symbol name>=<address>

Function: Specification of a global symbol address

Explanation:
- The absolute address of a global symbol is given for the referencing side.
- The symbols to be specified with this option should not be defined in the source as an actual address label that can be referred to.
- One or more spaces or tabs are necessary between -defsym and <symbol name>.

Sample description: `-defsym BOOT = 0x100`

When inputting an option in the command line, one or more spaces are necessary before and after the option.

Examples:
```
c:\e0c63\lk63 -defsym INIT=0x200 test.cm par63xxx.par
c:\e0c63\lk63 -g -e -s -m test1.o test2.o -o test.abs par63xxx.par
```

5.4 Messages

The linker delivers all its messages to the Standard Output (stdout).

Start-up message

The linker outputs only the following message when it starts up.

```
Linker 63 Ver x.xx
Copyright (C) SEIKO EPSON CORP. 199x
```

End message

The linker outputs the following messages to indicate which files has been created when it ends normally.

```
Created absolute object file <FILENAME.ABS>
Created absolute list file <FILENAME.ALS>
Created map file <FILENAME.MAP>
Created symbol file <FILENAME.SYM>
Created cross reference file <FILENAME.XRF>
Created error log file <FILENAME.ERR>

Link 0 error(s) 0 warning(s)
```

Usage output

If no file name was specified or an option was not specified correctly, the linker ends after delivering the following message concerning the usage:

```
Usage: lk63 [options] <file names>
Options: -d                            Disable full branch optimization
         -dr                           Disable removal branch optimization
         -e                            Output error log file (.ERR)
         -g                            Add source debug information
         -l                            Output absolute list file (.ALS)
         -m                            Output map file (.MAP)
         -o <file name>                Specify output file name
         -s                            Output symbol file (.SYM)
         -x                            Output cross reference file (.XRF)
         -code <address>               Specify CODE start address
         -data <address>               Specify DATA start address
         -bss <address>                Specify BSS start address
         -rcode <file name>=<address>  Specify CODE start address of the file
         -rdata <file name>=<address>  Specify DATA start address of the file
         -rbss <file name>=<address>   Specify BSS  start address of the file
         -defsym <symbol>=<address>    Define symbol address
File names: Relocatable object file (.O)
            Command parameter file (.CM)
            ICE parameter file (.PAR)
```

When error/warning occurs

If an error takes place, an error message will appear before the end message shows up.

Example:

```
Error: Cannot create absolute list file TEST.ABS
Link 1 error(s) 0 warning(s)
```

In the case of an error, the linker ends without creating an output file.

If a warning is issued, a warning message will appear before the end message shows up.

Example:

```
Warning: No debug information in TEST.O
Link 0 error(s) 1 warning(s)
```

In the case of a warning, the linker ends after creating an output file, but the result cannot be guaranteed.

For details on errors and warnings, refer to Section 5.12, "Error/Warning Messages".

5.5 Linker Command File

To simplify the keystroke in the command line at the time of start up, execute the link processing through the linker by inputting a linker command file (.cm) that holds the necessary specifications (any options and file names) described.

Sample linker command file

```
-e                      ; Generate error file
-g                      ; Add debug information
-code 0x0100            ; Fix CODE section start address
-rcode test2.o = 0x0110 ; Fix CODE section start position of test2.o
-data 0x8000            ; Fix DATA section start address
-bss 0x00e0             ; Fix BSS section start address

-defsym IO = 0xFF00     ; Set global symbol

-o test.abs             ; Specify output file name
test1.o                 ; Specify input file 1
test2.o                 ; Specify input file 2
```

Create the linker command file with the following rules:

File format

The linker command file is a general text format as shown above.
".cm" should be used for the file name extension.

Option description

All options should begin with a hyphen (-). Each individual option needs to be delineated with more than one space, tab, or line feed. For better visibility, it is recommended to describe each option in a separate line.

Notes: • *A numeric value to specify an address should be described in the hexadecimal format (0xnnnn). Decimal and binary notations will not be accepted.*

• *When an option that is only permitted in single setting is specified in a duplicated manner, the last entered option will be effective.*

Example:
```
-code 0x0000
-code 0x0100
```
... -code 0x0100 is effective.

Input file specification

Describe the relocatable object file names at the end of the link command file. The mapping by linking takes place in described order, unless otherwise specified.
The extension (.o) of the relocatable object files can be omitted.

Comment

A comment can be described in the linker command file.
As in the source file, the character string from a semicolon (;) to the end of the line is regarded as a comment.

Blank line

A blank line carrying only blank characters and a line feed will be ignored. It need not be converted to a comment using a semicolon.

5.6 Link Map File

The link map file serves to refer to the mapping information for the modules of each section. It is output if the -m option is specified.
The file format is a text file, and its file name is "<File name>.map". (<File name> is the same as that of the output object file.)

Sample link map file

```
Linker 63 ver x.xx Link map file TEST.MAP Sat Nov 07 12:40:41 1998

CODE section map of TEST.ABS
Index  Start   End     Size    Opt  Type File         SecNbr
    0: 0x0000  0x000d  0x000e   +0  Rel  SUB.S        1
    1: 0x000e  0x00ff  0x00f2  ---  ---  ------------ ---
    2: 0x0100  0x0102  0x0003   +1  Abs  MAIN.S       1
    3: 0x0103  0x010f  0x000d  ---  ---  ------------ ---
    4: 0x0110  0x0118  0x0009   +2  Abs  MAIN.S       2
    5: 0x0119  0x1fff  0x1ee7  ---  ---  ------------ ---
Total: 0x1a occupied, 0x1fe6 blank

BSS section map of TEST.ABS
Index  Start   End     Size    Type File         SecNbr
    0: 0x0000  0x0007  0x0008  Rel  MAIN.S       3
    1: 0x0008  0xf2bf  ------  ---  ------------ ---
    2: 0xf800  0xf8ff  ------  ---  ------------ ---
    3: 0xff00  0xffff  ------  ---  ------------ ---
Total: 0x8 occupied, 0xf4b8 blank
```

Contents of link map file

Index Indicates the index number of the section.

Start Indicates the start address of the section.

End Indicates the end address of the section.

Size Indicates the size of the section.

Opt Indicates the number of extension codes that are inserted or removed.

Type Indicates the section type: Rel = relocatable section and Abs = absolute section.

File Indicates the file names of the linked module.

SecNbr Indicates the section number.

Total Indicates the total map size and the unused area size.

"---" in the Size, Opt, Type, File and SecNbr columns indicate that no section is allocated.

5.7 Symbol File

The symbol file serves to refer to the symbols defined in all the modules and their address information. It is delivered if the -s start-up option is specified.
The file format is a text file, and its file name is "<File name>.sym". (<File name> is the same as that of the output object file.)

Sample symbol file

```
Linker 63 ver x.xx Symbol file TEST.SYM Sat Nov 07 12:40:41 1998

CODE section labels of TEST.ABS
Address Type   File          Symbol
0x0110  Local  "MAIN.O" .... BOOT
0x0007  Global "SUB.O" ..... INC_RAM_BLK1
0x0000  Global "SUB.O" ..... INIT_RAM_BLK1
0x0116  Local  "MAIN.O" .... LOOP
0x0100  Local  "MAIN.O" .... NMI

BSS section labels of TEST.ABS
Address Type   File          Symbol
0x0000  Global "MAIN.O" .... RAM_BLK0
0x0004  Global "MAIN.O" .... RAM_BLK1
```

Contents of symbol file

Symbol Indicates all the defined symbols in alphabetical order.

Address Indicates the absolute address defined for the symbol.

Type Indicates the scope of the symbol: Global or Local.

File Indicates the object file in which the symbol has been defined.

5.8 Absolute List File

The absolute list file is an assembly source file that carries the absolute addresses and object codes added to the first half of each line. It is delivered only when the -l option is specified. Its file format is a text file, and the file name is <file name>.als. (The <file name> is the same as that of the output object file.) While a relocatable list file can be made for each assembly source file, the absolute list file is made as a single file integrating all the linked objects and their according sources.

Sample absolute list file

```
Linker 63 ver x.xx Absolute list file TEST.ALS Sat Nov 07 12:40:41 1998

     1:                     ; sub.s
     2:                     ; AS63 test program (subroutine)
     3:
     4:                             .global RAM_BLK1
     5:
     6:                     ;***** RAM block 1 initialize *****
     7:
     8:                             .global INIT_RAM_BLK1
     9:                     INIT_RAM_BLK1:
    10:  0000   0800                ldb     %ext,RAM_BLK1@h
    11:  0001   0a04                ldb     %xl,RAM_BLK1@l    ;set RAM_BLK1 address to x
    12:  0002   1e90                ld      [%x]+,0x0
     :     :     :                         :
    55:                             .org    0x110
    56:                     BOOT:
    57:  0110   094b                ldb     %ba,SP1_INIT_ADDR
    58:  0111   1fc4                ldb     %sp1,%ba          ; set SP1
    59:  0112   091f                ldb     %ba,SP2_INIT_ADDR
    60:  0113   1fc6                ldb     %sp2,%ba          ; set SP2
    61:  0114   08fe   (+)          ldb  ext,fe
    62:  0115   02ea                calr    INIT_RAM_BLK1     ; initialize RAM block 1
    63:                     LOOP:
    64:  0116   08fe   (+)          ldb  ext,fe
    65:  0117   02ef                calr    INC_RAM_BLK1      ; increment RAM block 1
    66:  0118   00fd                jr      LOOP              ; infinity loop
     :     :     :                         :
```

Contents of absolute list file

The format of each line of the absolute list file is as follows:

Line No.   Absolute address   Code   Source statement

Line No. Indicates the line number from the top of the file.
Address Indicates the absolute address after the instruction is allocated.
Code Indicates the object code.
Source The contents of the assembly source file are delivered.

Results of branch optimization (extension code insertion/deletion/correction)

As the result of branch optimization, extension codes (ldb %ext, imm8) may be coded without accordance to the source part. To show the result of such code optimizations clearly, the following description will be made on an absolute list file.

When an extension code is inserted:

"(+)" is placed to the right of the code part. There is no original source for the code but the disassembled "ldb %exe, imm8" is delivered at the source part.

When an extension code is deleted:

"(-)" is placed to the left of the original source part. The original statement appears at the source part in the list file but no code is delivered.

When the operand of an extension code is corrected:

"(*)" is placed to the left of the source statement.

Instructions preprocessed in the assembler

The instructions expanded in the assembler (macros and include sources) are listed with a "+".

5.9 Cross Reference File

The cross reference file enumerates all the address labels with their absolute addresses and all the addresses where the address labels are referred to. It is delivered only when the -x option is specified. Its file format is a text file, and the file name is <file name>.xrf. (The <file name> is the same as that of the output object file.)

Sample cross reference file

```
Linker 63 ver x.xx Cross reference file TEST.XRF Sat Nov 07 12:40:41 1998

Label "INIT_RAM_BLK1" at 0x0000  SUB.O  CODE, Global
   0x0101  MAIN.O  CODE
   0x0115  MAIN.O  CODE

Label "RAM_BLK0" at 0x000  MAIN.O  BSS, Global
   0x0101  MAIN.O  CODE
   0x0115  MAIN.O  CODE

Label "RAM_BLK1" at 0x004  MAIN.O  BSS, Global
   0x0000  SUB.O  CODE
   0x0001  SUB.O  CODE
   0x0007  SUB.O  CODE
   0x0008  SUB.O  CODE

Label "INC_RAM_BLK1" at 0x0007  "SUB.O"  CODE, Global
   0x0117  MAIN.O  CODE

Label "NMI" at 0x0100  MAIN.O  CODE, Local

Label "BOOT" at 0x0110  MAIN.O  CODE, Local

Label "LOOP" at 0x0116  MAIN.O  CODE, Local
   0x0118  MAIN.O  CODE
```

Contents of cross reference file

The format of each label information is as follows:

```
Label information
     Address   File name  Type
        :          :
```

Label information
Indicates the following information:
- Label name
- Defined address
- Object file in which the label is defined.
- Section type
- Scope

Address — Indicates the address where the label is referred.

File — Indicates the object file in which the label is referred.

Type — Indicates the type of section that contains the address where the label is referred.

5.10 Linking

Linking rules

The linking process takes place in conformity with the following rules:

- Absolute sections are mapped ahead of relocatable sections, according to the absolute addresses which were defined at the time of assembling. If an absolute section exceeds the available memory area, an error will occur.
- The relocatable sections in the file of which the section start address was specified with an option (-rcode, -rdata, -rbss) are mapped from the specified address. Other relocatable sections are mapped from top of the relocatable CODE/DATA/BSS section.
- Basically, the relocatable sections except those that are specified with the -rcode, -rdata or -rbss option are arranged successively in the order of processing. However, if a relocatable section cannot be mapped subsequent to the previous mapped section, for instance, there is unused area indicated by the ICE parameter file or an already mapped absolute section, the linker searches another area to map the section. If there is no available area, an error will occur. A section is not divided into two or more blocks when it is mapped.
  After that, another section may be mapped in the vacant area if it is possible to map there.

Restrictions on linking

Note that all sections may not be mapped depending on each section size or address specifications even if the relocatable object size is within the available memory size.

Example of linking

A sample case where two relocatable object files, "test1.o" and "test2.o", are linked together under the following condition is described further below.

Memory configuration of the model

Code ROM: 0x0000 to 0x1fff
Data ROM: 0x8000 to 0x87ff
RAM: 0x0000 to 0x07ff
Display, I/O memory: 0xf000 to 0xffff

Relocatable object files

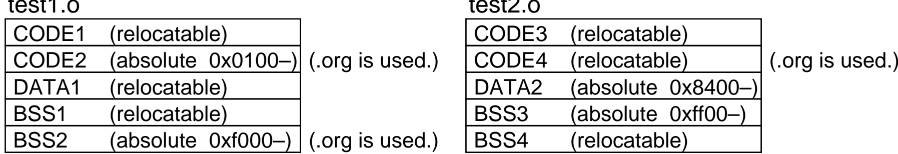


Fig. 5.10.1 Structure of sample relocatable files

Sample linker command file

```
-code 0x0000            ; Relocatable CODE section start address
-rcode test2.o = 0x0110 ; CODE section start address of test2.o
-data 0x8000            ; Relocatable DATA section start address
-bss 0x0000             ; Relocatable BSS section start address
-rbss test2.o = 0x0400  ; BSS section start address of test2.o
-o test.abs             ; Output file name
test1.o                 ; Input file 1
test2.o                 ; Input file 2
```

When linking is executed with the commands defined above, the linker maps the sections of each module in the manner graphically presented in Figure 5.10.2.

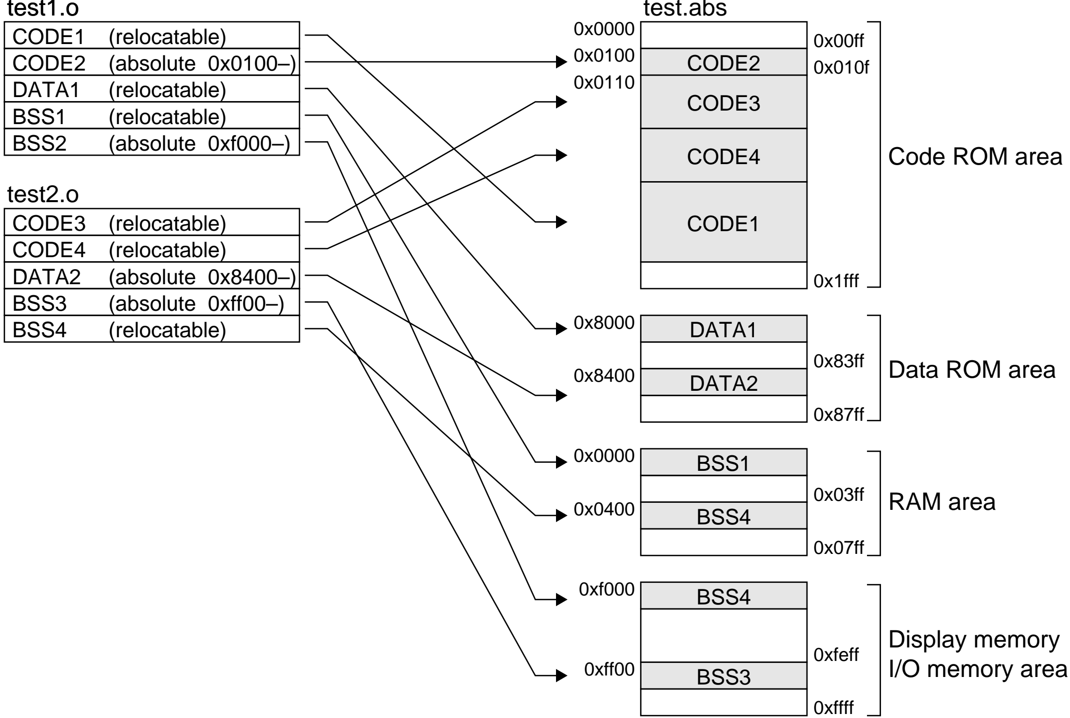


Fig. 5.10.2 Example of linking

The absolute sections CODE2, BSS2, DATA2 and BSS3 are mapped to the location specified in the source files.

The start addresses of the CODE and BSS relocatable sections in "test2.o" is specified by the -rcode and -rbss options, so CODE3 is mapped from address 0x0110 and CODE 4 follows CODE3. BSS4 is mapped from address 0x0400.

Since the start addresses of the relocatable CODE, DATA and BSS sections in "test1.o" have not been specified, they are mapped from the relocatable section start addresses specified by the -code, -data and -bss options. First the linker will try to map CODE1 from address 0x0000 to address 0x00ff. If CODE 1 is smaller than 0x100 words, CODE1 can be mapped from address 0x0000. In this example, CODE1 is mapped behind CODE4 because CODE1 is larger than 0x100 words.
DATA1 is mapped from address 0x8000 and BSS1 is mapped from address 0x0000.
A section cannot be overlapped to other sections, therefore an error will occur if there is no free area larger than the section size. For example, an error will occur if CODE2 is larger than 0x10 words.

5.11 Branch Optimization Function

The PC relative branch instructions (jr, jrc, jrnc, jrz, jrnz and calr) need an address extension instruction (ldb %ext, imm8) when the relative distance to the destination address exceeds the -127 to 128 range. Since the location of relocatable sections is not decided until the linking process is completed, the linker has a function that automatically inserts, removes or corrects the extension codes. This makes it possible to omit the address extension instruction in the source. However, this function is valid only for the branch instructions that use a label to specify the destination address.
This function can be disabled by specifying the -d option. The -dr option can also be specified to disable only the extension code deletion function (in the case of the -d option is not specified).

The linker checks the distance from a PC relative branch instruction code to the branch destination label, and inserts, removes or corrects the extension codes according to the check results.

(1) When the branch destination is located within the -127 to +128 range from the branch instruction:
If the branch instructon code does not have an extension code, no extension code is inserted.
If the branch instruction has an extension code, it is removed (if the -dr option is specified, existing expansion code will not be removed).
Examples:

```
jr     LABEL               →    jr     LABEL

ldb    %ext,LABEL@rh
calr     LABEL@rl          →    calr   LABEL@rl
```

(2) When the branch destination is located outside the -127 to +128 range from the branch instruction:
If the branch instructon code does not have an extension code, an appropriate extension code is inserted.
If the branch instruction has an illegal extension code, it is replaced with a correct extension code.
Examples:

```
jr     LABEL               →    ldb    %ext,LABEL@rh
                                jr     LABEL@rl

ldb    %ext,LABEL1@rh      →    ldb    %ext,LABEL2@rh
calr   LABEL2@rl                calr   LABEL2@rl
```

5.12 Error/Warning Messages

5.12.1 Errors

When an error occurs, the linker will immediately terminate the processing after displaying an error message. No object file will be output. Other files will be delivered only in the part which was processed prior to the occurrence of the error.
The error messages are given below.

Error message	Description
Branch destination too far from <address>	The branch destination address is out of range.
CALZ for non zero page at <address>	The specified address is out of the range (0x0000–0x00ff).
Cannot create absolute object file <FILE NAME>	The absolute object file cannot be created.
Cannot open <file kind> file <FILE NAME>	The file cannot be opened.
Cannot read <file kind> file <FILE NAME>	The file cannot be read.
Cannot relocate <section kind> section of <FILE NAME>	The relocatable section cannot be allocated.
Cannot write <file kind> file <FILE NAME>	Data cannot be written to the file.
Illegal address range <address> for a code at <address>	The address specified by TST/SET/CLR is out of the range (0x0000–0x003f or 0xffC0–0xffff).
Illegal file name <FILE NAME>	The file name is incorrect.
Illegal file name <FILE NAME> specified with option <option>	The file name specified with the option is incorrect.
Illegal ICE parameter at line <line number> of <FILE NAME>	The ICE parameter file contains an illegal parameter setting.
Illegal object <FILE NAME>	The input file is not an object file in IEEE-695 format.
Illegal option <option>	An illegal option is specified.
No address specified with option <option>	Address is not specified with the option.
No code to locate	There is no valid code for mapping.
No ICE parameter file specified	ICE parameter file is not specified.
No name and address specified with option <option>	Name and address are not specified with the option.
No object file specified	Object files to be linked are not specified.
Out of memory	Cannot secure memory space.
<section kind> section <address>-<address> overlaps with <section kind> section <address>-<address>	The address range of the section overlaps with another section's address range.
<section kind> section <address>-<address> overlaps with the unavailable memory	The address range of the section overlaps with the unavailable memory.
Unresolved external <label> in <FILE NAME>	Reference was made to an undefined symbol.
Unusable instruction code <instruction code> in <FILE NAME>	The object contains an instruction invalid for the model.

5.12.2 Warning

Even when a warning appears, the linker continues with the processing. It completes the processing after displaying a warning message, unless, in addition, an error takes place. The output files will all be delivered, but the operation of the program cannot be guaranteed.
The warning messages and their contents are given below.

Warning message	Description
Cannot create <file kind> file <FILE NAME>	The file cannot be created.
Cannot open <file kind> file <FILE NAME>	The file cannot be opened.
No debug information in <FILE NAME>	Debugging information is not included in the file.
No symbols found	Symbols cannot be found.
Second definition of label <label> in <FILE NAME>	The label has already been defined.
Second ICE parameter file <FILE NAME> ignored	Two or more ICE parameter files are specified.

5.13 Precautions

(1) Upper limits, such as a maximum section count and the number of objects to be linked, depend on the free memory space.

(2) To load the absolute object file created by the linker to the debugger, the same ICE parameter file must be specified when the debugger is invoked.

CHAPTER 6 HEX CONVERTER

This chapter describes the functions of hex converter, hx63.

6.1 Functions

The hex converter hx63 converts an absolute object file in IEEE-695 format output from the linker into a hex file in Motorola-S format or Intel-HEX format. This conversion is needed when debugging the program with the ROM or when creating mask data using the mask data checker.
When creating the ROM-image hex data, the hex converter fills the unused area of each model with 0xff.

6.2 Input/Output Files

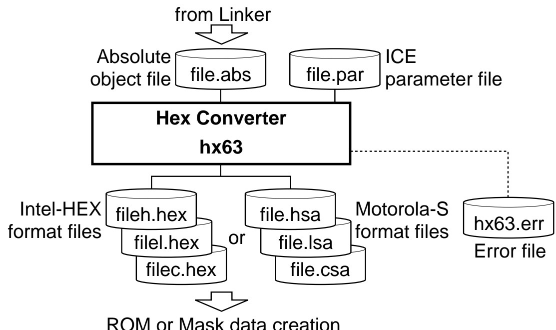


Fig. 6.2.1 Flow chart

6.2.1 Input Files

Absolute object file

File format: Binary file in IEEE-695 format
File name: <File name>.abs (A path can also be specified.)
Description: Absolute object file created by the linker.

ICE parameter file ∗ This file must always be specified.

File format: Binary file
File name: <File name>.par (A path can also be specified.)
Description: File to specify the memory mapping information of each E0C63 Family model. This file is supplied in the development tools for each model and is commonly used with the linker and debugger.

6.2.2 Output Files

Hex file

File format: Text file in Motorola-S format or Intel-HEX format
File name: Motorola-S format <File name>.hsa, <File name>.lsa and <File name>.csa
Intel-HEX format <File name>h.hex, <File name>l.hex and <File name>c.hex
Output destination: Current directory
Description: Three hex files are generated: ".hsa" or "h.hex" that contains the five high-order bits of the object codes with 0b000 extended, ".lsa" or "l.hex" that contains the eight low-order bits and ".csa" or "c.hex" that contains four-bit data for the data ROM.
Motorola-S format files are delivered by default. Intel-HEX format files can be specified using the -i option.

Error file

File format: Text file
File name: hx63.err
Output destination: Current directory
Description: The file is created if the -e start-up option is specified. It records information that the hex converter outputs to the Standard Output (stdout), such as error messages. The file name is "hx63.err" by default, but it can be changed using the -o start-up option.

6.3 Starting Method

General form of command line

hx63 ^ [Options] ^ <Absolute object file name> ^ <ICE parameter file name>

^ denotes a space.
[ ] indicates the possibility to omit.
The order of options and file names can be arbitrary.

File names

Absolute object file: <File name>.abs
ICE parameter file: <File name>.par

The extension of an absolute object file can be omitted. The ICE parameter file must be specified with its extension.

A long file name supported in Windows and a path name can be specified. When including spaces in the file name, enclose the file name with double quotation marks (").

Options

The hex converter comes provided with the following four start-up options:

-b

Function: Conversion of existing codes only
Explanation: Converts and delivers only the object codes that exist in the specified absolute object file. Data for unused addresses is not delivered.
Default: If this option is not specified, the hex data for the entire available memory range of the model is delivered to the output file. Unused addresses are filled with 0xff.

-e

Function: Output of error files
Explanation: Creates an .err file which contains the information that the hex converter outputs to the Standard Output (stdout), such as error messages.
Default: If this option is not specified, no error file will be created.

-i

Function: Conversion into Intel-HEX format
Explanation: Generates the hex files ("h.hex", "l.hex" and "c.hex") in Intel-HEX format.
Default: If this option is not specified, Motorola-S format files (".hsa", ".lsa" and ".csa") are generated.

-o <file name>

Function: Specification of output path/file name
Explanation: Specifies an output path/file name without extension or with an extension ".hsa", ".lsa", ".csa", "h.hex", "l.hex" or "c.hex". By specifying only one file name, three hex files will be generated.
If no extension is specified, an appropriate extension will be supplemented at the end of the specified output path/file name. In this case, ".hsa", ".lsa" or ".csa" is added to the output file name. If Intel-HEX format is specified, "h.hex", "l.hex" or "c.hex" is added to the output file name. It may change a DOS file name (8 character max.) to a long file name for Windows.
Default: The input file name is used for the output file names.

When entering an option in the command line, one or more spaces are necessary before and after the option.

Example: `c:\e0c63\bin\hx63 -e test.abs par63xxx.par`

6.4 Messages

The hex converter delivers all its messages via the Standard Output (stdout).

Start-up message

The hex converter outputs only the following message when it starts up.

```
Hex converter 63 Ver x.xx
Copyright (C) SEIKO EPSON CORP. 199x
```

End message

The hex converter outputs the following messages to indicate which files have been created when it ends normally.

```
Created hex file <FILE NAME>.HSA
Created hex file <FILE NAME>.LSA
Created hex file <FILE NAME>.CSA
Created error log file HX63.ERR

Hex conversion 0 error(s) 0 warning(s)
```

Usage output

If no file name was specified or an option was not specified correctly, the hex converter ends after delivering the following message concerning the usage:

```
Usage: hx63 [options] <file names>
Options: -b             Do not fill unused memory with 0xff
         -e             Output error log file (HX63.ERR)
         -i             Use Intel Hex format
         -o <file name> Specify output file name
File names: Absolute object file (.ABS)
            ICE parameter file (.PAR)
```

When error/warning occurs

If an error occurs, an error message will appear before the end message shows up.
Example:

```
Error : No ICE parameter file specified
Hex conversion 1 error(s) 0 warning(s)
```

In the case of an error, the hex converter ends without creating an output file.

If a warning is issued, a warning message will appear before the end message shows up.
Example:

```
Warning : Output file name conflict
Hex conversion 0 error(s) 1 warning(s)
```

In the case of a warning, the hex converter ends after creating the output files, but the result cannot be guaranteed.

For details on errors and warnings, refer to Section 6.6 "Error/Warning Messages".

6.5 Output Hex Files

6.5.1 Hex File Configuration

Since each E0C63000 instruction has a 13-bit code, the hex converter always generates two hex files for the high-order data and the low-order data of the program code. The low-order data hex file (".lsa" or "l.hex") contains the low-order bytes (bits 7 to 0) of the object codes. The high-order data hex file (".hsa" or "h.hex") contains the high-order bytes (bits 12 to 8 suffixed by high-order bits 0b000).
4-bit data for the data ROM is output to the ".csa" or "c.hex" file.
By specifying the -i option, the hex converter can convert the absolute object file into Intel-HEX files as well as Motorola-S format. However, use Motorola-S format format when loading the hex files to the debugger or creating the mask data by the mask data checker because the debugger and mask data checker do not support Intel-HEX files.

6.5.2 Motorola-S Format

The hex converter converts an absolute object file in the IEEE-695 format into the Motorola-S2 format files by default.
The files are generated with an extension ".hsa" for the high-order program file, ".lsa" for the low-order program file and ".csa" for the data ROM file.
The following shows a sample data in Motorola-S2 format:

```
length
  address                        data                              sum
S224000000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFB
S224000020FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFDB
                                       :
S22400010008E000F04200420606FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF89
                                       :
S804000000FB
```

S2 (1 bytes):	Indicates that the line is a data record.
S8 (1 bytes):	Indicates that the line is an end record (end of data).
length (1 byte):	Indicates the record length of "address + data + sum". The maximum length of a data record is 0x24, while the end record is fixed at 0x04.
address (3 bytes):	Indicates the address where the head data in a record is placed.
data (36 bytes max.):	The object codes are placed here. This is not included in the end record.
sum (1 byte):	This is a checksum (1's complement) from "length" to the last data.

The end records are always "S804000000FB".

6.5.3 Intel-HEX Format

The hex converter converts an absolute object file in the IEEE-695 format into the Intel-HEX format files when the -i option is specified.
The files are generated with a name "<file name>h.hex" for the high-order program file, "<file name>l.hex" for the low-order program file and "<file name>c.hex" for the data ROM file.
The following shows a sample data in Intel-HEX format:

```
data volume   type
  \  address /            data                     sum
:10000000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF00
:10001000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF0
                          :
:1001000008E000F04200420606FFFFFFFFFFFF8E
                          :
:100FF000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF01
                          :
:00000001FF
```

data volume (1 byte): Indicates the data length of each record. The maximum length of a data record is 0x10, while the end record is fixed at 0x00.
address (2 bytes): Indicates the address where the head data in a record is placed.
type (1 byte): Indicates the type of hexadecimal format, currently only "00".
data (16 bytes max.): The object codes are placed here. This is not included in the end record.
sum (1 byte): This is a checksum (2's complement) from "Data volume" to the last data.

The end records are always "00000001FF".

Note: When using hex files for creating the mask data, do not specify Intel-HEX format because the mask data checker does not support this format.

6.5.4 Conversion Range

By default, the hex converter generates the hex files that include all the codes of the ROM area available for each model. Data for unused addresses are delivered as 0xff. For example, if the model has a built-in 2KB code ROM and the program uses the area from address 0x0 to address 0x6ff, the hex converter fills the area from address 0x700 to address 0x7ff with 0xff. If there are unused addresses in the range from 0x0 to 0x6ff, those data are also delivered as 0xff.
When creating the mask data by the mask data checker, the hex files must be generated in this format.

When the -b option is specified, the hex converter does not deliver data in unused addresses of the absolute object file. This allows minimization of the output hex files. Note, however that the hex files generated in this format cannot be used for creating the mask data.

6.6 Error/Warning Messages

6.6.1 Errors

When an error occurs, the hex converter immediately terminates the processing after displaying an error message. It will not output hex files.
The hex converter error messages are given below.

Error message	Description
Cannot create <file kind> file <FILE NAME>	The file cannot be created.
Cannot open <file kind> file <FILE NAME>	The file cannot be opened.
Cannot read <file kind> file <FILE NAME>	The file cannot be read.
Cannot write <file kind> file <FILE NAME>	Data cannot be written to the file.
Illegal file name <FILE NAME> specified with option <option>	The specified hex file name is incorrect.
Illegal ICE parameter at line <line number> of <FILE NAME>	The ICE parameter file contains an illegal parameter setting.
Illegal file name <FILE NAME>	The specified input file name is incorrect.
Illegal option <option>	An illegal option is specified.
Illegal absolute object format	The input file is not an object file in IEEE-695 format.
No ICE parameter file specified	ICE parameter file is not specified.
Out of memory	Cannot secure memory space.

6.6.2 Warning

Even if a warning is issued, the hex converter keeps on processing, and completes the processing after displaying a warning message, unless, in addition, any error occurs.

Warning message	Description
Input file name extension .XXX conflict	Two or more file names with the same extension have been specified. The last one is used.

6.7 Precautions

(1) When creating the hex files for making the mask data file in the mask data checker, specify Motorola-S format and convert for the entire available memory range of the model (do not specify the -b and -i options). Otherwise, an error will occur in the mask data checker. Refer to the "Development Tool Manual" of each model for details of the mask data checker.

(2) If an 8-character output file name (DOS file name) without extension is specified for the Intel-HEX files, it will be changed to a long file name because "h.hex", "l.hex" or "c.hex" is added to the file name.

CHAPTER 7 DISASSEMBLER

This chapter describes the functions of the disassembler, ds63.

7.1 Functions

The disassembler's input is an object in IEEE-695 or Motorola-S format. The code in the object file is disassembled into mnemonics, and output as a source file. The restored source file can be processed in the assembler/linker/hex converter to obtain the same object or hex file.

7.2 Input/Output Files

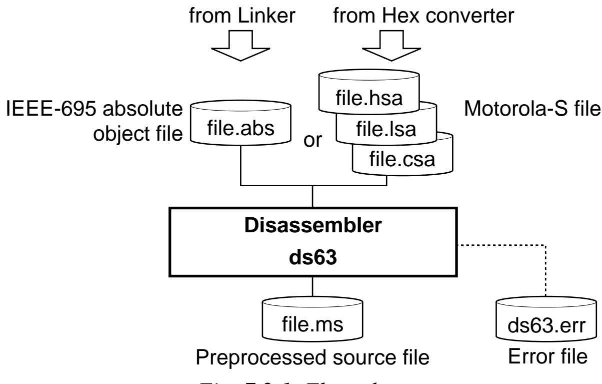


Fig. 7.2.1 Flow chart

7.2.1 Input Files

Absolute object file

File format: Binary file in IEEE-695 format
File name: <File name>.abs (A path can also be specified)
Description: Absolute object file created by the linker

Hex file

File format: Text file in Motorola-S format
File name: <File name>.hsa, <File name>.lsa and <File name>.csa
Description: Hex files created by the hex converter. Three hex files are needed: ".hsa" that contains the four high-order bits of the object codes with 0b000 extended, ".lsa" that contains the eight low-order bits and ".csa" that contains four-bit data for the data ROM. If there is no data ROM, the ".csa" file is not required.

7.2.2 Output Files

Source file

File format: Text file
File name: <File name>.ms
Output destination: Current directory
Description: Disassembled contents of the input file are delivered.

Error file

File format: Text file
File name: ds63.err
Output destination: Current directory
Description: The file is created if the -e start-up option is specified. It records the information that the disassembler outputs to the Standard Output (stdout), such as error messages. The file name is "ds63.err" by default, but it can be changed using the -o start-up option.

7.3 Starting Method

General form of command line

ds63 ^ [Options] ^ <Absolute object or hex file name>

^ denotes a space.
[ ] indicates the possibility to omit.

File names

Absolute object file: <File name>.abs
Motorola-S files: <File name>.hsa, <File name>.lsa, <File name>.csa

The input file must be specified with its extension.
The Motorola-S file can be specified with either ".hsa", ".lsa" or ".csa" as the extension. The other unspecified files will be automatically loaded.

A long file name supported in Windows and a path name can be specified. When including spaces in the file name, enclose the file name with double quotation marks (").

Options

The disassembler comes provided with the following four start-up options:

-cl

Function: Use of lower-case characters
Explanation: Creates all instructions and labels using lower-case characters.
Default: If neither this option nor the -cu option is specified, the source will be made with all labels in upper-case characters and instructions in lower-case characters.

-cu

Function: Use of upper-case characters
Explanation: Creates all instructions and labels using upper-case characters.
Default: If neither this option nor the -cl option is specified, the source will be made with all labels in upper-case characters and instructions in lower-case characters.

-e

Function: Output of error file
Explanation: Creates an .err file which contains the information that the disassembler outputs to the Standard Output (stdout), such as error messages.
Default: If this option is not specified, an error file will not be created.

-o <file name>

Function: Specification of output path/file name
Explanation: Specify an output path/file name without extension or with the extension ".ms". If no extension is specified, ".ms" will be supplemented at the end of the specified output path/file name.
Default: The input file name is used for the output file name.

When entering an option in the command line, one or more spaces are necessary before and after the option.
Example: `c:\e0c63\ds63 -e -o c:\output.ms`

7.4 Messages

The disassembler delivers all its messages via the Standard Output (stdout).

Start-up message

The disassembler outputs the following message when it starts up.

```
Disassembler 63 Ver x.xx
Copyright (C) SEIKO EPSON CORP. 199x
```

End message

The disassembler outputs the following messages to indicate which files have been created when it ends normally.

```
Created preprocessed source file <FILE NAME>.MS
Created error log file DS63.ERR

Disassembly 0 error(s) 0 warning(s)
```

Usage output

If no file name was specified or an option was not specified correctly, the disassembler ends after delivering the following message concerning the usage:

```
Usage: ds63 [options] <file name>
Options: -cl            Use lower case characters
         -cu            Use upper case characters
         -e             Output error log file (DS63.ERR)
         -o <file name> Specify output file name
File names: Absolute object file (.ABS or .CSA/.LSA/.HSA)
```

When error/warning occurs

If an error occurs, an error message will appear before the end message shows up.

Example:

```
Error: Cannot open file TEST.ABS
Disassembly 1 error(s) 0 warning(s)
```

In the case of an error, the disassembler ends without creating an output file.

If a warning is issued, a warning message will appear before the end message shows up.

Example:

```
Warning: Input file name extension .HSA conflict
Disassembly 0 error(s) 1 warning(s)
```

In the case of a warning, the disassembler ends after creating an output file.

For details on errors and warnings, refer to Section 7.6 "Error/Warning Messages".

7.5 Disassembling Output

The data/code mnemonics are restored from the target code. As for the branch instructions, a label will be automatically generated such as "CODEx:" where "x" denotes a hexadecimal number string. Other reference symbols will also be generated as "LABELx", "IOx" and "RAMx". The ".org" pseudo-instruction is used to specify the starting location of each code block.
The following shows examples of disassembled sources:

Sample outputs

Absolute list file "test.als"

```
Linker 63 ver x.xx Absolute list file "TEST.ALS" Sat Nov 07 12:40:41 1998

     1:                       ; sub.s
     2:                       ; AS63 test program (subroutine)
     3:
     4:                               .global RAM_BLK1
     5:
     6:                       ;***** RAM block 1 initialize *****
     7:
     8:                               .global INIT_RAM_BLK1
     9:                       INIT_RAM_BLK1:
    10:   0000   0800                 ldb    %ext,RAM_BLK1@h
    11:   0001   0a04                 ldb    %xl,RAM_BLK1@l        ;set RAM_BLK1 address to x
    12:   0002   1e90                 ld     [%x]+,0x0
    13:   0003   1e90                 ld     [%x]+,0x0
    14:   0004   1e90                 ld     [%x]+,0x0
    15:   0005   1e80                 ld     [%x],0x0              ;set 0x0000 to RAM_BLK1
    16:   0006   1ff8                 ret
    17:
    18:                       ;***** RAM block 1 increment *****
    19:
    20:                               .global INC_RAM_BLK1
    21:                       INC_RAM_BLK1:
    22:   0007   0800                 ldb     %ext,RAM_BLK1@h
    23:   0008   0a04                 ldb     %xl,RAM_BLK1@l       ;set RAM_BLK1 address to x
    24:   0009   1911                 add     [%x]+,1
    25:   000a   1990                 adc     [%x]+,0
    26:   000b   1990                 adc     [%x]+,0
    27:   000c   1980                 adc     [%x],0               ; increment 16bit value
    28:   000d   1ff8                 ret
    29:                       ; main.s
    30:                       ; AS63 test program (main routine)
    31:                       ;
    32:
    33:                       ;***** INITIAL SP1 & SP2 ADDRESS DEFINITION *****
    34:
    35:                       #ifdef SMALL_RAM
    36:                               .set SP1_INIT_ADDR 0xb       ;SP1 init addr = 0x2c
    37:                       #else
    38:                               .set SP1_INIT_ADDR 0x4b      ;SP1 init addr = 0x12c
    39:                       #endif
    40:
    41:                               .set SP2_INIT_ADDR 0x1f      ;SP2 init addr = 0x1f
    42:
    43:
    44:                       ;***** NMI & BOOT, LOOP *****
    45:
    46:                               .global INIT_RAM_BLK1        ; subroutine in sub.s
    47:                               .global INC_RAM_BLK1         ; subroutine in sub.s
    48:
    49:                               .org    0x100
    50:                       NMI:
    51:   0100   08fe   (+)           ldb  ext,fe
    52:   0101   02fe                 calr    INIT_RAM_BLK1        ; initialize RAM block 1
    53:   0102   1ff9                 reti                         ; in NMI(watchdog timer)
    54:
```

```
55:                            .org    0x110
56:                    BOOT:
57:   0110   094b              ldb     %ba,SP1_INIT_ADDR
58:   0111   1fc4              ldb     %sp1,%ba              ; set SP1
59:   0112   091f              ldb     %ba,SP2_INIT_ADDR
60:   0113   1fc6              ldb     %sp2,%ba              ; set SP2
61:   0114   08fe   (+)        ldb  ext,fe
62:   0115   02ea              calr    INIT_RAM_BLK1         ; initialize RAM block 1
63:                    LOOP:
64:   0116   08fe   (+)        ldb  ext,fe
65:   0117   02ef              calr    INC_RAM_BLK1          ; increment RAM block 1
66:   0118   00fd              jr      LOOP                  ; infinity loop
```

Output source file "test.ms" (default)

```
;Disassembler 63 Ver x.xx Assembly source file TEST.MS Fri Nov 06 13:10:20 1998

        .set LABEL1 0x4
        .set LABEL2 0x4
        .set LABEL3 0x4b
        .set LABEL4 0x1f
        .code
        .org    0x0
CODE1:
        ldb %ext,LABEL1@h
        ldb %xl,LABEL1@l
        ld [%x]+,0x0
        ld [%x]+,0x0
        ld [%x]+,0x0
        ld [%x],0x0
        ret
CODE2:
        ldb %ext,LABEL2@h
        ldb %xl,LABEL2@l
        add [%x]+,0x1
        adc [%x]+,0x0
        adc [%x]+,0x0
        adc [%x],0x0
        ret
        .code
        .org    0x100
        ldb %ext,CODE1@rh
        calr CODE1@rl
        reti
        .code
        .org    0x110
        ldb %ba,LABEL3@l
        ldb %sp1,%ba
        ldb %ba,LABEL4@l
        ldb %sp2,%ba
        ldb %ext,CODE1@rh
        calr CODE1@rl
CODE3:
        ldb %ext,CODE2@rh
        calr CODE2@rl
        jr CODE3@rl
```

Output source file "test.ms" (when -cl is specified)

```
;Disassembler 63 Ver x.xx Assembly source file TEST.MS Fri Nov 06 13:10:20 1998

        .set label1 0x4
        .set label2 0x4
        .set label3 0x4b
        .set label4 0x1f
        .code
        .org    0x0
code1:
        ldb %ext,label1@h
        ldb %xl,label1@l
        ld [%x]+,0x0
        ld [%x]+,0x0
        ld [%x]+,0x0
        ld [%x],0x0
        ret
code2:
        ldb %ext,label2@h
        ldb %xl,label2@l
        add [%x]+,0x1
        adc [%x]+,0x0
```

```
        adc [%x]+,0x0
        adc [%x],0x0
        ret
        .code
        .org    0x100
        ldb %ext,code1@rh
        calr code1@rl
        reti
        .code
        .org    0x110
        ldb %ba,label3@l
        ldb %sp1,%ba
        ldb %ba,label4@l
        ldb %sp2,%ba
        ldb %ext,code1@rh
        calr code1@rl
code3:
        ldb %ext,code2@rh
        calr code2@rl
        jr code3@rl
```

Output source file "test.ms" (when -cu is specified)

```
;Disassembler 63 Ver x.xx Assembly source file TEST.MS Fri Nov 06 13:10:20 1998

        .SET LABEL1 0X4
        .SET LABEL2 0X4
        .SET LABEL3 0X4B
        .SET LABEL4 0X1F
        .CODE
        .ORG    0X0
CODE1:
        LDB %EXT,LABEL1@H
        LDB %XL,LABEL1@L
        LD [%X]+,0X0
        LD [%X]+,0X0
        LD [%X]+,0X0
        LD [%X],0X0
        RET
CODE2:
        LDB %EXT,LABEL2@H
        LDB %XL,LABEL2@L
        ADD [%X]+,0X1
        ADC [%X]+,0X0
        ADC [%X]+,0X0
        ADC [%X],0X0
        RET
        .CODE
        .ORG    0X100
        LDB %EXT,CODE1@RH
        CALR CODE1@RL
        RETI
        .CODE
        .ORG    0X110
        LDB %BA,LABEL3@L
        LDB %SP1,%BA
        LDB %BA,LABEL4@L
        LDB %SP2,%BA
        LDB %EXT,CODE1@RH
        CALR CODE1@RL
CODE3:
        LDB %EXT,CODE2@RH
        CALR CODE2@RL
        JR CODE3@RL
```

7.6 Error/Warning Messages

7.6.1 Errors

When an error occurs, the disassembler immediately terminates the processing after displaying an error message. It will not output a source file.
The disassembler error messages are given below.

Error message	Description
Cannot create <file kind> file <FILE NAME>	The file cannot be created.
Cannot open <file kind> file <FILE NAME>	The file cannot be opened.
Cannot read <file kind> file <FILE NAME>	The file cannot be read.
Cannot write <file kind> file <FILE NAME>	Data cannot be written to the file.
Illegal file name <FILE NAME> specified with option <option>	The specified output source file name is incorrect.
Illegal file name <FILE NAME>	The specified input file name is incorrect.
Illegal HEX data format	The input file is not a Motorola-S format file.
Illegal option <option>	An illegal option is specified.
Out of memory	Cannot secure memory space.

7.6.2 Warning

Even if a warning is issued, the disassembler keeps on processing, and completes the processing after displaying a warning message, unless, in addition, an error is produced.

Warning message	Description
Input file name extension .XXX conflict	Two or more file names with the same extension have been specified. The last one is used.
Cannot open Hex file xxx.csa	The file cannot be opened. It is assumed there is no data memory.

CHAPTER 8 DEBUGGER

This chapter describes how to use the Debugger db63.

8.1 Features

The Debugger db63 is used to debug a program after reading an object file in the IEEE-695 format that is generated by the linker.
It has the following features and functions:

- Various data can be referenced at the same time using multiple windows.
- Frequently used commands can be executed from tool bars and menus using a mouse.
- Also available are source display and symbolic debug functions which correspond to assembly source codes.
- Consecutive program execution and two types of single-stepping are possible.
- Five break functions are supported.
- A real-time display function shows register and memory contents on-the-fly.
- A time display function showing execution time by both duration and steps.
- An advanced trace function.
- An automatic command execution function using a command file.

8.2 Input/Output Files

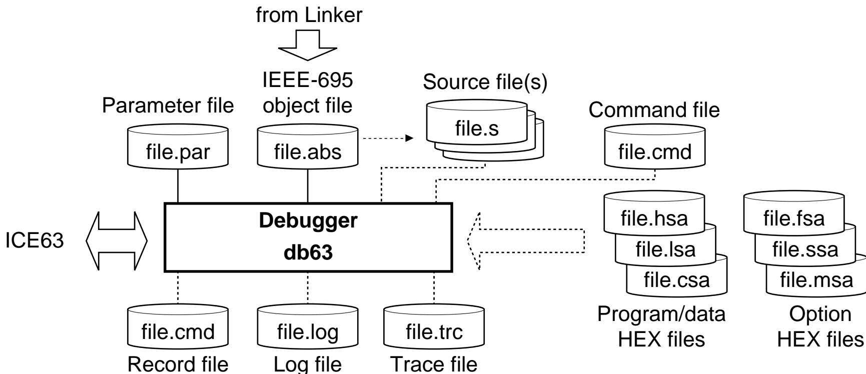


Fig. 8.2.1 Flow chart

8.2.1 Input Files

Parameter file

File format: Binary file
File name: <file name>.par
Description: This file contains memory information on each microcomputer model and is indispensable for starting the debugger. This file is included with the development tool package for each microcomputer model.

The following files are read by the debugger according to command specification.

Object file

File format: Binary file in the IEEE-695 format
File name: <file name>.abs (An extension other than ".abs" can also be used.)
Description: This is an object file generated by the linker. This file is read into the debugger by the ***lf*** command. By reading a file in the IEEE-695 format that contains debug information, source display and symbolic debugging can be performed.

Source file

File format: Text file
File name: <file name>.s
Description: This is the source file of the above object file. It is read when the debugger performs source display.

Program file

File format: HEX file in Motorola-S format
File name: <file name>.hsa, <file name>.lsa
Description: This is a load image file of the program ROM, and is read into the debugger by the ***lo*** command. The file ".hsa" corresponds to the 5 high-order bits of the program code and the file ".lsa" corresponds to the 8 low-order bits of the program code. These files are generated for the purpose of creating mask data from an object file in the IEEE-695 format by the Hex convertor. Unlike files in the IEEE-695 format, these files cannot be used for source display or symbolic debugging, but can be used to check the operation of final program data.

Data file for data ROM

File format: HEX file in Motorola-S format
File name: <file name>.csa
Description: This is a load image file of the data ROM, and is read into the debugger by the ***lo*** command. This file is generated for the purpose of creating mask data from an object file in the IEEE-695 format by the Hex convertor. When an absolute object file in the IEEE-695 format is loaded, it is not necessary to load this file.

Option data file

File format: HEX file in Motorola-S format
File name: <file name>.fsa, <file name>.ssa, <file name>.msa (Varies with the type of microcomputer)
Description: These data files are used to set up hardware options for each microcomputer model and is read by the ***lo*** command. These files are generated by a development tool available for each microcomputer model.

Command file

File format: Text file
File name: <file name>.cmd (An extension other than ".cmd" can also be used.)
Description: This file contains a description of debug commands to be executed successively. By writing a series of frequently used commands in this file, the time and labor required for entering commands from the keyboard can be saved. The command described in the file are read and executed using the ***com*** or ***cmw*** command.

8.2.2 Output Files

Log file

File format: Text file
File name: <file name>.log (An extension other than ".log" can also be used.)
Description: This file contains the information of executed commands and execution results that are output to a file. Output of this file can be controlled by the ***log*** command.

Record file

File format: Text file
File name: <file name>.cmd (An extension other than ".cmd" can also be used.)
Description: This file contains the information of executed commands that are output to a file. Output of this file can be controlled by the ***rec*** command.

Trace file

File format: Text file
File name: <file name>.trc (An extension other than ".trc" can also be used.)
Description: This file contains the specified range of trace information. Output of this file can be controlled by the ***tf*** command.

8.3 Starting Method

8.3.1 Start-up Format

General form of command line

db63 ^ <parameter file name> ^ [start-up option]

^ denotes a space.
[ ] indicates the possibility to omit.

Note: *The parameter file will be recognized by its extension ".par", so ".par" must be included in the parameter file name to be specified.*

8.3.2 Start-up Options

The debugger has three start up options available.

<command file name>

Function: Specifies a command file.
Explanation: For a series of commands to be executed immediately after the debugger starts up, specify a command file that describes those commands.

-comX

Function: Specifies a communication port.
Explanation: This option specifies the communication port through which a personal computer is communicated with by the ICE63. Specify a port number in the X part of this option. The port that can be used for this purpose varies among different personal computers.
Unless this option is specified, the com1 port is used to communicate with the ICE63.

-b <baud rate>

Function: Specifies a communication transmission rate.
Explanation: This option specifies the baud rate on the personal computer. For <baud rate>, select one from 2400, 4800, 9600, 19200, or 38400.
Unless specified otherwise, the baud rate is set to 9600 bps. This value is the same as the initial setting of the ICE63.
The baud rate on the ICE63 is set using the DIP switch mounted on the ICE63.

When entering an option in a command line, make sure that there is at least one space before and after the option.
Example: `c:\e0c63\bin\db63 par63xxx.par startup.cmd -com2 -b 19200`

The default start-up options are set as: -com1 & -b 9600
If no parameter file name was specified or the option was not specified correctly, the debugger ends after delivering the following message concerning the usage:

```
-Usage-
   db63.exe parameter file name <startup options>
Options:
   command file:  ... specifies a command file
   -comX(X:1-4)   ... com port, default com1
   -b             ... baud rate, 2400, 4800, 9600(default), 19200, 38400
```

8.3.3 Start-up Messages

When the debugger starts up, it outputs the following message in the [Command] window. (Refer to the next section for details about windows.)

```
  Debugger63 Ver x.xx Copyright SEIKO EPSON CORP. 199x

Connecting COMx with xxxxx baud rate ... done
Parameter file name      : xxxxxxxx.par
               Version   : xx
               Chip name : 63xxx
CPU version              : x.x
PRC board version        : x.x
LCD board version        : x.x
EXT board version        : x.x
ICE hardware version     : x.x
ICE software version     : x.x
DIAG test                : omitted
Map............................. done
Initialize...................... done
>
```

8.3.4 Hardware Check at Start-up

When the debugger is invoked, it first performs the tests and initializing operations described below.

(1) Testing connection of ICE63

Debugger db63 first checks to see that the ICE63 is connected to your system and that communication is possible without any problems. The following message is displayed in the [Command] window.

During test

```
Connecting COMx with xxxxx baud rate ...
```

When terminated normally

```
Connecting COMx with xxxxx baud rate ... done
```

When an error is encountered

```
Connecting COMx with xxxxx baud rate ... failure
<error message>
```

The error message indicates that communication between the personal computer and ICE63 is not functioning properly. In this case, to verify the following:

- A standard RS-232C cable is used
- The COM port is correct
- The baud rates on both sides are matched
- The PRC board is correctly fitted in place
- The ICE63's power is turned on
- The ICE63 remains reset

For the causes of errors, refer to Section 8.10, "Status/Error/Warning Messages".

If this test indicates that ICE63 is not in ready state, the debugger performs the following:

When the ICE63 is executing the target program:
In this case, the debugger sends a forcible break command to the ICE63; it then retests the connection of the ICE63 several seconds later.

When the ICE63 is in the BUSY state:
In this case, the debugger will try to retest the connection with the ICE63 several seconds later.

When the ICE63 is in a free-run state:
In this case, the debugger displays the following message:

```
Connecting COMx with xxxxx baud rate ... failure
Error : ICE is free run mode
```

Temporarily quit the debugger and set the ICE63 to the ICE mode (by turning the ICE/RUN switch to the ICE position), then restart up the debugger.

When the ICE63 is performing self-diagnosis:
In this case, the debugger waits until the ICE63's self-diagnosis is completed before it starts testing the connection of the ICE63. Note that the ICE63's self-diagnosis is executed simultaneously if it is activated when its DIP switch SW8 is in the up position. If the SW8 switch is in the down position, self-diagnosis is not executed. Self-diagnosis from start to finish requires about 5 minutes. Wait until it is completed.
You will then see the following message:

```
Connecting COMx with xxxxx baud rate ...
DIAG test, please wait 5 min.  .. done
```

If an error is found in self-diagnosis, an error message will be displayed on the screen instead of "done" above.

(2) Version check

When the connection test terminates normally, the debugger checks the contents of the parameter file, the version of the ICE63, and the versions of the boards inserted in the ICE63.

```
Parameter file name      : xxxxxxxx.par
               Version   : xx
               Chip name : 63xxx
CPU version              : x.x
PRC board version        : x.x
LCD board version        : x.x
EXT board version        : x.x
ICE hardware version     : x.x
ICE software version     : x.x
DIAG test                : omitted
```

Here, the debugger checks to see if the ICE's system configuration (including the PRC and LCD extension boards) and their versions are matched to the setup contents of the parameter file.
If the ICE63 does not have a necessary board, or contains an unnecessary board or a board of different version, a warning message appears on the screen.

(3) ICE initialization

When the above tests are finished, the debugger initializes the ICE as follows:
- Mapping (memory configuration is set according to the parameter file)
- Initializing mapped memory (RAM: 0xa; code ROM: 0x1fff; data ROM: 0xf)
- Initializing option data (cleared to 0)
- Initializing break conditions (all break conditions are cleared)
- Initializing trace conditions (normal trace is set and the trace trigger point is set to 0)
- Setting execution cycles counter to 0.
- Initial setting of watch data addresses (addresses 0, 4, 8, and C)
- Initializing CPU registers

When initialization is terminated normally:

```
Map.............................. done
Initialize....................... done
>
```

When an error is encountered:

```
Map.............................. done
Initialize....................... Error

    Please quit db63 and restart!
>
```

If an error occurs in the above initialization process, temporarily quit the debugger. Check the cause of the error and repair it before restarting the debugger.

After initialization, the state of the screen including the position and size of the windows will return the same as the last time the debugger was terminated. The contents displayed in each window if it is opened are as follows:

Window	Display contents
[Command] window	Initialization information (and waits for command input)
[Data] window	Data memory contents starting from data memory address 0
[Register] window	Current register values
[Source] window	Program memory contents starting from program memory address 0x0100 The previously set display mode (Unassemble, Source or Mix) is used.
[Trace] window	Blank

8.3.5 Method of Termination

To terminate the debugger, select [Exit] from the [File] menu.

You can also input the ***q*** command in the [Command] window to terminate the debugger.

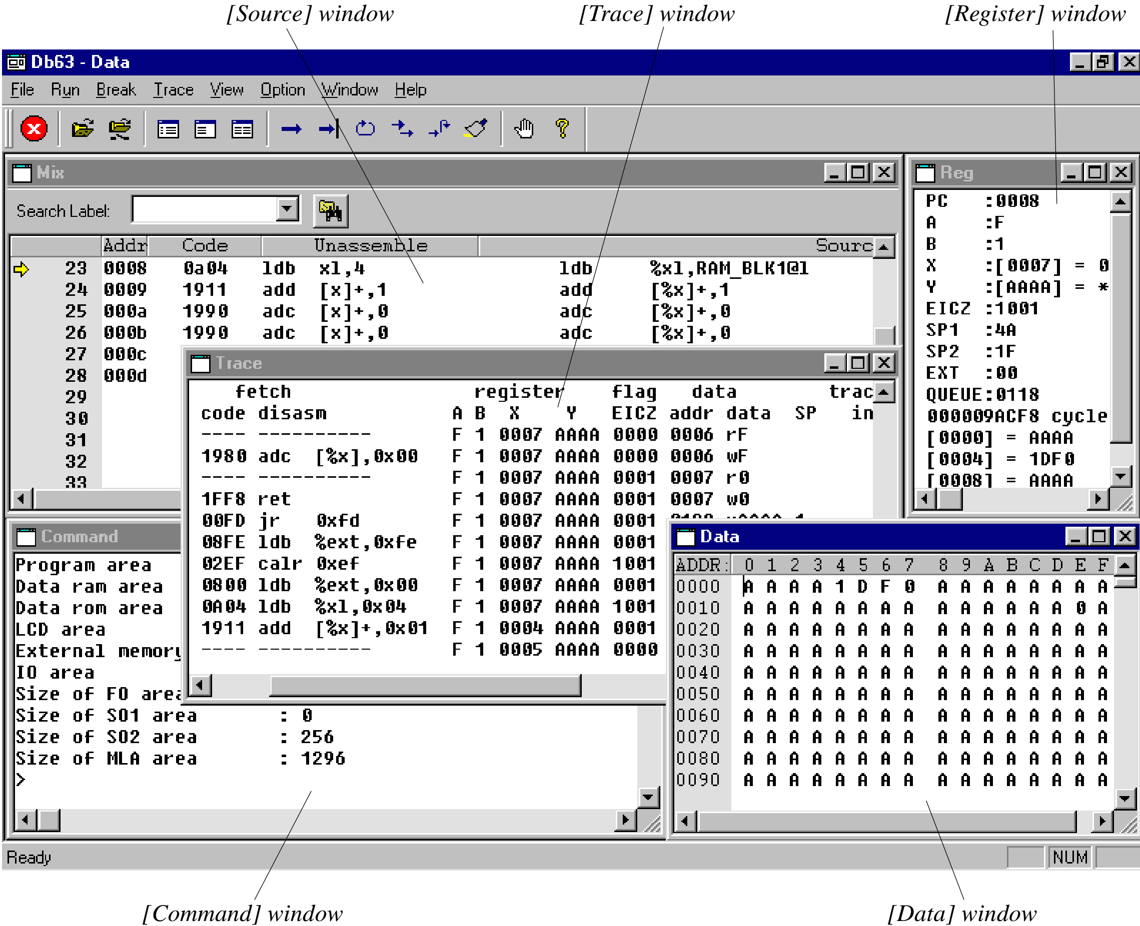
```
>q
```

8.4 Windows

This section describes the types of windows used by the debugger.

8.4.1 Basic Structure of Window

The diagram below shows the window structure of the debugger.



Depending on the computer used, the windows may differ from the above display depending on the screen resolution, the number of dots in system font, etc. Adjust the size of each window to suit needs.

Features common to all windows

(1) Open/close and activating a window

All windows except [Command] can be closed or opened.
To open a window, select the window name from the [View] menu. To close a window, click the [Close] box on the window. After initialization, the state of the screen including the position and size of the windows will return to the same as the last time the debugger was terminated.
The opened windows are listed in the [Window] menu. Selecting one from the list activates the selected window. It can also be done by simply clicking on an inactive window. Furthermore, pressing [Ctrl]+[Tab] switches the active window to the next open window.

(2) Resizing and moving a window

Each window can be resized as needed by dragging the boundary of the window with the mouse. The [Minimize] and [Maximize] buttons work in the same way as in general Windows applications. Each window can be moved to the desired display position by dragging the window's title bar with the mouse. However, windows can only be resized and moved within the range of the application window.

(3) Scrolling a window

All windows can be scrolled. (The [Register] window can be scrolled only when its size is reduced.)
Use one of the following three methods to scroll a window:

1. Click on an arrow button or enter an arrow key (cursor movement) to scroll a window one line at a time.
2. Click on the scroll bar of a window to scroll it one page (current window size) at a time.
3. Drag the scroll bar handle of a window to move it to the desired area.

(4) Other

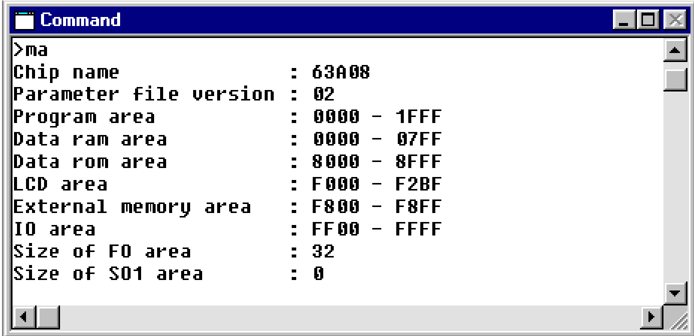
The opened windows can be cascaded or tiled using the [Window] menu.

Note for display

The windows may display incorrect contents caused by incompatibility between the OS and the video card or driver. If there is any problem try the following methods to fix it.

- Update the driver to the latest version if an older version has been installed.
  Please inquire about the version to the distributor.
- If the driver allows selection of extended function such as acceleration, turn the functions off.
- If the problem is not fixed using the above, try the standard driver supplied with Windows95 (NT).

8.4.2 [Command] Window



The [Command] window is used to do the following:

(1) Entering debug commands

When the prompt ">" appears in the [Command] window, the system will accept a command entered from the keyboard.
If some other window is selected, click on the [Command] window. A cursor will blink at the prompt, indicating that readiness to input a command. (Refer to Section 8.7.1, "Entering Commands from Keyboard".)

(2) Displaying debug commands selected from menus or tool bar

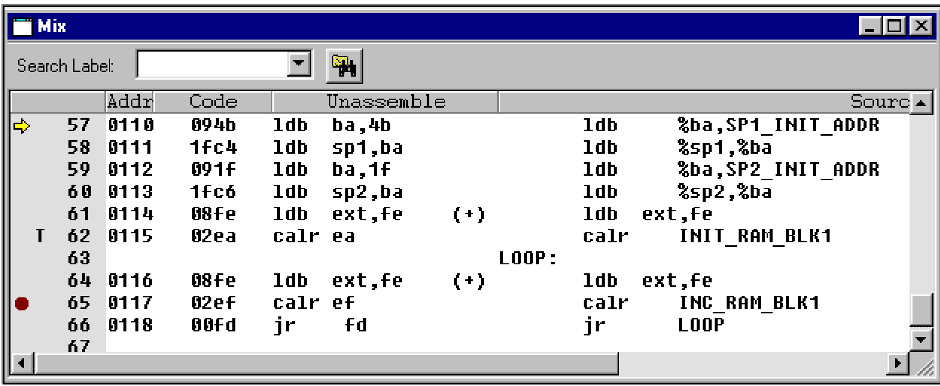
When a command is executed by selecting the menu item or tool bar button, the executed command line is displayed in the [Command] window.

(3) Displaying command execution results

The [Command] window displays command execution results. However, some command execution results are displayed in the [Source], [Data], [Register], or [Trace] windows. The contents of these execution results are displayed when their corresponding windows are open. If the corresponding window is closed, the execution result is displayed in the [Command] window.
When writing to a log file, the content of the write data is displayed in the window. (Refer to the description for ***log*** command.)

Note: The [Command] window cannot be closed.

8.4.3 [Source] Window



The [Source] window displays the contents of (1) to (4) listed below. This window also allows breakpoints to be set and words or labels to be found.

(1) Unassembled codes and source codes

You can choose one of the following three display modes:

1. Mix mode

[Mix] button

(selected by the [Mix] button or entering the ***m*** command)
In this mode, the window displays the addresses, codes, unassembled contents, and corresponding source line numbers and source statements. (See the diagram above.)

2. Source mode

[Source] button

(selected by the [Source] button or entering the ***sc*** command)
In this mode, the window displays the source line numbers and source statements.

3. Unassemble mode

[Unassemble] button

(selected by the [Unassemble] button or entering the ***u*** command)
In this mode, the window displays the addresses, codes, and unassembled contents. This format is selected when the debugger starts up.

*Note: The **m**, **sc** and **u** commands can update the [Source] window if the window is already opened. If the [Source] window is closed, the program code is displayed in the [Command] window. The [Mix], [Source] and [Unassemble] buttons open the [Source] window if the window is closed.*

All program code in the 64K address space can be referenced by scrolling the window. When a break occurs, the display content is updated so that the address line to be executed next is displayed, with an arrow mark at the beginning of the line for identification.
Use the scroll bar or arrow keys to scroll the window. Or enter a command to display the program code beginning with a specified position.

** Display of source line numbers and source statements*

The source line numbers and source statements can only be displayed when the IEEE-695 absolute object file including debugging information for the source display is loaded. Furthermore, the source statements that are actually displayed from this file are those which have had the -g option specified by the assembler.

* ***Updating of display***

When a program is loaded and executed (*g*, *gr*, *s*, *n*, or ***rst*** command), or the memory contents are changed (***a (as)***, *pe*, *pf*, or *pm* command), the display contents are updated. In this case the [Source] window updates its display contents so that the current PC address can always be displayed. The display contents are also updated when the display mode is changed.

(2) Current PC

The current PC (program counter) address line is indicated by an arrow mark at the beginning of the line. (Address 0x0110 in the diagram)

(3) PC breakpoint

The address line where a breakpoint is set is indicated by a red ● mark at the beginning of the line. (Address 0x0117 in the diagram)

(4) Trace trigger point

The address line where a trace trigger point is set is indicated by the letter "T" at the beginning of the line. (Address 0x0115 in the diagram)

(5) Break setting at the cursor position

Place the cursor at an address line where a breakpoint is to be set (not available for a source-only line).

[Break] button

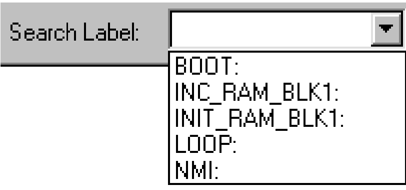
Then click on the [Break] button. A PC breakpoint will be set at that address. If the same is done at the address line where a PC breakpoint has been set, the breakpoint will be cleared.

[Go to Cursor] button

If the [Go to Cursor] button is clicked, the program will execute beginning with the current PC position, and program execution breaks at the line where the cursor is located.

(6) Finding labels and words

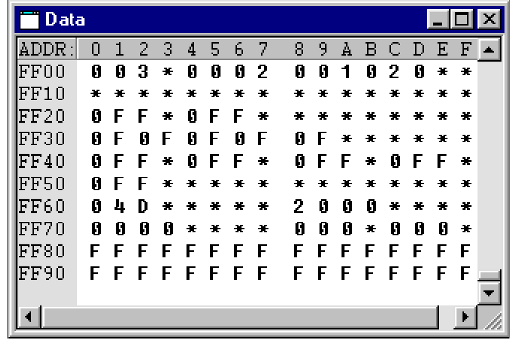
Any labels and words can be found using the [Search Label] pull-down list box or the [Find] button on the [Source] window.



[Search Label] pull-down list box

[Find] button

8.4.4 [Data] Window



(1) Displaying data memory contents

The [Data] window displays the memory dump results in hexadecimal numbers.
The display area is the entire 64K-word data memory space (RAM, data ROM, I/O). The contents of all addresses from 0x0000 to 0xffff can be displayed by scrolling the window. The contents of unmapped addresses in each microcomputer model are indicated by an "*".

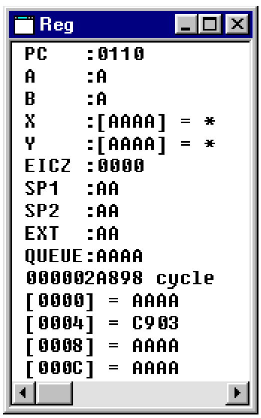
* ***Updating of display***

The display contents of the [Data] window are updated automatically when memory contents are modified with a command (***de***, ***df***, or ***dm*** command), or by direct modification. After executing the program (***g***, ***gr***, ***s***, ***n***, or ***rst*** command), the display contents are also updated. To refresh the [Data] window manually, execute the ***dd*** command or click the vertical scroll bar.

(2) Direct modification of data memory contents

The [Data] window allows direct modification of data memory contents. To modify data on the [Data] window, place the cursor at the front of the data to be modified or double click the data, and then type a hexadecimal character (0–9, a–f). Data in the address will be modified with the entered number and the cursor will move to the next address. This allows successive modification of a series of addresses.

8.4.5 [Register] Window



(1) Displaying register contents

The [Register] window displays the contents of the PC, A register, B register, X register and its memory, Y register and its memory and flags (E, I, C, Z), stack pointers (SP1, SP2), EXT register, and QUEUE register.

(2) Execution cycle counter

This counter calculates and indicates the number of executed cycles or execution time since the CPU was reset.

(3) Monitor data

The debugger allows you to specify four addresses in RAM and monitor the memory contents at these addresses. The [Register] window displays the contents of these four watch data addresses (4 words each beginning from the specified address). When the debugger starts up, addresses 0, 4, 8, and C are initially set as the watch data addresses. The contents are arranged sequentially from left to right in order of their addresses as they are displayed on the screen.

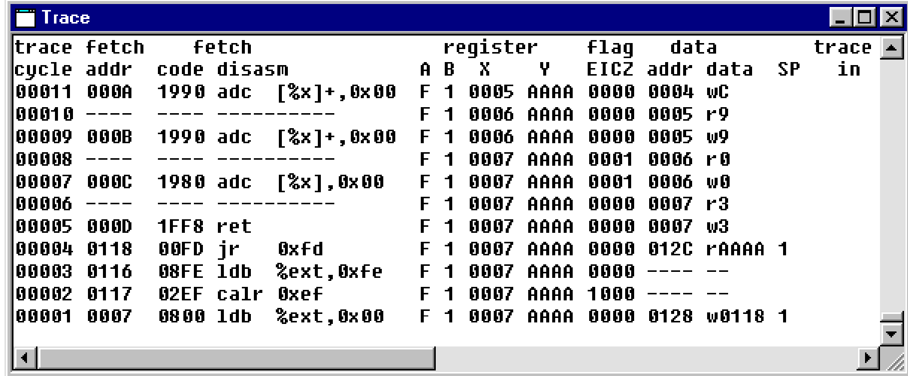
* ***Updating the display***

The display is updated when registers are dumped (***rd*** command), when watch data addresses are set (***dw*** command), when register data is modified (***rs*** command), when the CPU is reset (***rst*** command), or after program execution (***g***, ***gr***, ***s***, or ***n*** command) is completed.
When the on-the-fly function is enabled, the PC, flag and watch data are updated in real time at 0.5 second intervals while the program is being executed. Other contents are left blank until the program is stopped by a break.

(4) Direct modification of register contents

The [Register] window allows direct modification of register contents. To modify data on the [Register] window, select (highlight) the data to be modified and type a hexadecimal number (0–9, a–f), then press [Enter]. The register data will be modified with the entered number.

8.4.6 [Trace] Window



The [Trace] window displays the trace result up to 8,192 cycles by reading it from the ICE63's trace memory.
The following lists the trace contents:
• Traced cycle number
• Fetched address
• Fetched code and disassembled contents
• Register contents (A, B, X, Y, and flags)
• Memory access status (address, R/W, data, and SP1/SP2)
• TRCIN pin input status

This window also displays the trace data search results by the ***ts*** command.

∗ ***Updating of display:***

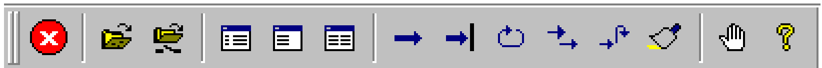
The contents of the [Trace] window are cleared when the target program is being executed. During this period, the [Trace] window does not accept scrolling and resizing operations.
After an program execution is terminated, this window displays the latest data traced during the execution. To specify a display start cycle, execute the ***td*** command.

8.5 Tool Bar

This section outlines the tool bar available with the debugger.

8.5.1 Tool Bar Structure

The tool bar has 14 buttons, each one assigned to a frequently used command.



The specified function is executed when you click on the corresponding button.

8.5.2 [Key Break] Button

This button forcibly breaks execution of the target program. This function can be used to cause the program to break when the program has fallen into an endless loop.

8.5.3 [Load File] and [Load Option] Buttons

[Load File] button

This button reads an object file in the IEEE-695 format into the debugger. It performs the same function when the ***lf*** command is executed.

[Load Option] button

This button reads a program file, data file for the data ROM or an optional HEX file in Motorola-S format into the debugger. It performs the same function when the ***lo*** command is executed.

8.5.4 [Source], [Mix], and [Unassemble] Buttons

These buttons open the [Source] window or switch over the display modes.

[Source] button

This button switches the display of the [Source] window to the source mode. The [Source] window opens if it is closed. This button performs the same function when the ***sc*** command is executed.

[Unassemble] button

This button switches the display of the [Source] window to the unassemble mode. The [Source] window opens if it is closed. This button performs the same function when the ***u*** command is executed.

[Mix] button

This button switches the display of the [Source] window to the mix mode (unassemble & source). The [Source] window opens if it is closed. This button performs the same function when the ***m*** command is executed.

8.5.5 [Go], [Go to Cursor], [Go from Reset], [Step], [Next], and [Reset] Buttons

[Go] button

This button executes the target program from the address indicated by the current PC. It performs the same function when the ***g*** command is executed.

[Go to Cursor] button

This button executes the target program from the address indicated by the current PC to the cursor position in the [Source] window (the address of that line). It performs the same function when the ***g <address>*** command is executed.
Before this button can be selected, the [Source] window must be open and the address line where the program is to break must be clicked. Selecting a break address by clicking on the address line is valid for only the lines that have actual code, and is invalid for the source-only lines.

[Go from Reset] button
This button resets the CPU and then executes the target program from the program start address (0x110). It performs the same function when the ***gr*** command is executed.

[Step] button
This button executes one instruction step at the address indicated by the current PC. It performs the same function when the ***s*** command is executed.

[Next] button
This button executes one instruction step at the address indicated by the current PC. If the instruction to be executed is calr, calz or int, it is assumed that a program section until control returns to the next address constitutes one step and all steps of their subroutines are executed. This button performs the same function when the ***n*** command is executed.

[Reset] button
This button resets the CPU. It performs the same function when the ***rst*** command is executed.

8.5.6 [Break] Button

Use this button to set and clear a breakpoint at the address where the cursor is located in the [Source] window. This function is valid only when the [Source] window is open. Note that selecting a break address by clicking on the address line is valid for only the lines that have actual code and is invalid for the source-only lines.

8.5.7 [Help] Button

By clicking on this button, a help window appears on the screen, displaying the contents of help topics.

8.6 Menu

This section outlines the menu bar available with the debugger.

8.6.1 Menu Structure

The menu bar has eight menus, each including frequently-used commands.

8.6.2 [File] Menu

[Load File...]
This menu item reads an object file in the IEEE-695 format into the debugger. It performs the same function when the ***lf*** command is executed.

[Load Option...]
This menu item reads a program file, data file for the data ROM or an optional HEX file in Motorola-S format into the debugger. It performs the same function when the ***lo*** command is executed.

[Flash Memory Operation...]
This menu item reads/writes data from/to the Flash memory or erases the Flash memory contents. It performs the same function when the ***lfl***, ***sfl*** or ***efl*** command is executed.

[Exit]
This menu item quits the debugger. It performs the same function when the ***q*** command is executed.

8.6.3 [Run] Menu

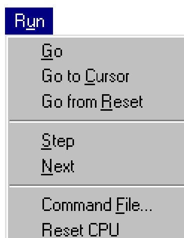
[Go]
This menu item executes the target program from the address indicated by the current PC. It performs the same function when the ***g*** command is executed.

[Go to Cursor]
This menu item executes the target program from the address indicated by the current PC to the cursor position in the [Source] window (the address of that line). It performs the same function when the ***g <address>*** command is executed.
Before this menu item can be selected, the [Source] window must be open and the address line where the program is to break must be clicked. Selecting a break address by clicking on the address line is valid for only the lines that have actual code, and is invalid for the source-only lines.

[Go from Reset]
This menu item resets the CPU and then executes the target program from the program start address (0x0110). It performs the same function when the ***gr*** command is executed.

[Step]
This menu item executes one instruction step at the address indicated by the current PC. It performs the same function when the ***s*** command is executed.

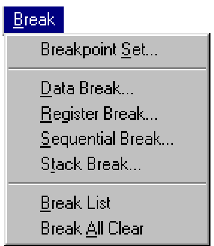
[Next]
This menu item executes one instruction step at the address indicated by the current PC. If the instruction to be executed is calr, calz or int, it is assumed that a program section until control returns to the next address constitutes one step and all steps of their subroutines are executed. This menu item performs the same function when the ***n*** command is executed.



[Command File...]
This menu item reads a command file and executes the debug commands written in that file. It performs the same function when the ***com*** or ***cmw*** command is executed.

[Reset CPU]
This menu item resets the CPU. It performs the same function when the ***rst*** command is executed.

8.6.4 [Break] Menu



[Breakpoint Set...]
This menu item displays, sets or clears PC breakpoints using a dialog box. It performs the same function as executing the ***bp*** command.

[Data Break...]
This menu item displays, sets or clears data break conditions using a dialog box. It performs the same function as executing the ***bd*** command.

[Register Break...]
This menu item displays, sets or clears register break conditions using a dialog box. It performs the same function as executing the ***br*** command.

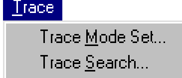
[Sequential Break...]
This menu item displays, sets or clears sequential break conditions using a dialog box. It performs the same function as executing the ***bs*** command.

[Stack Break...]
This menu item displays or sets stack break conditions using a dialog box. It performs the same function as executing the ***bsp*** command.

[Break List]
This menu item displays the all break conditions that have been set. It performs the same function as executing the ***bl*** command.

[Break All Clear]
This menu item clears all break conditions. It performs the same function as executing the ***bac*** command.

8.6.5 [Trace] Menu



[Trace Mode Set...]
This menu item sets a trace mode and trace conditions using a dialog box. It performs the same function as executing the ***tm*** command.

[Trace Search...]
This menu item searches trace information from the trace memory under the condition specified using a dialog box. It performs the same function as executing the ***ts*** command.

[Trace File...]
This menu item saves the specified range of the trace information displayed in the [Trace] window to a file. It performs the same function as executing the ***tf*** command.

8.6.6 [View] Menu

[Command]
This menu item activates the [Command] window.

[Program]

This menu item opens or activates the [Source] window and displays the program from the current PC address in the display mode selected from the sub menu items. These sub menu items perform the same functions as executing the ***u***, ***sc***, and ***m*** command, respectively.

[Data Dump]
This menu item opens or activates the [Data] window and displays the data memory contents from the memory start address.

[Register]
This menu item opens or activates the [Register] window and displays the current values of the registers.

[Trace]
This menu item opens or activates the [Trace] window and displays the trace data sampled in the ICE trace memory.

[Toolbar]
This menu item shows or hides the toolbar.

[Status Bar]
This menu item shows or hides the status bar.

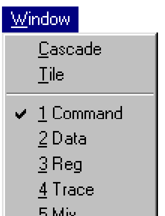
8.6.7 [Option] Menu

[Log...]
This menu item starts or stops logging using a dialog box. It performs the same function as executing the ***log*** command.

[Record...]
This menu item starts or stops recording of a command execution using a dialog box. It performs the same function as executing the ***rec*** command.

[Mode Setting...]
This menu item sets the on-the-fly display, break and execution counter modes using a dialog box. It performs the same functions as executing the ***md*** command.

8.6.8 [Windows] Menu



[Cascade]
This menu item cascades the opened windows.

[Tile]
This menu item tiles the opened windows.

This menu shows the currently opened window names. Selecting one activates the window.

8.6.9 [Help] Menu

[Contents...]
This menu item displays the contents of help topics.

[About Db63...]
This menu item displays an About dialog box for the debugger.

8.7 Method for Executing Commands

All debug functions can be performed by executing debug commands. This section describes how to execute these commands. Refer to the description of each command for command parameters and other details.

To execute a debug command, activate the [Command] window and input the command from the keyboard. The menu and tool bar can be used to execute frequently-used commands.

8.7.1 Entering Commands from Keyboard

Select the [Command] window (by clicking somewhere on the [Command] window). When the prompt ">" appears on the last line in this window and a cursor is blinking behind it, the system is ready to accept a command from the keyboard.
Input a debug command at the prompt position. The commands are not case-sensitive; they can be input in either uppercase or lowercase.

General command input format

>command [parameter [parameter ... parameter]] ↵

- A space is required between a command and parameter.
- A space is required between parameters.

Use the arrow keys, [Back Space] key, or [Delete] key to correct erroneous input.
When you press the [Enter] key after entering a command, the system executes that command. (If the command entered is accompanied by guidance, the command is executed when the necessary data is input according to the displayed guidance.)

Input example:

```
>g↵                    (Only a command is input.)
>com test.cmd↵         (A command and parameter are input.)
```

Command input accompanied by guidance

For commands that cannot be executed unless a parameter or the commands that modify the existing data are specified, a guidance mode is entered when only a command is input. In this mode, the system brings up a guidance field, so input a parameter there.

Input example:

```
>lf↵
File name   ? test.abs↵   ... Input data according to the guidance (underlined part).
>
```

- **Commands requiring parameter input as a precondition**

The *lf* command shown in the above example reads an absolute object file into the debugger. Commands like this that require an entered parameter as a precondition are not executed until the parameter is input and the [Enter] key pressed. If a command has multiple parameters to be input, the system brings up the next guidance, so be sure to input all necessary parameters sequentially. If the [Enter] key is pressed without entering a parameter in some guidance session of a command, the system assumes the command is canceled and does not execute it.

- **Commands that replace existing data after confirmation**

  The commands that rewrite memory or register contents one by one provide the option of skipping guidance (do not modify the contents), returning to the immediately preceding guidance, or terminating during the input session.

  [Enter] key .... Skips input.
  [^] key .... Returns to the immediately preceding guidance.
  [q] key .... Terminates the input session.

  Input example:

```
>de↵                                ... Command to modify data memory.
Data enter address ? :0↵            ... Inputs the start address.
0000   A:1↵                         ... Modifies address 0x0000 to 1.
0001   A:^↵                         ... Returns to the immediately preceding address.
0000   1:0↵                         ... Inputs address 0x0000 back again.
0001   A:↵                          ... Skips address 0x0001 by pressing [Enter] alone.
0002   A:↵
0001   A:q↵                         ... Terminates the input session.
>
```

Numeric data format of parameter

For numeric values to be accepted as a parameter, they must be input in hexadecimal numbers for almost all commands. However, some parameters accept decimal or binary numbers.

The following characters are valid for specifying numeric data:

Hexadecimal: 0–9, a–f, A–F, ∗
Decimal: 0–9
Binary: 0, 1, ∗
("∗" is used to mask bits when specifying a data pattern.)

Specification with a symbol

For address specifications, symbols defined in the source can also be used. However, it is necessary to load an absolute object file that contains debug information.
Symbols should be used as follows:

Global symbol	@<symbol name>	e.g. @RAM_BLK1
Local symbol	@<symbol name>@<source file name>	e.g. @LOOP@main.s

Successive execution using the [Enter] key

The commands listed below can be executed successively by using only the [Enter] key after executing once. Successive execution here means repeating the previous operation or continuous display of the previous contents.

Execution commands: ***g*** (go), ***s*** (step), ***n*** (next), ***com*** (execute command file)
Display commands: ***sc*** (source), ***m*** (mix), ***u*** (unassemble), ***dd*** (data memory dump), ***od*** (option data dump), ***td*** (trace data display), ***cv*** (coverage), ***sy*** (symbol list), ***ma*** (map information)

The successive execution function is terminated when some other command is executed.

8.7.2 Executing from Menu or Tool Bar

The menu and tool bar are assigned frequently-used commands as described in Sections 8.5 and 8.6. A command can be executed simply by selecting desired menu command or clicking on the tool bar button. Table 8.7.2.1 lists the commands assigned to the menu and tool bar.

Table 8.7.2.1 Commands that can be specified from menu or tool bar

Command	Function	Menu	Button
lf	Load IEEE-695 absolute object file	[File | Load File...]	
lo	Load Motorola-S file	[File | Load Option...]	
g	Execute program successively	[Run | Go]	
g <address>	Execute program to <address> successively	[Run | Go to Cursor]	
gr	Reset CPU and execute program successively	[Run | Go from Reset]	
s	Step into	[Run | Step]	
n	Step over	[Run | Next]	
com, cmw	Load and execute command file	[Run | Command File...]	–
rst	Reset CPU	[Run | Reset CPU]	
bp, bc (bpc)	Set/clear PC breakpoint	[Break | Breakpoint Set...]	
bd, bdc	Set/clear data break	[Break | Data Break...]	–
br, brc	Set/clear register break	[Break | Register Break...]	–
bs, bsc	Set/clear sequential break	[Break | Sequential Break...]	–
bsp	Set stack break	[Break | Stack Break...]	–
bl	Break list	[Break | Break List...]	–
bac	Clear all break conditions	[Break | Break All Clear]	–
tm	Set trace mode	[Trace | Trace Mode Set...]	–
ts	Search trace information	[Trace | Trace Search...]	–
tf	Save trace information to a file	[Trace | Trace File...]	–
u	Unassemble display	[View | Program | Unassemble]	
sc	Source display	[View | Program | Source Display]	
m	Mix display	[View | Program | Mix Mode]	
lfl	Load from flash memory	[File | Flash Memory Operation...]	–
sfl	Save to flash memory	[File | Flash Memory Operation...]	–
efl	Erase flash memory	[File | Flash Memory Operation...]	–
dd	Dump data memory	[View | Data Dump]	–
rd	Display register values	[View | Register]	–
td	Display trace information	[View | Trace]	–
log	Turn log output on or off	[Option | Log...]	–
rec	Record commands to a command file	[Option | Record...]	–
md	Set modes	[Option | Mode Setting...]	–

8.7.3 Executing from a Command File

Another method for executing commands is to use a command file that contains descriptions of a series of debug commands. By reading a command file into the debugger the commands written in it can be executed.

Creating a command file

Create a command file as a text file using an editor.
Although there are no specific restrictions on the extension of a file name, Seiko Epson recommends using ".cmd".

Command files can also be created using the ***rec*** command. The ***rec*** command creates a command file and saves the executed commands to the file.

Example of a command file

The example below shows a command group necessary to read an object file and an option file.

Example: File name = startup.cmd

```
lf test.abs
lo test.fsa
lo test.ssa
```

A command file to write the commands that come with a guidance mode can be executed. In this case, be sure to break the line for each guidance input item as a command is written.

Reading in and executing a command file

There are two methods to read a command file into the debugger and to execute it, as described below.

(1) Execution by the start-up option

By specifying a command file in the debugger start-up command, one command file can be executed when the debugger starts up.
If the above example of a command file is specified, for example, the necessary files are read into the debugger immediately after the debugger starts up, so everything is ready to debug the program.

Example: Startup command of the debugger

```
db63 startup.cmd par63xxx.par
```

(2) Execution by a command

The debugger has the ***com*** and ***cmw*** commands available that can be used to execute a command file.
The ***com*** command reads in a specified file and executes the commands in that file sequentially in the order they are written.
The ***cmw*** command performs the same function as the ***com*** command except that each command is executed at intervals specified by the ***md*** command (1 to 256 seconds).

Examples:

```
com startup.cmd

cmw test.cmd
```

The commands written in the command file are displayed in the [Command] window.

Restrictions

Another command file can be read from within a command file. However, nesting of these command files is limited to a maximum of five levels. An error is assumed and the subsequent execution is halted when the ***com*** or ***cmw*** command at the sixth level is encountered.

8.7.4 Log File

The executed commands and the execution results can be saved to a file in text format that is called a "log file". This file allows verification of the debug procedures and contents.
The contents displayed in the [Command] window are saved to this file.

Command example

```
>log tst.log
```

After the debugger is set to the log mode by the ***log*** command (after it starts outputting to a log file), the ***log*** command toggles (output turned on in log mode ↔ output turned off in normal mode). Therefore, you can output only the portions needed can be output to the log file.

Display of [Command] window in log mode

The contents displayed in the [Command] window during log mode differ from those appearing in normal mode.

(1) When executing a command when each window is open

(When the window that displays the command execution result is opened)

Normal mode: The contents of the relevant display window are updated. The execution results are not displayed in the [Command] window.

Log mode: The same contents as those displayed in the relevant window are also displayed in the [Command] window. However, changes made to the relevant window by scrolling or opening it are not reflected in the [Command] window.

(2) When executing a command while each window is closed

When the relevant display window is closed, the execution results are always displayed in the [Command] window regardless of whether operation is in log mode or normal mode.

For the display format in the [Command] window, refer to each command description.

8.8 Debug Functions

This section outlines the debug features of the debugger, classified by function.
Refer to Section 8.9, "Command Reference" for details about each debug command.

8.8.1 Loading Program and Data Files

Loading files

The debugger can read a file in IEEE-695 format or Motorola-S format in the debugging process. Table 8.8.1.1 lists the files that can be read by the debugger and the load commands.

Table 8.8.1.1 Files and load commands

File type	Data type	Ext.	Generation tool	Com.	Menu	Button
IEEE-695	Program/data	.abs	Linker	**lf**	[File | Load File...]	
Motorola-S	Program (5 high-order bits)	.hsa	HEX convertor	**lo**	[File | Load Option...]	
	Program (8 low-order bits)	.lsa	HEX convertor			
	Function option	.fsa	Function option generator			
	Segment option	.ssa	Segment option generator			
	Melody data	.msa	Melody assembler			

(Ext. = Extension, Com. = Command)

Debugging a program with source display

To debug a program using the source display and symbols, the object file must be in IEEE-695 format read into the debugger. If any other program file is read, only the unassemble display is produced.

8.8.2 Source Display and Symbolic Debugging Function

The debugger allows program debugging while displaying the assembly source statements. Address specification using a symbol name is also possible.

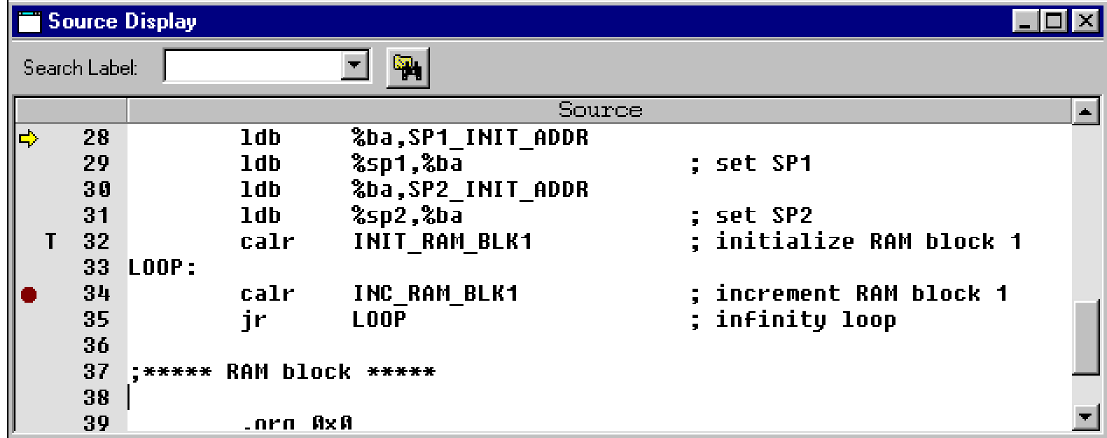
Displaying program code

The [Source] window displays the program in the specified display mode. The display mode can be selected from among the three modes: Unassemble mode, Source mode, Mix mode.

Table 8.8.2.1 Commands/tool bar buttons to switch display mode

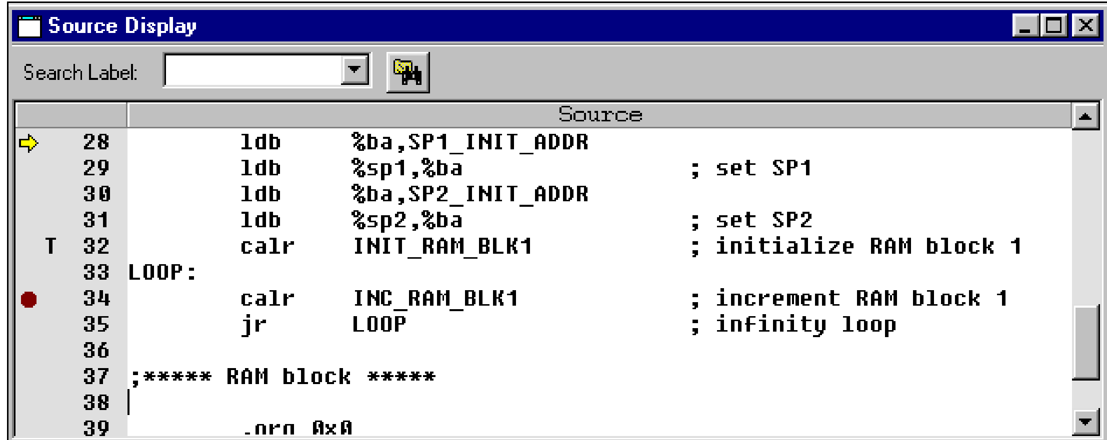
Display mode	Command	Menu	Button
Unassemble	**u**	[View | Program | Unassemble]	
Source	**sc**	[View | Program | Source Display]	
Mix	**m**	[View | Program | Mix Mode]	

(1) Unassemble mode



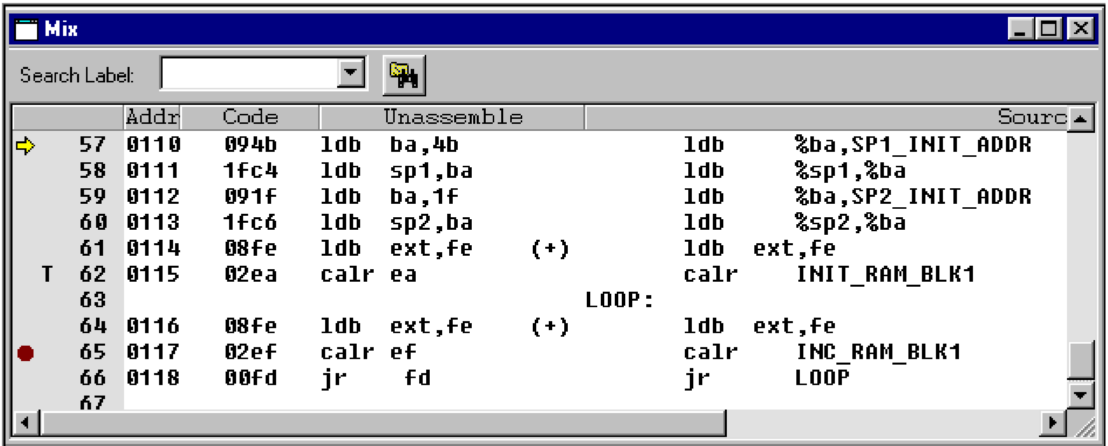
In this mode, the debugger displays the program codes after unassembling into mnemonics.

(2) Source mode



In this mode, the source that contains the code at the current PC address is displayed like an editor screen. This mode is available only when an absolute object file that contains source debugging information has been loaded.

(3) Mix mode



In this mode, both unassembled codes and sources are displayed like an absolute list. This mode is available only when an absolute object file that contains source debugging information has been loaded.

Refer to Section 8.4.3, "[Source] Window" for details about the display contents.

Symbol reference

When debugging a program after reading an object file in IEEE-695 format, the symbols defined in the source file can be used to specify an address. This feature can be used when entering a command having <address> in its parameter from the [Command] window or a dialog box.

(1) Referencing global symbols

Follow the method below to specify a symbol that is declared to be a global symbol/label by the .global or .comm pseudo-instruction.

@<symbol>

Example of specification:

```
>m @BOOT
>de @RAM_BLK1
```

(2) Referencing local symbols

Follow the method below to specify a local symbol/label that is used in only the defined source file.

@<symbol>@<file name>

The file name here is the source file name (.s) in which the symbol is defined.

Example of specification:

```
>bp @SUB1@test.s
```

(3) Displaying symbol list

All symbols used in the program and the defined addresses can be displayed in the [Command] window.

Table 8.8.2.2 Command to display symbol list

Function	Command
Displaying symbol list	**sy**

8.8.3 Displaying and Modifying Program, Data, Option Data and Register

The debugger has functions to operate on the program memory, data memory, and registers, as well as option data. Each memory area is set to the debugger according to the map information that is given in a parameter file.

Operating on program memory area

The following operations can be performed on the program memory area:

Table 8.8.3.1 Commands to operate on program memory

Function	Command
Entering/modifying program code	**pe**
In-line assemble	**a (as)**
Rewriting specified area	**pf**
Copying specified area	**pm**

(1) Entering/modifying program code

The program code at a specified address is modified by entering hexadecimal data.

(2) In-line assemble

The program code at a specified address is modified by entering a mnemonic code.

(3) Rewriting specified area

An entire specified area is rewritten with specified code.

(4) Copying specified area

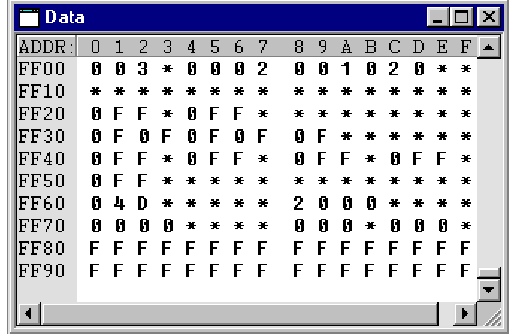
The content of a specified area is copied to another area.

Operating on data memory area

The following operations can be performed on the data memory areas (RAM, data ROM, display memory, I/O memory):

Table 8.8.3.2 Commands/menu item to operate on data memory

Function	Command	Menu
Dumping data memory	**dd**	[View | Data Dump]
Entering/modifying data	**de**	–
Rewriting specified area	**df**	–
Copying specified area	**dm**	–



(1) Dumping data memory

The contents of the data memory are displayed in hexadecimal dump format. If the [Data] window is opened, the contents of the [Data] window are updated; if not, the contents of the data memory are displayed in the [Command] window.

(2) Entering/modifying data

Data at a specified address is rewritten by entering hexadecimal data. Data can be directly modified on the [Data] window.

(3) Rewriting specified area

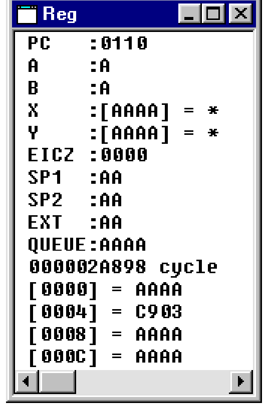
An entire specified area is rewritten with specified data.

(4) Copying specified area

The content of a specified area is copied to another area.

(5) Monitoring memory

Four memory locations, each with area to store 4 consecutive words, can be registered as watch data addresses. The registered watch data can be verified in the [Register] window. The content of this window is updated in real time at 0.5-second intervals by the on-the-fly function. Addresses 0, 4, 8, and C are made the watch data addresses by default.



The memory content displayed at the left indicates data at a specified address, and the one displayed at the right indicates 4-word data at the high-order address.

Monitor data

Operating registers

The following operations can be performed on registers:

Table 8.8.3.3 Commands/menu items to operate registers

Function	Command	Menu
Displaying registers	**rd**	[View | Register]
Modifying register values	**rs**	–

```
Reg
PC    :0110
A     :A
B     :A
X     :[AAAA] = *
Y     :[AAAA] = *
EICZ  :0000
SP1   :AA
SP2   :AA
EXT   :AA
QUEUE:AAAA
000002A898 cycle
[0000] = AAAA
[0004] = C903
[0008] = AAAA
[000C] = AAAA
```

(1) Displaying registers

Register contents can be displayed in the [Register] or [Command] window.
Registers: PC, A, B, X and [X], Y and [Y], F, SP1, SP2, EXT, and QUEUE

While the program is being executed, the PC address and F register are updated in real time every 0.5 seconds by the on-the-fly function.

(2) Modifying register values

The contents of the above registers can be set to any desired value.
The register values can be directly modified on the [Register] window.

Displaying option data

Option data in the ICE63's option areas (function option data, segment option data, or melody data).
Data is displayed in the [Command] window in hexadecimal dump format.

Table 8.8.3.4 Command to display option data

Function	Command
Displaying option data	**od**

8.8.4 Executing Program

The debugger can execute the target program successively or execute instructions one step at a time (single-stepping).

Successive execution

(1) Types of successive execution

There are two types of successive execution available:

- Successive execution from the current PC
- Successive execution from the program start address (0x0110) after resetting the CPU

Table 8.8.4.1 Commands/menu items/tool bar buttons for successive execution

Function	Command	Menu	Button
Successive execution from current PC	**g**	[Run | Go]	
		[Run | Go to Cursor]	
Successive execution after resetting CPU	**gr**	[Run | Go from Reset]	

(2) Stopping successive execution

Using the successive execution command (*g*), can specify up to two temporary break addresses that are only effective during program execution.
The temporary break address can also be specified from the [Source] window (one location only).
If the cursor is placed on an address line in the [Source] window and the [Go to Cursor] button clicked, the program starts executing from the current PC address and breaks before executing the instruction at the address the cursor is placed.

Except being stopped by this temporary break, the program continues execution until it is stopped by one of the following causes:

- Break conditions set by a break set up command are met.
- The [Key Break] button is clicked or the [Esc] key is pressed.
- A map break, etc. occurs.

[Key Break] button ∗ When the program does not stop, use this button to forcibly stop it.

(3) On-the-fly function

The ICE63 and debugger provide the on-the-fly function to display the PC address, F register and watch data values every 0.5 seconds (default) during successive execution. These contents are displayed in the relevant positions of the [Register] window. If the [Register] window is closed, they are displayed in the [Command] window. In the initial debugger settings, the display update interval of the on-the-fly function is set to twice per second. It can be modified to 0 (OFF)–5 (times) per second using the ***md*** command. This function provides a complete real-time display that is implemented using the ICE63 hardware.

Single-stepping

(1) Types of single-stepping

There are two types of single-stepping available:

- Stepping through all instructions (STEP)
  All instructions are executed one step at a time according to the PC, regardless of the type of instruction.
- Stepping through instructions except subroutines (NEXT)
  The calr, calz and int instructions are executed under the assumption that one step constitutes the range of statements until control is returned to the next step by a return instruction. Other instructions are executed in the same way as in ordinary single-stepping.

In either case, the program starts executing from the current PC.

Table 8.8.4.2 Commands/menu items/tool bar buttons for single-stepping

Function	Command	Menu	Button
Stepping through all instructions	**s**	[Run | Step]	
Stepping through all instructions except subroutines	**n**	[Run | Next]	

When executing single-stepping by command input, the number of steps to be executed can be specified, up to 65,535 steps. When using menu commands or tool bar buttons, the program is executed one step at a time.
In the following cases, single-stepping is terminated before a specified number of steps is executed:
- When the [Key Break] button is clicked or the [Esc] key is pressed.
- When a map break or similar break occurs.

Single-stepping is not suspended by breaks set by the user such as a PC break or data break.

[Key Break] button ∗ When the program does not stop, use this button to forcibly stop it.

(2) Display during single-stepping

In the initial debugger settings, the display is updated as follows:
The display contents of the [Register] window are updated every step. If the [Register] window is closed, its contents are displayed in the [Command] window. This default display mode can be switched over by the ***md*** command so that the display contents are updated at only the last step in a specified number of steps.
The display of the [Source] and [Data] windows are updated after the specified number of step executions are completed.

(3) HALT and SLEEP states and interrupts

The CPU is placed in a standby mode when the halt or slp instruction is executed. An interrupt is required to cancel this mode.
The debugger has a mode to enable or disable an external interrupt for use in single-step operation.

Table 8.8.4.3 External interrupt modes

	Enable mode	Disable mode
External interrupt	Interrupt is processed.	Interrupt is not processed.
halt and slp instructions	Executed as the halt instruction. Processing is continued by an external interrupt or clicking on the [Key Break] button.	The halt and slp instructions are replaced with a nop instruction as the instruction is executed.

In the initial settings, the debugger is set to the interrupt disable mode. The interrupt enable mode can also be set by using the ***md*** command.

Measuring execution cycles/execution time

(1) Execution cycle counter and measurement mode

The ICE63 contains a 31-bit execution cycle counter allowing you to measure the program execution time or the number of bus cycles executed. The measurement mode (time or bus cycle) can be selected using the ***md*** command. In the initial debugger settings, the bus cycle mode is selected.
The following lists the maximum values that can be measured by the execution cycle counter:

Execution time mode:	2,147,483,647 µsec = approx. 36 min.	(error = ±1 µsec)
Bus cycle mode:	2,147,483,647 cycles	(error = ±0)

(2) Displaying measurement results

The measurement result is displayed in the [Register] window. This display is cleared during program execution and is updated after completion of execution. If the [Register] window is closed, the measurement result can be displayed in the [Command] window using the ***rd*** command. The execution results of single-stepping are also displayed here.
If the counter's maximum count is exceeded, the system indicates "over flow".

(3) Hold mode and reset mode

In the initial debugger settings, the execution cycle counter is set to hold mode. In this mode, the measured values are combined until the counter is reset.
The reset mode can be set by the ***md*** command. In this mode, the counter is reset each time the program is executed. In successive execution, the counter is reset when the program is made to start executing by entering the ***g*** command and measurement is taken until the execution is terminated (beak occurs). (The same applies for the ***gr*** command except that the counter is reset simultaneously when the CPU is reset. Consequently, the counter operates the same way in both hold and reset modes.)
In single-stepping, the counter is reset when the program is made to start executing by entering the ***s*** or ***n*** command and measurement is taken until execution of a specified number of steps is completed. The counter is reset every step if execution of only one step is specified or execution is initiated by a tool bar button or menu command.

(4) Resetting execution cycle counter

The execution cycle counter is reset in the following cases:

- When the CPU is reset with the ***rst*** command, [Reset] in the [Run] menu, or the [Reset] button
- When the ***gr*** command or [Go from Reset] in the [Run] menu is executed
- When the execution cycle counter mode is switched over by the ***md*** command (between execution time and bus cycle modes or between hold and reset modes)
- When program execution is started in reset mode

Resetting the CPU

The CPU is reset when the ***gr*** command is executed, or by executing the ***rst*** command.
When the CPU is reset, the internal circuits are initialized as follows:

(1) Internal registers of the CPU

PC	... 0x0110
A, B	... 0xa
X, Y, QUEUE	... 0xaaaa
F	... 0b0000
SP1, SP2, EXT	... 0xaa

(2) The execution cycle counter is reset to 0.

(3) The [Source] and [Register] windows are redisplayed.

Because the PC is set to 0x0110, the [Source] window is redisplayed beginning with that address.
The [Register] window is redisplayed with the internal circuits initialized as described above.

The data memory contents are not modified.

8.8.5 Break Functions

The target program is made to stop executing by one of the following causes:
- Break command conditions are satisfied.
- The [Key Break] button is activated.
- The ICE63's BRKIN pin is pulled low.
- A map break or similar break occurs.

Break by command

The debugger has five types of break functions that allow the break conditions to be set by a command. When the set conditions in one of these break functions are met, the program under execution is made to break.

(1) Break by PC

This function causes the program to break when the PC matches the set address. The program is made to break before executing the instruction at that address. The PC breakpoints can be set for multiple addresses.

Table 8.8.5.1 Commands/menu items/tool bar button to set breakpoints

Function	Command	Menu	Button
Set breakpoints	**bp**	[Break | Breakpoint Set...]	
Clear breakpoints	**bc (bpc)**	[Break | Breakpoint Set...]	

The addresses that are set as PC breakpoints are marked with a ● as they are displayed in the [Source] window.
Using the [Break] button easily allows the setting and canceling of breakpoints.
Click on the address line in the [Source] window at where the program break is desired (after moving the cursor to that position) and then click on the [Break] button. A ● mark will be placed at the beginning of the line indicating that a breakpoint has been set there, and the address is registered in the breakpoint list. Clicking on the line that begins with a ● and then the [Break] button cancels the breakpoint you have set, in which case the address is deleted from the breakpoint list.

∗ The temporary break addresses that can be specified by the successive execution commands (*g*) do not affect the set addresses in the breakpoint list.

(2) Data break

This break function allows a break to be executed when a location in the specified data memory area is accessed. In addition to specifying a memory area in which to watch accesses, specification as to whether the break is to be caused by a read or write, as well as specification of the content of the data read or written. The read/write condition can be masked, so that a break will be generated for whichever operation, read or write, is attempted. Similarly, the data condition can also be masked in bit units. A break occurs after completing the cycle in which an operation to satisfy the above specified condition is performed.

Table 8.8.5.2 Commands/menu item to set data break

Function	Command	Menu
Set data break condition	**bd**	[Break | Data Break...]
Clear data break condition	**bdc**	[Break | Data Break...]

For example, if the program is executed after setting the data break condition as Address = 0x10, Data pattern = ∗ (mask) and R/W = W, the program breaks after writing any data to the data memory address 0x10.

(3) Register break

This break function causes a break when the A, B, F, X, and Y register reach a specified value. Each register can be masked (so they are not included in break conditions). The F register can be masked in bit units. A break occurs when the above registers are modified to satisfy all set conditions.

Table 8.8.5.3 Commands/menu item to set register break

Function	Command	Menu
Set register break conditions	**br**	[Break | Register Break...]
Clear register break conditions	**brc**	[Break | Register Break...]

For example, if the program is executed after setting 0 for the data of the A register and "**1*" for the data of the F register (C flag = 1) and masking all others, the program breaks when the A register is cleared to 0 and the C flag is set to 1.

(4) Sequential break

This break function allows settings of up to three break addresses and the number of times the instructions of the last address to be executed. While passing through all addresses sequentially in the order set, the program executes instructions at the final specified address the directed number of times, and then fetches the instruction at that address one more time before it breaks.

Table 8.8.5.4 Commands/menu item to set sequential break

Function	Command	Menu
Set sequential break conditions	**bs**	[Break | Sequential Break...]
Clear sequential break conditions	**bsc**	[Break | Sequential Break...]

For example, if you execute the program after first setting a break address in two locations at addresses 0x1000 and 0x2000 and specifying 3 for the execution count using the ***bs*** command, the program executes address 0x2000 three times after executing address 0x1000 more than one time, and when the PC reaches 0x2000, it breaks before performing the 4th execution.
The execution count can be set up to 4,095.

(5) Accessing outside stack area

In this case, a break occurs when a location outside the stack area is accessed by stack pointer SP1 or SP2.
Before this function can be used, the SP1 and SP2 areas must be set by the ***bsp*** command. The initial value is 0x0 to 0x3ff for SP1, and 0x0 to 0xff for SP2. The address of SP1 must be specified in units of 4 words.

Table 8.8.5.5 Command/menu item to set stack break

Function	Command	Menu
Set stack break conditions	**bsp**	[Break | Stack Break...]

Forced break by the [Key Break] button or the [Esc] key

The [Key Break] button or the [Esc] key can be used to forcibly terminate the program under execution when the program has fallen into an endless loop or cannot exit a standby (HALT or SLEEP) state.

[Key Break] button

Pulling ICE63's BRKIN pin low

The program is made to break by pulling the ICE63's BRKIN pin low (by applying a low-level pulse for more than 20 ns).

Map break and illegal instruction break

The program also breaks when one of the following errors is encountered during program execution:

(1) Access to undefined program area

A break occurs when an undefined area of the program memory map is accessed.

(2) Access to undefined data area

A break occurs when an undefined area of the data memory map is accessed.

(3) Write to data ROM area

A break occurs when a write to the data ROM area is attempted.

Notes:

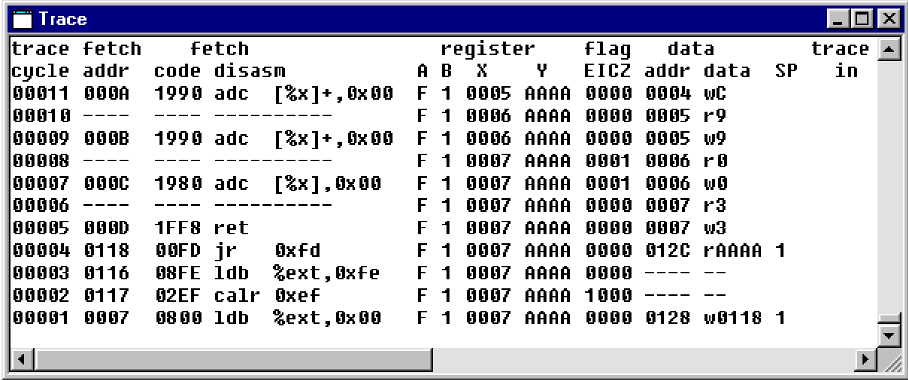
- *If the return address is popped from the stack by a ret or reti instruction in an area with prohibited 16-bit access, invalid data is read out from a 16-bit data bus that does not have any memory connected. In the ICE63, because the bus is pulled up, 0xffff is read out, causing control to return to that address. This could result in generating a map break.*
- *A break caused by an undefined program area access occurs before execution of such operation. On the other hand, a map break caused by access to an undefined data area or a write to the data ROM area occurs one or two instructions after execution of such operation.*
- *In user breaks based on command settings also, a PC break and sequential break occur before execution of operation. However, other breaks such as a data break, register break, and stack break occur one or two instructions after execution of operation.*

8.8.6 Trace Functions

The debugger has a function to trace program execution.

Trace memory and trace information

The ICE63 contains a trace memory. When the program executes instructions in the trace range according to the trace mode, the trace information on each cycle is taken into this memory. The trace memory has the capacity to store information for 8,192 cycles, making it possible to trace up to 4,096 instructions (for two-clock instructions only). When the trace information exceeds this capacity, the data is overwritten, the oldest data first unless operating in single-delay trigger mode. Consequently, the trace information stored in the trace memory is always within 8,192 cycles. The trace memory is cleared when a program is executed, starting to trace the new execution data.



The following lists the trace information that is taken into the trace memory in every cycle. This list is corresponded to display in the [Trace] window.

trace cycle: Trace cycle (decimal). The last information taken into the trace memory becomes 00001.

fetch addr: Fetch address (hexadecimal).

fetch code disasm: Fetch code (hexadecimal) and disassembled content.

register: Values of A, B, X, and Y registers after cycle execution (hexadecimal).

flag: States of E, I, C, and Z flags after cycle execution (binary).

data: Accessed data memory address (hexadecimal), read/write (denoted by r or w at the beginning of data), and data (1-digit hexadecimal for 4-bit access; 4-digit hexadecimal for 16-bit access).

SP: Stack access (1 for SP1 access; 2 for SP2 access).

trace in: Input to TRCIN pin (denoted by L when low-level signal is input).

Notes: The E0C63000 CPU uses two-stage pipelined instruction processing, one for fetch and one for execution. Therefore, please pay attention to the following:

- *The CPU fetches the next instruction in the last execution cycle of an instruction. Because the instruction is executed beginning from the cycle which is after the fetch, the displayed states of the registers, etc. are not the execution results of the fetch instruction that is displayed on the same line.*

- *For reasons of the ICE63's operation timing, the trace data at the boundary of operations, such as in the fetch cycle at which trace starts or the execution cycle at which trace ends, will not always be stored in memory.*

Trace modes

Three trace modes are available, depending on the method for sampling trace information.

Table 8.8.6.1 Trace mode setup command

Function	Command	Menu
Set trace mode	**tm**	[Trace | Trace Mode Set...]

(1) Normal trace mode

In this mode, the trace information on all bus cycles is taken into the trace memory during program execution. Therefore, until a break occurs, the trace memory always contains the latest information on bus cycles up to the one that is executed immediately beforehand.

(2) Single delay trigger trace mode

In this mode as in other modes, trace is initiated by a start of program execution. When the address (trace trigger point) that is set by the ***tm*** command is executed, trace is performed beginning from that point before being halted according to the next setting, which is also set by the command.

- If the trace trigger point is set to "start"
  Trance is halted after sampling trace information for 8,192 cycles beginning from the trace trigger point. In this case, the trace information at the trace trigger point is the oldest information stored in the trace memory.
  If the program stops before tracing all 8,192 cycles, trace information on some cycles preceding the trace trigger point may be left in the trace memory within its capacity.

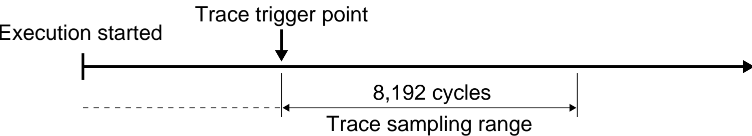


Fig. 8.8.6.1 Trace range when "start" is selected

- If the trace trigger point is set to "middle"
  Trace is halted after sampling trace information for 4,096 cycles beginning from the trace trigger point. In this case, the trace information of 4,096 cycles before and after the trace trigger point are sampled into the trace memory.
  If the program stops before tracing all 4,096 cycles, trace information for the location 4,096 cycles before the trace trigger point may be left in the trace memory, according to its capacity.

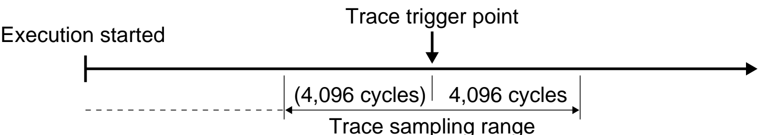


Fig. 8.8.6.2 Trace range when "middle" is selected

- If the trace trigger point is set to "end"
  Trace is halted after sampling trace information at the trace trigger point. In this case, the trace information at the trace trigger point is the latest information stored in the trace memory.
  If the program stops before tracing the trace trigger point, the system operates in the same way as in normal mode.

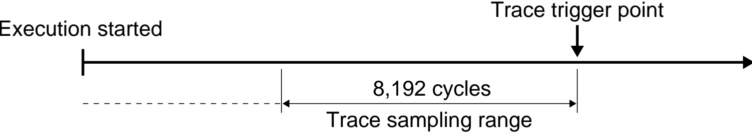


Fig. 8.8.6.3 Trace range when "end" is selected

If the program is halted in the middle of single delay trigger trace, bus cycles are traced from the beginning when trace is executed next.

(3) Address-area trace

In this mode, trace information is taken into the trace memory only when instructions within (or outside) a specified address range are executed. This address range can be set in up to four locations by the ***tm*** command. Whether you want trace to be performed within or outside that address range can also be specified by a command.

* ***Trace trigger address***

The ***tm*** command sets a trace trigger address regardless of the trace mode specified. When the [Source] window is open, the address thus set is marked by a "T" at the beginning of the address. When the program executes that address, the ICE63 outputs a low-level pulse from its TRGOUT pin.

Displaying and searching trace information

The sampled trace information can be displayed in the [Trace] window by a command. If the [Trace] window is closed, the information is displayed in the [Command] window. In the [Trace] window, the entire trace memory data can be seen by scrolling the window. The trace information can be displayed beginning from a specified cycle.
The display contents are as described above.

Table 8.8.6.2 Command/menu item to display trace information

Function	Command	Menu
Display trace information	**td**	[View | Trace]

It is possible to specify a search condition and display the trace information that matches a specified condition.
The search condition can be selected from the following three:
1. Program's execution address
2. Address from which data is read
3. Address to which data is written

When the above condition and one address are specified, the system starts searching. When the trace information that matches the specified condition is found, the system displays the found data in the [Trace] window (or in the [Command] window if the [Trace] window is closed).

Table 8.8.6.3 Command/menu item to search trace information

Function	Command	Menu
Search trace information	**ts**	[Trace | Trace Search...]

Saving trace information

After the trace information is displayed in the [Trace] window using the ***td*** or ***ts*** commands, the trace information within the specified range can be saved to a file.

Table 8.8.6.4 Command/menu item to save trace information

Function	Command	Menu
Save trace information	**tf**	[Trace | Trace File...]

8.8.7 Operation of Flash Memory

The ICE63 in-circuit emulator contains flash memory. This memory is designed to allow data to be transferred to and from the ICE63's emulation memory and the target memory by a command.
The flash memory retains data even when the ICE63 is turned off. By writing the program and/or data under debug into the flash memory before turning off the power, you can call it up and continue debugging next time. Also, even when operating the ICE63 in free-run mode (in which a program is executed using only the ICE63), you may need to write the program into the flash memory.

The following operations can be performed on the flash memory:

(1) Read from flash memory
Data is loaded from the flash memory into the emulation and/or target memory.

(2) Write to flash memory
Data in the emulation and/or target memory is saved to the flash memory. Also, the contents of the parameter file can be written to the flash memory as necessary. After writing to the flash memory in this way, you can protect it against read and write.

(3) Erasing flash memory
All contents of the flash memory are erased.

Table 8.8.7.1 Commands to operate on flash memory

Function	Command	Menu
Read from flash memory	**lfl**	[File | Flash Memory Operation...]
Write to flash memory	**sfl**	[File | Flash Memory Operation...]
Erase flash memory	**efl**	[File | Flash Memory Operation...]

*Note: Unless the contents of the parameter file that is specified when invoking Debugger db63 match the contents of parameters in the flash memory, neither write (**sfl**) nor read (**lfl**) to and from the flash memory can be performed. After you have received the shipment of the ICE63, erased the flash memory, or used a different parameter file (designed for some other microcomputer model in the E0C63 Family), be sure to write the contents of your parameter file along with other data into the flash memory using the **sfl** command.*

∗ *Free-run of ICE63*

When operating the ICE63 in free-run mode (with the program executed using only the ICE63), the ICE63 uses the data written in the flash memory. Therefore, before the ICE63 can be used in free-run mode, the entire program, data, and option data must be written into the flash memory.
To operate the ICE63 in free-run mode, set the ICE/RUN switch to the RUN position and turn on the power. During free-run, map breaks caused by operation in the program and data areas set by a parameter file are effective. When a map break occurs, the PC LED on the ICE63 stops and the EMU LED turns off. All other break settings are invalid because they cannot be written into the flash memory.

8.8.8 Coverage

The ICE63 retains coverage information (i.e., information on addresses at which a program is executed) and it can be displayed in the [Command] window.
Because the executed address range is displayed as shown below, it is possible to know which areas have not been executed.

```
Coverage Information:
    0: 0110..0118
    1: 0200..020f
```

Table 8.8.8.1 Coverage commands

Function	Command
Display coverage information	**cv**
Clear coverage information	**cvc**

8.9 Command Reference

8.9.1 Command List

Table 8.9.1.1 lists the debug commands available with the debugger.

Table 8.9.1.1 Command list

Classification	Command		Function	Page
Program memory operation	a / as	(assemble)	Assemble mnemonic	158
	pe	(program memory enter)	Input program code	160
	pf	(program memory fill)	Fill program area	161
	pm	(program memory move)	Copy program memory	162
Data memory operation	dd	(data memory dump)	Dump data memory	163
	de	(data memory enter)	Input data	165
	df	(data memory fill)	Fill data area	167
	dm	(data memory move)	Copy data area	168
	dw	(data memory watch)	Set watch data address	169
Option information	od	(option data dump)	Dump option data	171
Register operation	rd	(register display)	Display register values	173
	rs	(register set)	Modify register values	174
Program execution	g	(go)	Execute successively	176
	gr	(go after reset CPU)	Reset CPU and execute successively	178
	s	(step)	Step into	179
	n	(next)	Step over	181
CPU reset	rst	(reset CPU)	Reset CPU	182
Break	bp	(breakpoint set)	Set breakpoint	183
	bc / bpc	(breakpoint clear)	Clear breakpoint	185
	bd	(data break)	Set data break	186
	bdc	(data break clear)	Clear data break	188
	br	(register break)	Set register break	189
	brc	(register break clear)	Clear register break	191
	bs	(sequential break)	Set sequential break	192
	bsc	(sequential break clear)	Clear sequential break	194
	bsp	(break stack pointer)	Specify stack area (for illegal stack access detection)	195
	bl	(breakpoint list)	Display all break conditions	197
	bac	(break all clear)	Clear all break conditions	198
Program display	u	(unassemble)	Unassemble display	199
	sc	(source code)	Source display	201
	m	(mix)	Mix display	203
Symbol information	sy	(symbol list)	List symbols	205
Load file	lf	(load file)	Load IEEE-695 format absolute object file	206
	lo	(load option)	Load Motorola-S format file	207
Flash memory operation	lfl	(load from flash memory)	Read from flash memory	208
	sfl	(save to flash memory)	Write to flash memory	210
	efl	(erase flash memory)	Erase flash memory	212
Trace	tm	(trace mode)	Set trace mode	213
	td	(trace data display)	Display trace information	215
	ts	(trace search)	Search trace information	218
	tf	(trace file)	Save trace information into a file	220
Coverage	cv	(coverage)	Display coverage information	221
	cvc	(coverage clear)	Clear coverage information	222
Command file	com	(execute command file)	Load & execute command file	223
	cmw	(execute command file with wait)	Load/execute command file with execution intervals	224
	rec	(record commands to file)	Record commands to a command file	225
Log	log	(log)	Turn log output on or off	226
Map information	ma	(map information)	Display map information	227
Mode setting	md	(mode)	Set mode	228
Quit	q	(quit)	Quit debugger	231
Help	?	(help)	Display command usage	232

8.9.2 Reference for Each Command

The following sections explain all the commands by functions.

The explanations contain the following items.

Function

Indicates the functions of the command.

Format

Indicates the keyboard input format and parameters required for execution.

Example

Indicates a sample execution of the command.

Note

Shows notes on using.

GUI utility

Indicates a menu item or tool bar button if they are available for the command.

Notes:
- *In the command format description, the parameters enclosed by < > indicate they are necessary parameters that must be input by the user; while the ones enclosed by [ ] indicate they are optional parameters.*
- *The input commands are case-insensitive, you can use either upper case or lower case letters or even mixed.*
- *An error results if the number of parameters is not correct when you input a command using direct input mode.*

```
Error : Incorrect number of parameters
```

8.9.3 Program Memory Operation

a / as (assemble mnemonic)

Function

This command assembles the input mnemonic and rewrites the corresponding code to the program memory at the specified address.

Format

(1) >a <address> <mnemonic> [<file name>]↵ **(direct input mode)**

(2) >a [<address>]↵ **(guidance mode)**
Start address ? : <address>↵ ... Displayed only when <address> is omitted.
Address Original code Original mnemonic **: <mnemonic>↵**
....
>

<address>: Start address from which to write code; hexadecimal or symbol (IEEE-695 format only)
<mnemonic>: Input mnemonic; valid mnemonic of E0C63000 (expression and symbols are supported)
<file name>: File in which the symbol used in the operand was defined.
Condition: 0 ≤ address ≤ last program memory address

Examples

Format (1)

```
>a 200 "ld %a,f"↵          ... Assembles "LD  %A,0xF" and rewrites the code at address 0x200.
```

Format (2)

```
>a↵
Start address ? 200↵                              ... Address is input.
0200  1ff6  ld   %a,%f  : add %a,%b↵              ... Mnemonic is input.
Source file name (enter to ignore)   ?↵           ... Ignored *
0201  1fff  *nop  : ^↵                            ... Returned to previous address.
0200  1972  add  %a,%b  : ↵                       ... Input is skipped.
0201  1fff  *nop  : q↵                            ... Command is terminated.
>
```

* Source file name should be entered when a symbol/label is used as the operand. Specify the source file name in which the symbol was defined.

```
0200  1972  add  %a,%b  : jr LOOP↵                ... Symbol is used.
Source file name (enter to ignore)   ? main.s↵    ... Source file name is input.
```

Notes

- The ***a*** and ***as*** commands have the same function.
- The start address you specified must be within the range of the program memory area available with each microcomputer model.
  An error results if the input one is not a hexadecimal number or not a valid symbol.
  `Error : invalid value     (no such symbol / symbol type error)`
  An error results if the limit is exceeded.
  `Error : Address out of range, use 0-0xXXXX`
- An error results if the input mnemonic is invalid for E0C63000.
  `Error : illegal mnemonic`
- In guidance mode, the following keyboard inputs have special meaning:
  "q↵" … Command is terminated. (finish inputting and start execution)
  "^↵" … Return to previous address.
  "↵" … Input is skipped. (keep current value)
  If the maximum address of program memory is reached and gets a valid input other than "^↵", the command is terminated.
- When the contents of the program memory are modified using the ***a (as)*** command, the unassemble contents of the [Source] window are updated immediately.
- Although the contents of the unassemble display are modified by rewriting code, those of source display remain unchanged.

GUI utility

None

pe (program memory enter)

Function

This command rewrites the contents of the specified address in the program memory with the input hexadecimal code.

Format

(1) >pe <address> <code1> [<code2> [...<code8>]]↵ **(direct input mode)**

(2) >pe [<address>]↵ **(guidance mode)**
Program enter address ? <address>↵ ... Displayed only when <address> is omitted.
Address Original code **: <code>↵**
....
>

<address>: Start address from which to write code; hexadecimal or symbol (IEEE-695 format only)
<code(1–8)>: Write code; hexadecimal (valid operation code of E0C63000)
Condition: 0 ≤ address ≤ last program memory address, 0 ≤ input code ≤ 0x1fff

Examples

Format (1)

```
>pe 200 1972↵
```
... Rewrites the code at address 0x200 with 0x1972 (add %a, %b).

Format (2)

```
>pe↵
Program enter address ? 200↵      ... Address is input.
0200 1fff   : 1972↵              ... Code is input.
0201 1fff   : ↵                  ... Address 0x201 is skipped.
0202 1fff   : q↵                 ... Command is terminated.
>
```

Notes

- The start address you specified must be within the range of the program memory area available with each microcomputer model.
  An error results if the input one is not hexadecimal number or not a valid symbol.
  `Error : invalid value     (no such symbol / symbol type error)`
  An error results if the limit is exceeded.
  `Error : Address out of range, use 0-0xXXXX`
- Code must be input using a hexadecimal number in the range of 13 bits (0 to 0x1fff).
  An error results if the input one is not a hexadecimal number.
  `Error : invalid value`
  An error results if the input code exceeds the limit or it is invalidated in the .PAR file.
  `Error : illegal code`
- In guidance mode, the following keyboard inputs have special meaning:
  "q↵" … Command is terminated. (finish inputting and start execution)
  "^↵" … Return to previous address.
  "↵" … Input is skipped. (keep current value)
  If the maximum address of program memory is reached and gets a valid input other than "^↵", the command is terminated.
- When the contents of the program memory are modified using the *pe* command, the unassemble contents of the [Source] window are updated immediately.
- Although the contents of the unassemble display are modified by rewriting code, those of source display remain unchanged.

GUI utility

None

pf (program memory fill)

Function

This command rewrites the contents of the specified program memory area with the specified code.

Format

(1) >pf <address1> <address2> <code>↵ (direct input mode)
(2) >pf↵ (guidance mode)
Start address ? <address1>↵
End address ? <address2>↵
Fill code ? <code>↵
>

<address1>: Start address of specified range; hexadecimal or symbol (IEEE-695 format only)
<address2>: End address of specified range; hexadecimal or symbol (IEEE-695 format only)
<code>: Write code; hexadecimal (valid operation code of E0C63000)
Condition: 0 ≤ address1 ≤ address2 ≤ last program memory address, 0 ≤ code ≤ 0x1fff

Examples

Format (1)
```
>pf 200 20f 1ffe↵
```
... Fills the area from address 0x200 to address 0x20f with 0x1ffe (nop).

Format (2)
```
>pf↵
Start address ? 200↵        ... Start address is input.
End address ? 20f↵          ... End address is input.
Fill code ? 1fff↵           ... Code is input.
>
```
* Command execution can be canceled by entering only the [Enter] key and nothing else.

Notes

- The addresses specified here must be within the range of the program memory area available with each microcomputer model.
  An error results if the input one is not a hexadecimal number or not a valid symbol.
  ```
  Error : invalid value     (no such symbol / symbol type error)
  ```
  An error results if the limit is exceeded.
  ```
  Error : Address out of range, use 0-0xXXXX
  ```
- An error results if the start address is larger than the end address.
  ```
  Error : end address < start address
  ```
- When the contents of the program memory is modified using the *pf* command, the contents of the [Source] window are updated automatically.
- Although the contents of the unassemble display are modified by rewriting code, those of source display remain unchanged.

GUI utility

None

pm (program memory move)

Function

This command copies the content of a specified program memory area to another area.

Format

(1) >pm <address1> <address2> <address3>↵ **(direct input mode)**

(2) >pm↵ **(guidance mode)**
Start address ? <address1>↵
End address ? <address2>↵
Destination address ? <address3>↵
>

<address1>: Start address of source area to be copied from; hexadecimal or symbol (IEEE-695 format only)
<address2>: End address of source area to be copied from; hexadecimal or symbol (IEEE-695 format only)
<address3>: Address of destination area to be copied to; hexadecimal or symbol (IEEE-695 format only)
Condition: 0 ≤ address1 ≤ address2 ≤ last program memory address
0 ≤ address3 ≤ last program memory address

Examples

Format (1)

```
>pm 200 2ff 280↵          ... Copies the codes within the range from address 0x200 to address 0x2ff
                              to the area from address 0x280.
```

Format (2)

```
>pm↵
Start address ? 200↵          ... Source area start address is input.
End address ? 2ff↵            ... Source area end address is input.
Destination address ? 280↵    ... Destination area start address is input.
>
```

∗ Command execution can be canceled by entering only the [Enter] key and nothing else.

Notes

- The addresses you specified must be within the range of the program memory area available with each microcomputer model.
  An error results if the input one is not a hexadecimal number or not a valid symbol.
  `Error : invalid value     (no such symbol / symbol type error)`
  An error results if the limit is exceeded.
  `Error : Address out of range, use 0-0xXXXX`
- An error results if the start address is larger than the end address.
  `Error : end address < start address`
- When the contents of the program memory is modified using the ***pm*** command, the contents of the [Source] window are updated automatically.
- Although the contents of the unassemble display are modified by rewriting code, those of source display remain unchanged.

GUI utility

None

8.9.4 Data Memory Operation

dd (data memory dump)

Function

This command displays the content of the data memory in a 16 words/line hexadecimal dump format.

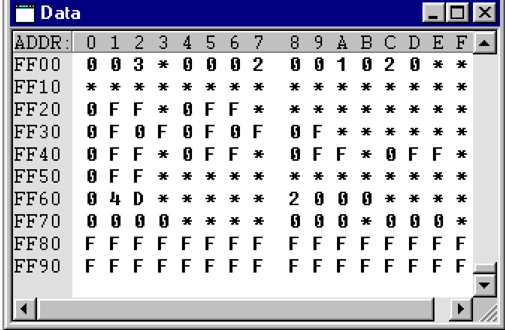
Format

>dd [<address1> [<address2>]]↵ (direct input mode)

<address1>: Start address to display; hexadecimal or symbol (IEEE-695 format only)
<address2>: End address to display; hexadecimal or symbol (IEEE-695 format only)
Condition: 0 ≤ address1 ≤ address2 ≤ 0xffff

Display

(1) When [data] window is opened



If both <address1> and <address2> are not defined, the [Data] window is redisplayed beginning with address 0x0000.
If <address1> is defined , or even <address2> is defined, the [Data] window is redisplayed in such a way that <address1> is displayed at the uppermost line.
Even when <address1> specifies somewhere in 16 addresses/line, data is displayed beginning with the top of that line. For example, even though you may have specified address 0xff08 for <address1>, data is displayed beginning with address 0xff00.
However, if an address near the uppermost part of data memory (e.g. maximum address is 0xffff), such as 0xffc0, is specified as <address1>, the last line displayed in the window in this case is 0xfff0, the specified address is not at the top of the window.
Since the [Data] window can be scrolled to show the entire data memory, defining <address2> does not have any specific effect. Only defining <address1> and both defining <address1> and <address2> has same display result.

(2) When [data] window is closed

If both <address1> and <address2> are not defined, the debugger displays data for 256 words from address 0x000 in the [Command] window.

```
>dd↵
       0 1 2 3 4 5 6 7  8 9 A B C D E F
0000:  A A A A D C 0 3  A A A A A A A A
0010:  A A A A A A A A  A A A A A A A A
  :          :                :
00E0:  A A A A A A A A  A A A A A A A A
00F0:  A A A A A A A A  A A A A A A A A
>
```

If only <address1> is defined, the debugger displays data for 256 words from <address1>.

```
>dd ff00↵
FF00:  0 0 3 * 0 0 0 2  0 0 1 0 2 0 * *
FF10:  * * * * * * * *  * * * * * * * *
 :          :                :
FFE0:  0 0 0 0 0 0 0 0  * * * * * * * *
FFF0:  0 0 0 0 0 0 0 0  * * * * * * * *
>
```

"*" indicates an unused address.

If both <address1> and <address2> are defined, the debugger displays data from <address1> to <address2>.

```
>dd 008 017↵
      0 1 2 3 4 5 6 7  8 9 A B C D E F
0000:                  0 0 0 0 0 0 0 0
0010: 0 0 0 0 0 0 0 0
>
```

(3) During log output

If a command execution is being output to a log file by the ***log*** command when you dump the data memory, data is displayed in the [Command] window even if the [Data] window is opened and are also output to the log file.
If the [Data] window is closed, data is displayed in the [Command] window in the same way as in (2) above.
If the [Data] window is open, it is redisplayed to show data in the same way as in (1) above. In this case, the same number of lines is displayed in the [Command] window as are displayed in the [Data] window.

(4) Successive display

Once you execute the ***dd*** command, data can be displayed successively with the [Enter] key only until some other command is executed.
When you hit the [Enter] key, the [Data] window is scrolled one full screen.
When displaying data in the [Command] window, data is displayed for the 16 lines following the previously displayed address (same number of lines as displayed in the [Data] window during log output).

```
>dd↵
      0 1 2 3 4 5 6 7  8 9 A B C D E F
0000: A A A A A A A A  A A A A A A A A
0010: A A A A A A A A  A A A A A A A A
 :          :                :
00F0: A A A A A A A A  A A A A A A A A
>↵
      0 1 2 3 4 5 6 7  8 9 A B C D E F
0100: A A A A A A A A  A A A A A A A A
0110: A A A A A A A A  A A A A A A A A
 :          :                :
01F0: A A A A A A A A  A A A A A A A A
>
```

When the line at address 0xfff0 is displayed, the system stands by waiting for command input. If you hit the [Enter] key here, data is displayed beginning with address 0x0000.

Notes

- Both the start and end addresses specified here must be within the range of the data memory area available with each microcomputer model.
  An error results if the input one is not a hexadecimal number or not a valid symbol.
  `Error : invalid value     (no such symbol / symbol type error)`
  An error results if the limit is exceeded.
  `Error : Address out of range, use 0-0xFFFF`
- An error results if the start address is larger than the end address.
  `Error : end address < start address`

GUI utility

[View | Data Dump] menu item

When this menu item is selected, the [Data] window opens or becomes active and displays the current data memory contents.

de (data memory enter)

Function

This command rewrites the contents of the data memory with the input hexadecimal data. Data can be written to continuous memory locations beginning with a specified address.

Format

(1) >de <address> <data1> [<data2> [...<data16>]]↵ **(direct input mode)**

(2) >de↵ **(guidance mode)**

Data enter address ? : <address>↵

Address Original data **: <data>↵**

....

>

<address>: Start address from which to write data; hexadecimal or symbol (IEEE-695 format only)

<data(1–16)>: Write data; hexadecimal

Condition: 0 ≤ address ≤ 0xffff, 0 ≤ data ≤ 0xf

Examples

Format (1)

```
>de 100 0↵              ... Rewrites data at address 0x100 with 0.
```

Format (2)

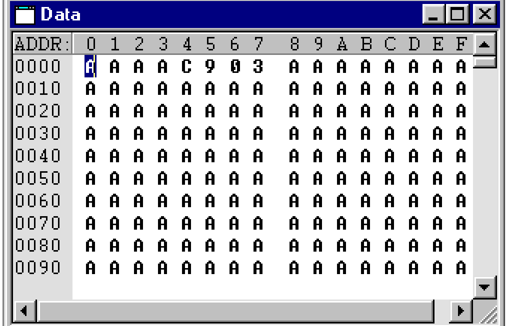
```
>de↵
Data enter address ? :100↵   ... Address is input.
0100   0 : a↵                ... Data is input.
0101   0 : ↵                 ... Skipped.
0102   0 : q↵                ... Command is terminated.
>
```

Notes

- The start address specified here must be within the range of the data memory area available with each microcomputer model.
  An error results if the input one is not a hexadecimal number or a valid symbol.
  `Error : invalid value     (no such symbol / symbol type error)`
  An error results if the limit is exceeded.
  `Error : Address out of range, use 0-0xFFFF`
- The contents of the unused area will be marked as "*". If you encounter any address marked by "*", press [Enter] key to skip that address or terminate the command.
- Data must be input using a hexadecimal number in the range of 4 bits (0 to 0xf). An error results if the limit is exceeded.
  `Error : Data out of range, use 0-0xF`
- When the contents of the data memory is modified using the ***de*** command, the displayed contents of the [Data] window are updated automatically.
- In guidance mode, the following keyboard inputs have special meaning:
  "q↵" … Command is terminated. (finish inputting and start execution)
  "^↵" … Return to previous address.
  "↵" … Input is skipped. (keep current value)
  If the maximum address of data memory is reached and gets a valid input other than "^↵", the command is terminated.

GUI utility

[Data] window



The [Data] window allows direct modification of data. Click the [Data] window and select the displayed data to be modified then enter a hexadecimal number.

df (data memory fill)

Function

This command rewrites the contents of the specified data memory area with the specified data.

Format

(1) >df <address1> <address2> <data> ↵ **(direct input mode)**

(2) >df↵ **(guidance mode)**

Start address ? <address1>↵
End address ? <address2>↵
Data pattern ? <data>↵
>

<address1>: Start address of specified range; hexadecimal or symbol (IEEE-695 format only)
<address2>: End address of specified range; hexadecimal or symbol (IEEE-695 format only)
<data>: Write data; hexadecimal
Condition: 0 ≤ address1 ≤ address2 ≤ 0xffff, 0 ≤ data ≤ 0xf

Examples

Format (1)

```
>df 200 2ff 0↵            ... Fills the data memory area from address 0x200 to address 0x2ff with 0x0.
```

Format (2)

```
>df↵
Start address ? 200↵      ... Start address is input.
End address ? 2ff↵        ... End address is input.
Data pattern ? 0↵         ... Data is input.
>
```

∗ Command execution can be canceled by entering only the [Enter] key and nothing else.

Notes

- Both the start and end addresses specified here must be within the range of the data memory area available with each microcomputer model.
  An error results if the input one is not a hexadecimal number or a valid symbol.
  `Error : invalid value     (no such symbol / symbol type error)`
  An error results if the limit is exceeded.
  `Error : Address out of range, use 0-0xFFFF`
- An error results if the start address is larger than the end address.
  `Error : end address < start address`
- Data must be input using a hexadecimal number in the range of 4 bits (0 to 0xf). An error results if the limit is exceeded.
  `Error : Data out of range, use 0-0xF`
- Write operation is not performed to the read only address of the I/O area.
- When there is an unused area in the specified address range, no error occurs. The area other than the unused area will be filled with the specified data.
- When the contents of the data memory is modified using the ***df*** command, the displayed contents of the [Data] window are updated automatically.

GUI utility

None

dm (data memory move)

Function

This command copies the contents of the specified data memory area to another area.

Format

(1) >dm <address1> <address2> <address3>↵ **(direct input mode)**

(2) >dm↵ **(guidance mode)**
Start address ? <address1>↵
End address ? <address2>↵
Destination address ? <address3>↵
>

<address1>: Start address of source area to be copied from; hexadecimal or symbol (IEEE-695 format only)
<address2>: End address of source area to be copied from; hexadecimal or symbol (IEEE-695 format only)
<address3>: Address of destination area to be copied to; hexadecimal or symbol (IEEE-695 format only)
Condition: $0 \leq$ address1 $\leq$ address2 $\leq$ 0xffff, $0 \leq$ address3 $\leq$ 0xffff

Examples

Format (1)

```
>dm 200 2ff 280↵
```
... Copies data within the range from address 0x200 to address 0x2ff to the area from address 0x280.

Format (2)

```
>dm↵
Start address ? 200↵          ... Source area start address is input.
End address ? 2ff↵            ... Source area end address is input.
Destination address 280↵      ... Destination area start address is input.
>
```

∗ Command execution can be canceled by entering only the [Enter] key and nothing else.

Notes

- All the addresses specified here must be within the range of the data memory area available with each microcomputer model.
  An error results if the input one is not a hexadecimal number or a valid symbol.
  `Error : invalid value     (no such symbol / symbol type error)`
  An error results if the limit is exceeded.
  `Error : Address out of range, use 0-0xFFFF`
- Write operation is not performed to the read-only address of the I/O area.
- Data in the write-only area cannot be read. If the source area contains write-only address, 0 is written to the corresponding destination. If the destination area contains read-only address, the data of that address can not be rewritten. If the source and destination areas contain I/O address of mixed read-only bits and write-only bits, either read or write operation can be executed for the corresponding bits.
- When the contents of the data memory is modified using the *dm* command, the displayed contents of the [Data] window are updated automatically.

GUI utility

None

dw (data memory watch)

Function

This command registers four data memory locations as the watch data addresses. Memory contents equivalent to 4 words at each watch address are displayed in the [Register] window.

Format

(1) >dw <address1> [... <address4>] ↵ **(direct input mode)**

(2) >dw↵ **(guidance mode)**
Address 1 = *Old value* **: <address1>↵**
Address 2 = *Old value* **: <address2>↵**
Address 3 = *Old value* **: <address3>↵**
Address 4 = *Old value* **: <address4>↵**
>

<address1–4>: Watch address; hexadecimal or symbol (IEEE-695 format only)
Condition: 0 ≤ address1 ≤ address2 ≤ 0xffff

Examples

Format (1)

```
>dw 10 14 18 1C↵
```

... Sets watch addresses to 0x10, 0x14, 0x18, and 0x1c.

Format (2)

```
>dw↵
Address1 = 0010 :0↵
Address2 = 0014 :4↵
Address3 = 0018 :8↵
Address4 = 001c :c↵
>
```

Notes

- When the debugger starts up, four locations at addresses 0, 4, 8, and 0xc are initially set as the watch data addresses.
- The address specified here must be within the range of the data memory area available with each microcomputer model.
  An error results if the input one is not a hexadecimal number or a valid symbol.
  `Error : invalid value     (no such symbol / symbol type error)`
  An error results if the limit is exceeded.
  `Error : Address out of range, use 0-0xFFFF`
- The watch data addresses are set in units of 4 words. A warning results if you specify an address that is outside the 4-word boundary, with your specified address rounded down to a multiple of 4.
  Example:

```
>dw↵
Address1 = 0000 :0↵
Address2 = 0004 :10↵
Address3 = 0008 :15↵        ... Illegal address
Address4 = 000c :19↵        ... Illegal address
Warning : round down to multiple of 4
Address1 =    0
Address2 =   10
Address3 =   14
>
```

- Be aware that a value is displayed as the watch data even if the invalid address, which is displayed as an "*" in the ***dd*** command, is registered. The value in this case is indeterminate.
- The value displayed to the left shows the content of the start address, and that displayed to the right is the content of an address that is equal to the start address + 3.

GUI utility

None

8.9.5 Command to Display Option Information

od (option data dump)

Function

This command displays option data in the [Command] window in a hexadecimal dump format after reading it from the ICE63.

Option data	Target memory address range
Function option data (fog)	0 to 0xef
Segment option data (sog)	0 to 0x1fff
Melody data (mla)	0 to 0xfff

Format

(1) >od <type> [<address1> [<address2>]]↵ (direct input mode)

(2) >od↵ (guidance mode)

1. fog 2. sog 3. mla . . . ? <type>↵
Start address ? <address1>↵
End address ? <address2>↵
Option data display . . . .
>

<type>: Option type; fog, sog, or mla
<address1>: Start address of specified range; hexadecimal
<address2>: End address of specified range; hexadecimal
Condition: 0 ≤ address1 ≤ address2 ≤ 0xef (fog), 0x1fff (sog) or 0xfff (mla)

Examples

Format (1)

```
>od fog 0 f↵                ... Displays function option data within the range of  0 to 0xf.
      0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F
0000: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
```

Format (2)

```
>od
1.fog 2.sog 3.mla ...? 1↵     ... Function option is selected.
Start address ? 10↵           ... Start address is input.
End address ? 1f↵             ... End address is input.
      0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F
0010: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
>
```

Notes

- The start and end addresses can be omitted by entering the [Enter] key only.
  If the start address is omitted, data is displayed beginning with address 0.
  If the end address is omitted, data is displayed for up to 16 lines within the range of the option area.
- Data in unused areas is marked by an "*" as it is displayed in the window.
- The maximum number of lines that can be displayed at once is 16 (fog data is limited to 15 lines).
  Even if you specify the end address in an attempt to display more than 16 lines, the system will only display data for 16 lines and then stand by waiting for a command input. As with the ***dd*** command, this command allows you to display data for the following addresses by entering the [Enter] key only. (The maximam number of lines is 16.)

- Both the start and end addresses must be specified within the setup range of each option. An error results if this limit is exceeded.
  `Error : FO address out of range, use 0-0xEF`
  ... Specified address for the function option is outside the range.
  `Error : SO address out of range, use 0-0x1FFF`
  ... Specified address for the segment option is outside the range.
  `Error : MLA address out of range, use 0-0xFFF`
  ... Specified address for the melody data is outside the range.
- An error results if the start address is larger than the end address.
  `Error : end address < start address`
- The default value of option data is 0.

GUI utility

None

8.9.6 Register Operation

rd (register display)

Function

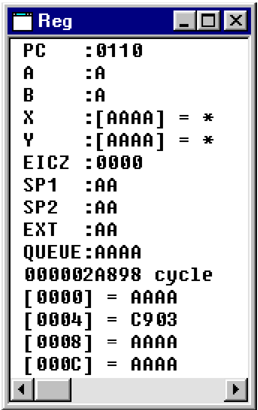
This command displays the contents of the registers, execution cycle counter, and watch data.

Format

>rd↵ **(direct input mode)**

Display

(1) Contents of display



The following lists the contents displayed by this command.

PC: Program counter
A: A register
B: B register
X: Contents of X register and indirectly addressed data memory
Y: Contents of Y register and indirectly addressed data memory
EICZ: Flags
SP1: Stack pointer SP1
SP2: Stack pointer SP2
EXT: EXT register
QUEUE: QUEUE register
bus cycle: Execution cycle counter
[xxxx]: Watch data at four locations

∗ If the memory locations indicated by the X and Y registers are in an unused area, the data in that area is marked by an "∗" as it is displayed.
Note that watch data is always displayed even if it resides in an unused area (indeterminate).

(2) When [Register] window is opened

When the [Register] window is opened, all the above contents are displayed in the [Register] window according to the program execution. When you use the *rd* command, the displayed contents of the [Register] window is updated.

(3) When [Register] window is closed

Data is displayed in the [Command] window in the following manner:

```
>rd↵
PC:0110 A:A B:A X:[AAAA] = * Y:[AAAA] = * EICZ:0000 SP1:AA SP2:AA EXT:AA
QUEUE:AAAA  bus cycle:000002AB3D cycle
[0000] = 0000 [0010] = AAAA [0014] = AAAA [0018] = AAAA
>
```

(4) During log output

If a command execution result is being output to a log file by the ***log*** command, the register values are displayed in the [Command] window even if the [Register] window is opened and are also output to the log file.

GUI utility

[View | Register] menu item

When this menu item is selected, the [Register] window opens or becomes active and displays the current register contents.

rs (register set)

Function

This command modifies the register values.

Format

(1) >rs <register> <value> [<register> <value> [...<register> <value>]]↵ (direct input mode)

(2) >rs↵ (guidance mode)

```
  PC = Old value : <value>↵
   A = Old value : <value>↵
   B = Old value : <value>↵
   X = Old value : <value>↵
   Y = Old value : <value>↵
  FE = Old value : <value>↵
  F I = Old value : <value>↵
  FC = Old value : <value>↵
  FZ = Old value : <value>↵
 SP1 = Old value : <value>↵
 SP2 = Old value : <value>↵
 EXT = Old value : <value>↵
   Q = Old value : <value>↵
>
```

<register>: Register name (PC, A, B, X, Y, F, SP1, SP2, EXT, Q)

<value>: Value to be set to the register; hexadecimal

Examples

Format (1)

```
>rs pc 110 f 0↵
```

... Sets PC to 0x0110 and resets all the flags.

Format (2)

```
>rs↵
 PC= 116: 110↵
  A=   0: f↵
  B=   0: ↵
  X=   0: 100↵
  Y=   0: 100↵
 FE=   0: ↵
 FI=   0: ↵
 FC=   1: 0↵
 FZ=   1: 0↵
SP1=  aa: ff↵
SP2=  aa: ff↵
EXT=   0: ↵
  Q=   0: ↵
```

When a register is modified, the [Register] window is updated to show the contents you have input. If you input "q↵" to stop entering in the middle, the contents input up to that time are updated.

Notes

- An error results if you input a value exceeding the register's bit width.
```
Error : invalid value
```
- An error results if you input a register name other than PC, A, B, X, Y, F, SP1, SP2, EXT or Q in direct input mode.
```
Error : Incorrect register name, use PC/A/B/X/Y/F/SP1/SP2/EXT/Q
```
- In guidance mode, the following keyboard inputs have special meaning:
  "q↵" … Command is terminated. (finish inputting and start execution)
  "^↵" … Return to previous register.
  "↵" … Input is skipped. (keep current value)

GUI utility

[Register] window

The [Register] window allows direct modification of data. Click the [Register] window, select the displayed data to be modified and enter a value then press [Enter].

8.9.7 Program Execution

g (go)

Function

This command executes the target program from the current PC position.

Format

>g [<address1> [<address2>]]↵ (direct input mode)

<address1–2>: Temporary break addresses; hexadecimal or symbol (IEEE-695 format only)
Condition: 0 ≤ address1(2) ≤ last program memory address

Operation

(1) Program execution

The target program is executed from the address indicated by the PC. Program execution is continued until it is made to break for one of the following causes:

- The set break condition is met
- The [Key Break] button is clicked or the [Esc] key is pressed
- A map break, etc., occurs

If a temporary break is specified, the program execution will be suspended before executing the instruction at the specified address. Up to two temporary break addresses can be specified.

`>g 1a0↵` ... Executes the program from the current PC address to address 0x1a0.

When program execution breaks, the system stands by waiting for a command input after displaying a break status message. When you hit the [Enter] key here, program execution is resumed beginning with a PC address next to the break address. Temporary break address settings are also valid.

(2) Window display by program execution

In the initial debugger settings, the on-the-fly function is turned on.
During program execution, the PC, flags and watch data contents in the [Register] window are updated in real time every 0.5 seconds (default) by the on-the-fly function. If the [Register] window is closed, the above contents are displayed in the [Command] window. The on-the-fly function can be turned off by the ***otf*** command. In this case, the [Register] window is updated after a break.

The [Source] window is updated after a break in such a way that the break address is displayed within the window.

If the [Trace] window is opened, the display contents are cleared as the program is executed. It is updated with the new trace information after a break.

If the [Data] window is opened, the display contents are updated after a break.

(3) Display during log mode

If the program is executed after turning on the log mode, an on-the-fly display appears in the [Command] window as well as the [Register] window.

Example:

```
>g
PC:0007 EICZ:0001 [0000] = AAAA [0004] = 3D30 [0008] = AAAA [000C] = AAAA
PC:000C EICZ:0000 [0000] = AAAA [0004] = 5250 [0008] = AAAA [000C] = AAAA
PC:0117 EICZ:1001 [0000] = AAAA [0004] = 6760 [0008] = AAAA [000C] = AAAA
PC:000B EICZ:0000 [0000] = AAAA [0004] = 8C70 [0008] = AAAA [000C] = AAAA
Key Break
PC:0008 A:F B:1 X:[0007] = 0 Y:[AAAA] = * EICZ:1001 SP1:4A(128) SP2:1F EXT:00
QUEUE:0118  bus cycle:0000029332 cycle [0000] = AAAA [0004] = E280 [0008] = AAAA
[000C] = AAAA
>
```

When a break occurs, the same display appears as when data is displayed by the ***rd*** command.

(4) Execution cycle counter

The execution cycle counter displayed in the [Register] window indicates the number of cycles executed or the execution time of the target program. (Refer to Section 8.8.4 for details.)
In the initial debugger settings, the execution cycle counter is set to hold mode so that execution time is added up until the CPU is reset. If this mode is changed to reset mode by the ***md*** command, the execution cycle counter is cleared to 0 each time the *g* command is executed. The counter is also reset simultaneously when execution is restarted by hitting the [Enter] key.

Notes

- If a break condition is met, program execution is suspended and the PC will be set to the program address at the breakpoint.
- The address you specified must be within the range of the program memory area available with each microcomputer model.
  An error results if the input one is not a hexadecimal number or a valid symbol.
  ```
  Error : invalid value     (no such symbol / symbol type error)
  ```
  An error results if the limit is exceeded.
  ```
  Error : Address out of range, use 0-0xXXXX
  ```

GUI utility

[Run | Go] menu item, [Go] button

When this menu item or button is selected, the *g* command without temporary break is executed.

[Go] button

[Run | Go to Cursor] menu item, [Go to Cursor] button

When this menu item or button is selected after placing the cursor to the temporary break address line in the [Source] window, the *g* command with a temporary break is executed. The program execution will be suspended after executing the address at the cursor position.

[Go to Cursor] button

gr (go after reset CPU)

Function

This command executes the target program from the boot address after resetting the CPU.

Format

>gr [<address1> [<address2>]]↵ (direct input mode)

<address1–2>: Temporary break addresses; hexadecimal or symbol (IEEE-695 format only)
Condition: 0 ≤ address1(2) ≤ last program memory address

Operation

This command resets the CPU before executing the program. This causes the PC to be set at address 0x0110, from which the command starts executing the program.
Once the program starts executing, the command operates in the same way as the *g* command, except that the *gr* command does not support the function for restarting execution by hitting the [Enter] key. Refer to the explanation of the *g* command for more information.

Notes

- If a break condition is met, program execution is suspended and the PC will be set to the program address at the breakpoint.
- The address you specified must be within the range of the program memory area available with each microcomputer model.
  An error results if the input one is not a hexadecimal number or a valid symbol.
  `Error : invalid value     (no such symbol / symbol type error)`
  An error results if the limit is exceeded.
  `Error : Address out of range, use 0-0xXXXX`

GUI utility

[Run | Go from Reset] menu item, [Go from Reset] button

When this menu item or button is selected, the *gr* command is executed.

[Go from Reset] button

S (step)

Function

This command single-steps the target program from the current PC position by executing one instruction at a time.

Format

>s [<step>]↵ (direct input mode)

<step>: Number of steps to be executed; decimal (default is 1)
Condition: 0 ≤ step ≤ 65,535

Operation

(1) Step execution

If the <step> is omitted, only the program step at the address indicated by the PC is executed, otherwise the specified number of program steps is executed from the address indicated by the PC.

```
>s↵       ...Executes one step at the current PC address.
>s 20↵    ...Executes 20 steps from the current PC address.
```

The program execution is suspended by the following cause even before the specified number of steps is completed.

- The [Key Break] button is clicked or the [Esc] key is pressed
- A map break, etc. occurs

After each step is completed, the register contents in the [Register] window are updated. If the [Register] window is closed, the register contents are displayed in the [Command] window same as executing the ***rd*** command.
When program execution is completed by stepping through instructions, the system stands by waiting for command input. If you hit the [Enter] key here, the system single-steps the program in the same way again.

(2) HALT and SLEEP states and interrupts

When the halt or slp instruction is executed, the CPU is placed in standby mode. An interrupt is required to clear this mode. The debugger has a mode to enable or disable an external interrupt for use in a single-step operation.

	Enable mode	Disable mode
External interrupt	Interrupt is processed.	Interrupt is not processed.
halt and slp instructions	Executed as the halt instruction. Processing is continued by an external interrupt or clicking on the [Key Break] button.	The halt and slp instructions are replaced with a nop instruction as the instruction is executed.

In the initial settings, the debugger is set to the interrupt disable mode.
The interrupt enable mode can also be set by using the ***md*** command.

(3) Execution cycle counter

The execution cycle counter displayed in the [Register] window indicates the number of cycles executed or the execution time of the target program.
In the initial debugger settings, the execution cycle counter is set to hold mode so that execution time is added up until the CPU is reset. If this mode is changed to reset mode by the ***md*** command, the execution cycle counter is cleared to 0 each time the ***s*** command is executed. The counter is also reset simultaneously when execution is restarted by hitting the [Enter] key.

(4) During log mode

If the program is single-stepped after turning on the log mode, the same contents as when executing the ***rd*** command are displayed in the [Command] window after each step is completed.

Notes

- The step count must be specified within the range of 0 to 65,535. An error results if the limit is exceeded.
```
Error : Number of steps out of range, use 0-65535
```
- If the [Data] window is opened, its display contents are updated after the execution.
- During a single-step operation, the program will not break even if the break condition set by a command is met.
- Unlike in successive executions (***g*** or ***gr*** command), the [Register] window is updated every time a step is executed.

GUI utility

[Run | Step] menu item, [Step] button

When this menu item or button is selected, the ***s*** command without step count is executed.

[Step] button

n (next)

Function

This command single-steps the target program from the current PC position by executing one instruction at a time.

Format

>n [<step>]↵ **(direct input mode)**

<step>: Number of steps to be executed; decimal (default is 1)
Condition: 0 ≤ step ≤ 65,535

Operation

This command basically operates in the same way as the *s* command.
However, the calr, calz and int instructions, including all subroutines until control returns to the next address, are executed as one step.

Notes

- The step count must be specified within the range of 0 to 65,535. An error results if the limit is exceeded.
```
Error : Number of steps out of range, use 0-65535
```
- If the [Data] window is opened, its display contents are updated after the execution.
- During a single-step operation, the program will not break even if the break condition set by a command is met.
- Unlike in successive executions (*g* or *gr* command), the [Register] window is updated every time a step is executed.

GUI utility

[Run | Next] menu item, [Next] button
When this menu item or button is selected, the *n* command without step count is executed.

[Next] button

8.9.8 CPU Reset

rst (reset CPU)

Function

This command resets the CPU.

Format

>rst↵ **(direct input mode)**

Notes

- The registers and flags are set as follows:

  PC: 0110
  A: A
  B: A
  X: AAAA
  Y: AAAA
  EICZ: 0000
  SP1: AA
  SP2: AA
  EXT: AA
  QUEUE: AAAA

- The execution cycle counter is cleared to 0.
- If the [Source] window is open, the window is redisplayed beginning with address 0x0110. If the [Register] window is open, the window is redisplayed with the above contents.
- The debug status, such as memory contents, breaks, and trace, is not reset.

GUI utility

[Run | Reset CPU] menu item, [Reset] button

When this menu item or button is selected, the ***rst*** command is executed.

8.9.9 Break

bp (break point set)

Function

This command sets or clears breakpoints using a program's execution address.

Format

(1) >bp <break1> [<break2> [... <break16>]]↵ **(direct input mode)**

(2) >bp↵ **(guidance mode)**

PC break set status

1. set 2. clear 3. clear all ... ? <1 | 2 | 3>↵

.... (guidance depends on the above selection, see examples)

>

<break1–16>: Break address; hexadecimal or symbol (IEEE-695 format only)

Condition: 0 ≤ address ≤ last program memory address

Examples

Format (1)

```
>bp 116 200↵          ... Sets break points at addresses 0x0116 and 0x0200.
                          * The direct input mode cannot clear the set break points.
```

Format (2)

```
>bp↵    (Set)
No PC break is set.
1. set   2. clear   3. clear all ...? 1↵     ... "1. set" is selected.
Set break address ? : 116↵                   ... Address 0x0116 is set as a breakpoint.
Set break address ? : 200↵                   ... Address 0x0200 is set as a breakpoint.
Set break address ? : ↵                      ... Terminated by [Enter] key.
>bp↵    (Clear)
    1: 0116
    2: 0200
1. set   2. clear   3. clear all ...? 2↵     ... "2. clear" is selected.
Clear break address ? : 200↵                 ... Break address 0x0200 is cleared.
Clear break address ? : ↵                    ... Terminated by [Enter] key.
>bp↵    (Clear all)
    1: 0116
1. set   2. clear   3. clear all ...? 3↵     ... "3. clear all" is selected.
>bp↵
No PC break is set.
1. set   2. clear   3. clear all ...? ↵      ... Terminated by [Enter] key.
>
```

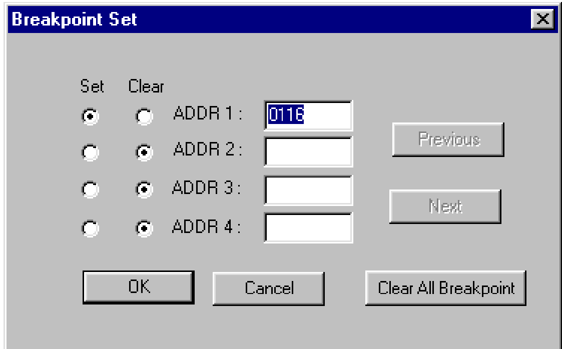
Notes

- The addresses must be specified within the range of the program memory area available for each microcomputer model.
  An error results if the input one is not a hexadecimal number or a valid symbol.
  `Error : invalid value     (no such symbol / symbol type error)`
  An error results if the limit is exceeded.
  `Error : Address out of range, use 0-0xXXXX`
- An error results if you attempt to clear an address that has not been set.
  `Error : Input address does not exist`
- For direct input mode, an error results if you attempt to set breakpoints at more than 16 locations at a time. But for guidance mode, there is no such limitation, so you can specify more than 16 breakpoints before terminating the command by the [Enter] key.
- You can use this command for multiple times to set new breakpoints.

GUI utility

[Break | Breakpoint Set...] menu item

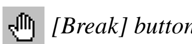
When this menu item is selected, a dialog box appears for setting breakpoints.



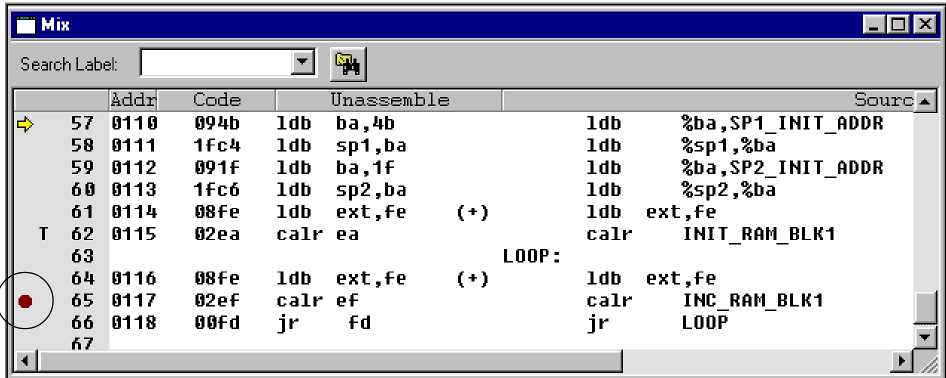
To set a breakpoint, select a [Set] button and enter an address in the text box corresponding to the selected button.
When setting more than four breakpoints, click the [Next] button to continue settings.
The [Previous] and [Next] buttons are used to view previous and subsequent four breakpoints.
To clear a breakpoint, select the [Clear] button of the address to be cleared.
The [Clear All Breakpoint] button clears all the set breakpoints

[Break] button

When this button is clicked after placing the cursor to a line in the [Source] window, the address at the cursor position is set as a breakpoint. If the address has been set as a breakpoint, this button clears the breakpoint.


[Break] button

The set breakpoints are marked with a ● at the beginning of the address lines in the [Source] window.



bc / bpc (break point clear)

Function

This command clears the specified breakpoints that have been set.

Format

>bc [<break1> [. . .<break16>]]↵ (direct input mode)

<break1–16>: Break address; hexadecimal or symbol (IEEE-695 format only)

Examples

```
>bp↵
    1: 0116                                       ... Breakpoints that have been set.
    2: 0200
    3: 0260
1. set   2. clear   3. clear all ...? ↵
>bc 200↵                                          ... Clears breakpoints at address 0x0200.
>bp↵
    1: 0116
    2: 0260
1. set   2. clear   3. clear all ...? ↵
>bc↵                                              ... Clears all breakpoints.
>bp↵
No PC break is set.
1. set   2. clear   3. clear all ...? ↵
>
```

Notes

- The ***bc*** and ***bpc*** commands have the same functions.
- If no address parameter is specified, all the breakpoints that have been set are cleared.
- The format of parameters is same as the ***bp*** command. You can also use the guidance input mode of ***bp*** command to do the same operation.
- You can use this command for multiple times to clear breakpoints.
- An error results if an address that is not set at a breakpoint is specified.
  `Error : Input address does not exist`

GUI utility

[Break | Breakpoint Set …] menu item

When this menu item is selected, a dialog box appears for clearing breakpoints. (See the ***bp*** command.)

[Break] button

When this button is clicked after placing the cursor to a break address line in the [Source] window, the breakpoint is cleared. If the address has not been set as a breakpoint, this button sets a new breakpoint at the address.

[Break] button

bd (data break)

Function

This command sets or clears data break. This command allows you to specify the following break conditions:

1. Memory address range to be read or written (one area)
2. Data pattern to be read or written (bit mask possible)
3. Memory read/write (three conditions: read, write, or read or write)

The program breaks after completing a memory access that satisfies the above conditions.

Format

(1) >bd <data> <option> <address1> <address2>↵ **(direct input mode)**

(2) >bd↵ **(guidance mode)**

Data break set status

1. set 2. clear ...? <1 | 2>↵ (Command is completed when "2" is selected.)

data *Old data* **: <data>↵**

R/W (R,W,*) *Old option* **: <option>↵**

Start address *Old address* **: <address1>↵**

End address *Old address* **: <address2>↵**

>

<data>: Data pattern; binary (* can be input for the bits to be masked)

<option>: Memory read/write option; r, w, or *

<address1–2>: The specified address; hexadecimal or symbol (IEEE-695 format only)

Condition: 0 ≤ address1 ≤ address2 ≤ 0xffff, 0 ≤ data ≤ 0b1111

Examples

Format (1)

```
>bd 1000 W 0 f↵
```

... Sets a data break condition so that the program breaks when 0x8 is written to the address range from 0x0 to 0xf.

* The direct input mode cannot clear the set condition.

Format (2)

```
>bd↵
data: -     R/W: -  area:     -
1. set   2. clear ...? 1↵                 ... "1. set" is selected.
data           ---- : 1***↵               ... Data pattern is set to 0b1***.
R/W (R,W,*)    -    : w↵                  ... R/W condition is set for write access.
Start address ---- : 0↵                   ... Break address range is set to 0x0–0xf.
End   address ---- : f
>bd↵
data: 1***  R/W: W  area: 0000 - 000F     ... Currently set condition.
1. set   2. clear ...? 2↵                 ... "2. clear" is selected.
>bd↵
data: -     R/W: -  area:     -
1. set   2. clear ...? ↵                  ...Terminated by [Enter] key.
>
```

"*" in the binary data pattern specifies that the bit will not be compared with the actual read/write data.

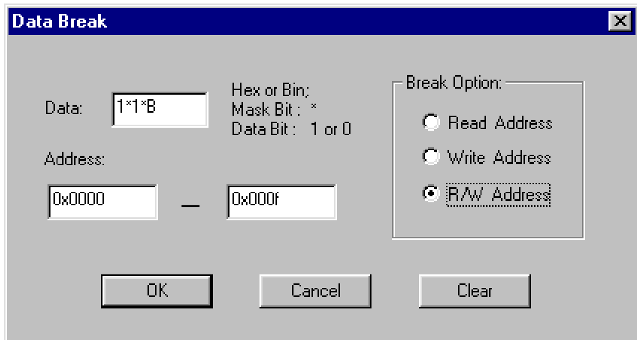
Notes

- For the first time this command is executed, no item can be skipped because no default value is set.
- In guidance mode, the following keyboard inputs have special meaning:
  "q↵" … Command is terminated. (finish inputting and start execution)
  "^↵" … Return to previous item.
  "↵" … Input is skipped. (keep current value)
  When the command is terminated in the middle of guidance by "q↵", the contents that have been input up to that time will be modified. However, these contents will not be modified if some cleared settings are left intact.
- The addresses must be specified within the range of the data memory area available for each microcomputer model.
  An error results if the input one is not a hexadecimal number or a valid symbol.
  ```
  Error : invalid value     (no such symbol / symbol type error)
  ```
  An error results if the limit is exceeded.
  ```
  Error : Address out of range, use 0-0xFFFF
  ```
- An error results if the start address in the address range is larger than the end address.
  ```
  Error : end address < start address
  ```
- Address and R/W specifications are effective even for 16-bit access (push/pop to and from SP1 stack). However, the data specification will not have any effect because data is compared with a 4-bit bus. In this case, specify data with "****". When setting a break for 4-bit access, be careful not to specify an address that overlaps the 16-bit access area, because such specification can cause the system to operate erratically.
- The data value can be input as a binary number with or without mask bits in the range of 4 bits (0 to 0xf). An error results if the limit is exceeded.
  ```
  Error : invalid data pattern
  ```
- An error results if you input the R/W option other than "r", "w" or "*".
  ```
  Error : Incorrect r/w option, use r/w/*
  ```
- The program stops one to two instructions after the break condition has been met.

GUI utility

[Break | Data Break ...] menu item

When this menu item is selected, a dialog box appears for setting a data break condition.



To set a data break condition, enter an address and a data pattern in the text box, and select R/W condition from the radio buttons. Then click [OK].
To clear the set data break condition, click [Clear].

bdc (data break clear)

Function

This command clears the data break condition that has been set.

Format

>bdc↵ **(direct input mode)**

GUI utility

[Break | Data Break …] menu item

When this menu item is selected, a dialog box appears for clearing the set data break condition. (See the ***bd*** command.)

br (register break)

Function

This command sets or clears register break. This command allows you to specify data or a mask that constitutes a break condition for each register (A, B, F, X, and Y). The program will break when all setting conditions are met.

Format

(1) >br <register> <value> [<register> <value> [...<register> <value>]]↵ **(direct input mode)**

(2) >br↵ **(guidance mode)**

Register break set status
1. set 2. clear ...? <1 | 2>↵ (Command is completed when "2" is selected.)
A *Old value* **: <value>↵**
B *Old value* **: <value>↵**
FE *Old value* **: <value>↵**
FI *Old value* **: <value>↵**
FC *Old value* **: <value>↵**
FZ *Old value* **: <value>↵**
X *Old value* **: <value>↵**
Y *Old value* **: <value>↵**
>

<register>: Register name; A, B, F, X or Y
<value>: Data pattern for the register; hexadecimal or binary (F register) (* can be used for the bits to be masked)

Examples

Format (1)

```
>br f **1*↵          ... Sets a register break condition so that the program breaks when the C flag is set.
```

Format (2)

```
>br↵
A: -   B: -   X: -      Y: -      EICZ: -
1. set    2. clear ...? 1↵                  ... "1. set" is selected.
A             - : a↵                        ... Data 0xa is set for A register condition.
B             - : *↵                        ... "*" masks the register condition.
FE            - : *↵
FI            - : *↵
FC            - : 1↵
FZ            - : *↵
X             - : 20↵
Y             - : ^↵                        ... "^↵" returns guidance to previous setting.
X            20 : 60↵
Y             - : *↵
>br↵
A:a  B:*  X:60  Y:*  EICZ:**1*
1. set    2. clear ...? 2↵                  ... "2. clear" is selected.
>br↵
A: -   B: -   X: -      Y: -      EICZ: -
1. set    2. clear ...? ↵                   ...Terminated by [Enter] key.
>
```

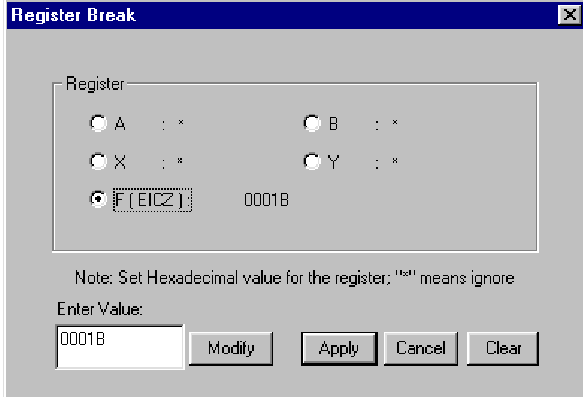
Notes

- For the first time this command is executed, no item can be skipped because no default value is set.
- In guidance mode, the following keyboard inputs have special meaning:
  "q↵" … Command is terminated. (finish inputting and start execution)
  "^↵" … Return to previous address.
  "↵" … Input is skipped. (keep current value)
  When the command is terminated in the middle of guidance by "q↵", the contents that have been input up to that time will be modified. However, these contents will not be modified if some cleared settings are left intact.
- An error results if you input the register name other than A, B, X, Y or F when using the direct input mode.
  ```
  Error : Incorrect register name, use A/B/X/Y/F
  ```
- You can use the direct input mode to set register break condition at a time, or change one or several items for register break setting.
- The register value can be input as a binary number with or without mask bits or a hexadecimal number in the range of the bit width of each register. An error results if the limit is exceeded.
  ```
  Error : invalid data pattern
  ```
- The program stops one to two instructions after the break condition has been met.

GUI utility

[Break | Register Break ...] menu item

When this menu item is selected, a dialog box appears for setting register break conditions.



To set a register condition, select the radio button for the register and enter a value in the [Enter Value:] box, then click [Modify]. All the register condition must be set. Enter an "*" to exclude the register from the break condition.
When the [Apply] button is clicked, the dialog box closes and the register break is set with the specified conditions. However, if there is a register of which the condition has not been set (indicated with "---"), no register break condition is set.
To clear the register break conditions, click [Clear].

brc (register break clear)

Function

This command clears the register break conditions that have been set.

Format

>brc↵ **(direct input mode)**

GUI utility

[Break | Register Break …] menu item

When this menu item is selected, a dialog box appears for clearing the register break conditions. (See the ***br*** command.)

bs (sequential break)

Function

This command sets and clears sequential break and displays the sequential break condition that have been set.
This command allows you to set break addresses in up to three locations and the number of times you want the program to be executed at the last of the three addresses. While passing through all addresses sequentially in the order they are set, the program executes the last-specified address a specified number of times, then breaks after fetching the instruction from that address again.

Format

(1) >bs <pass> <address1> [<address2> [<address3>]]↵ **(direct input mode)**

(2) >bs↵ **(guidance mode)**

Sequential break set status
1. set 2. clear ...? <1 | 2>↵ (Command is completed when "2" is selected.)
Number of sequential address (1–3) ? : <1 | 2 | 3>↵
Set address ? *Old address* **: <address1>↵**
Set address ? *Old address* **: <address2>↵**
Set address ? *Old address* **: <address3>↵**
Pass count ? *Old count* **: <pass>↵**
>

<pass>: Pass count; decimal
<address1–3>: Program execution address; hexadecimal or symbol (IEEE-695 format only)
Condition: 0 ≤ address1–3 ≤ last program memory address, 0 ≤ pass ≤ 4095

Examples

Format (1)
```
>bs 3 116 120↵
```
... Sets two sequential addresses and the pass count. In this case, a break will occur when the CPU fetches the instruction at address 0x0120 after the instruction at address 0x0116 is executed and the instruction at address 0x0120 is executed three times.

Format (2)
```
>bs↵
1: -     2: -     3: -     pass: -
1. set   2. clear ...? 1↵                        ... "1. set" is selected.
Number of seqential address (1-3) ? : 2↵         ... Number of addresses is input.
Set address ? : 116↵                             ... 1st address is input.
Set address ? : 120↵                             ... 2nd address is input.
Pass count  ? : 3↵                               ... Pass count is input.
>bs↵
1: 0116  2: 0120  3: -     pass:3
1. set   2. clear ...? 2↵                        ... "2. clear" is selected.
>bs↵
1: -     2: -     3: -     pass: -
1. set   2. clear ...? ↵                         ...Terminated by [Enter] key.
>
```

∗ If you press [Enter] in the middle of a guidance, the command is canceled.

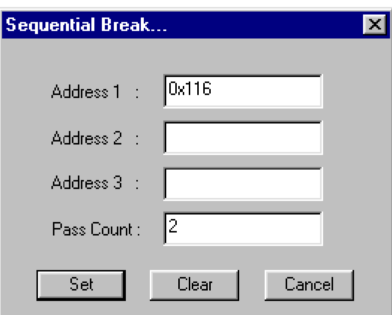
Notes

- The maximum number of times a program can be executed is 4,095. Specifying a pass count exceeding this limit will result in an error.
  ```
  Error : Number of passes out of range, use 0-4095
  ```
- The addresses must be specified within the range of the program memory area available for each microcomputer model.
  An error results if the input one is not a hexadecimal number or a valid symbol.
  ```
  Error : invalid value     (no such symbol / symbol type error)
  ```
  An error results if the limit is exceeded.
  ```
  Error : Address out of range, use 0-0xXXXX
  ```

GUI utility

[Break | Sequential Break ...] menu item

When this menu item is selected, a dialog box appears for setting sequential break conditions.



To set a sequential break, enter sequential addresses and a pass count in the text boxes, then click [OK]. At least one address (Address 1) and the pass count must be set.
To clear the sequential break condition, click [Clear].

bsc (sequential break clear)

Function

This command clears the sequential break condition that has been set.

Format

>bsc↵ **(direct input mode)**

GUI utility

[Break | Sequential Break …] menu item

When this menu item is selected, a dialog box appears for clearing sequential break conditions. (See the ***bs*** command.)

bsp (break stack pointer)

Function

This command allows you to specify a stack area to generate a break for illegal stack access.
A break occurs when stack operation is performed in locations other than the area specified by this command.

Format

(1) >bsp <address1> <address2> <address3> <address4>↵ **(direct input mode)**

(2) >bsp↵ **(guidance mode)**

Stack area set status
SP1 start address ? : <address1>↵
SP1 end address ? : <address2>↵
SP2 start address ? : <address3>↵
SP2 end address ? : <address4>↵
>

<address1>: SP1 start address; hexadecimal or symbol (IEEE-695 format only)
<address2>: SP1 end address; hexadecimal or symbol (IEEE-695 format only)
<address3>: SP2 start address; hexadecimal or symbol (IEEE-695 format only)
<address4>: SP2 end address; hexadecimal or symbol (IEEE-695 format only)
Condition: $0 \leq$ address1(2) $\leq$ 0x03ff, $0 \leq$ address3(4) $\leq$ 0x00ff

Examples

Format (1)

```
>bsp 0 3ff 0 ff↵                    ... Sets SP1 area to 0x0–0x3FF and SP2 area to 0x0–0xFF.
```

Format (2)

```
>bsp↵
SP1 : 0000 - 03FF    SP2 : 0000 - 00FF
SP1 start address ? : 0↵                    ... Address is input.
SP1 end   address ? : 1ff↵
SP2 start address ? : 0↵
SP2 end   address ? : ff↵
>bsp↵
SP1 : 0000 - 01FF    SP2 : 0000 - 00FF
SP1 start address ? : ↵                     ... Terminated by [Enter] key.
>
```

* If you press only [Enter] in the middle of a guidance, the command is canceled.

Notes

- The stack area that is set by this command will not affect the stack operation performed in the program.
- Specify the SP1 address in the range of 0 to 0x3ff and the SP2 address in the range of 0 to 0xff. Entering an address exceeding this limit will result in an error.

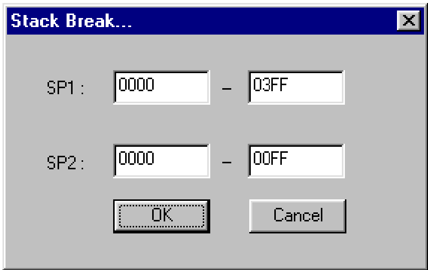
```
Error : SP1 address out of range, use 0-0x3FF
Error : SP2 address out of range, use 0-0xFF
```

- Specify the SP1 address in units of 4 words (start address = multiple of 4; end address = multiple of 4 + 3).
- Due to the E0C63000 CPU's prefetch function, SP1 can access the top end of the actually used stack + 4 words. Depending on your system configuration, add 4 to the end address when you set it.
- The program stops one to two instructions after the break condition has been met.

GUI utility

[Break | Stack Break ...] menu item

When this menu item is selected, a dialog box appears for setting stack areas.



To set stack areas, enter start and end addresses in the text boxes, then click [OK].

bl (break point list)

Function

This command lists the current setting of all break conditions.

Format

>bl↵ **(direct input mode)**

Example

```
>bl↵
PC break:
    1: 0116
    2: 0200
Sequential break:
1: 0116  2: 0120  3: -     pass:3
Data break:
data: 1***  R/W: W  area: 0000 - 000F
Register break:
A:*  B:*  X:*  Y:*  EICZ:**1*
Stack break:
SP1 : 0000 - 03FF    SP2 : 0000 - 00FF
>
```

GUI utility

[Break | Break List] menu item

When this menu item is selected, the ***bl*** command is executed.

bac (break all clear)

Function

This command clears all break conditions set by the ***bp***, ***bd***, ***br*** and/or ***bs*** commands.

Format

>bac↵ (direct input mode)

GUI utility

[Break | Break All Clear] menu item
When this menu item is selected, the ***bac*** command is executed.

8.9.10 Program Display

u (unassemble)

Function

This command displays a program in the [Source] window after unassembling it. The display contents are as follows:

- Program memory address
- Object code
- Unassembled contents of the program

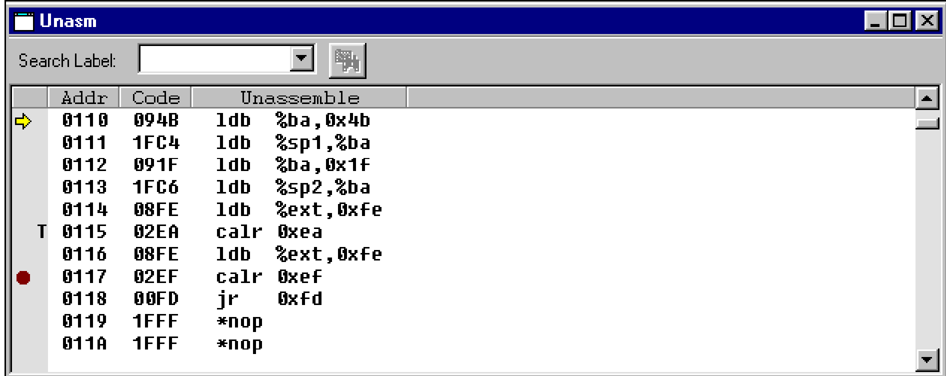
Format

>u [<address>]↵ (direct input mode)

<address>: Start address for display; hexadecimal or symbol (IEEE-695 format only)
Condition: 0 ≤ address ≤ last program memory address

Display

(1) When [Source] window is opened



If <address> is not specified, display in the [Source] window is changed to the unassemble display mode. If <address> is specified, display in the [Source] window is changed to the unassemble display mode. At the same time, code is displayed beginning with <address>.

(2) When [Source] window is closed

The 16 lines of unassembled result are displayed in the [Command] window. The system then waits for a command input.
If <address> is not specified, this display begins with the current PC (displayed in the [Register] window). If <address> is specified, the display begins with <address>.

```
>u↵
 ADDR   CODE    UNASSEMBLE
 0110   094B    ldb   %ba,0x4b
 0111   1FC4    ldb   %sp1,%ba
 0112   091F    ldb   %ba,0x1f
 0113   1FC6    ldb   %sp2,%ba
  :      :       :       :
 011E   1FFF    *nop
 011F   1FFF    *nop
>
```

(3) During log output

If the command execution result is being output to a log file as specified by the ***log*** command, code is displayed in the [Command] window and its contents are also output to the log file.
If the [Source] window is closed, the result is displayed in the same way as in (2) above.
If the [Source] window is opened, the window is redisplayed. In this case, the same number of lines is displayed in the [Command] window as displayed in the [Source] window.

(4) Successive display

If you execute the ***u*** command after entering it from the keyboard, code can be displayed successively by entering the [Enter] key only until some other command is executed.
When you press the [Enter] key, the [Source] window is scrolled forward one screen.
When displaying code in the [Command] window, 16 lines of code following the previously displayed address are displayed (the same number of lines as displayed in the [Source] window if the ***u*** command is executed during log output).

Note

The display start address you specified must be within the range of the program memory area available with each microcomputer model.
An error results if the input one is not a hexadecimal number or not a valid symbol.

```
Error : invalid value    (no such symbol / symbol type error)
```

An error results if the limit is exceeded.

```
Error : Address out of range, use 0-0xXXXX
```

GUI utility

[View | Program | Unassemble] menu item, [Unassemble] button

When this menu item or button is selected, the [Source] window opens or activates and displays the program from the current PC address.

[Unassemble] button

SC (source code)

Function

This command displays the contents of the program source file in the [Source] window. The display contents are as follows:

- Line number in the source file
- Source code

Format

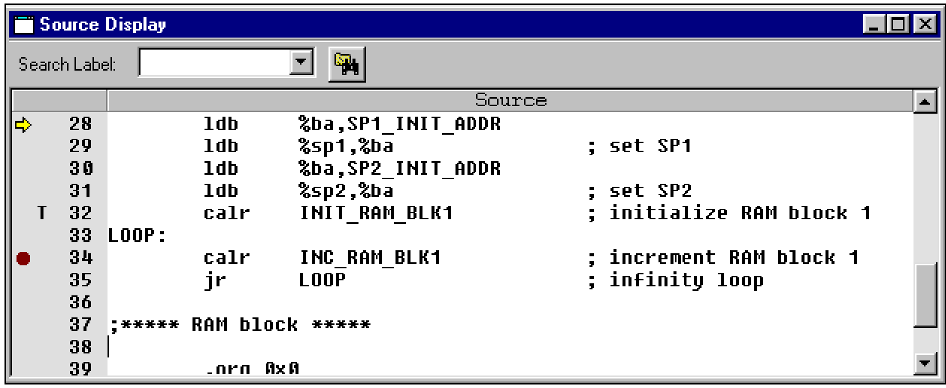
>sc [<address>]↵ (direct input mode)

<address>: Start address for display; hexadecimal or symbol (IEEE-695 format only)

Condition: 0 ≤ address ≤ last program memory address

Display

(1) When [Source] window is opened



If <address> is not specified, display in the [Source] window is changed to the source display mode. If <address> is specified, display in the [Source] window is changed to the source display mode. At the same time, code is displayed beginning with <address>.

(2) When [Source] window is closed

The 16 lines of source code are displayed in the [Command] window. The system then waits for a command input.

If <address> is not specified, this display begins with the current PC (displayed in the [Register] window). If <address> is specified, the display begins with <address>.

```
>sc↵
        ldb  %ba,SP1_INIT_ADDR
        ldb  %sp1,%ba           ; set SP1
        ldb  %ba,SP2_INIT_ADDR
        ldb  %sp2,%ba           ; set SP2
        calr INIT_RAM_BLK1      ; initialize RAM block 1
LOOP:
        ldb  %ext,INC_RAM_BLK1@rh
        calr INC_RAM_BLK1@rl    ; increment RAM block 1
        ldb  %ext,LOOP@rh
        jr   LOOP@rl            ; infinity loop
>
```

(3) During log output

If the command execution result is output to a log file as specified by the ***log*** command, code is displayed in the [Command] window and its contents are also output to the log file.
If the [Source] window is closed, code is displayed in the same way as in (2) above.
If the [Source] window is open, the window is redisplayed. In this case, the same number of lines is displayed in the [Command] window as displayed in the [Source] window.

(4) Successive display

If you execute the ***sc*** command after entering it from the keyboard, code can be displayed successively by entering the [Enter] key only until some other command is executed.
When you press the [Enter] key, the [Source] window is scrolled forward one screen.
When displaying code in the [Command] window, 16 lines of code following the previously displayed address are displayed (the same number of lines as displayed in the [Source] window if the ***sc*** command is executed during log output).

Notes

- Source codes can be displayed only when an absolute object file that contains source debug information has been loaded.
- The display start address you specified must be within the range of the program memory area available with each microcomputer model.
  An error results if the input one is not a hexadecimal number or not a valid symbol.
  ```
  Error : invalid value    (no such symbol / symbol type error)
  ```
  An error results if the limit is exceeded.
  ```
  Error : Address out of range, use 0-0xXXXX
  ```

GUI utility

[View | Program | Source Display] menu item, [Source] button

When this menu item or button is selected, the [Source] window opens or activates and displays the program from the current PC address.

[Source] button

m (mix)

Function

This command displays the unassembled result of the program and the contents of the program source file in the [Source] window. The display contents are as follows:

- Line number
- Program memory address
- Object code
- Unassembled contents of the program
- Source code

Format

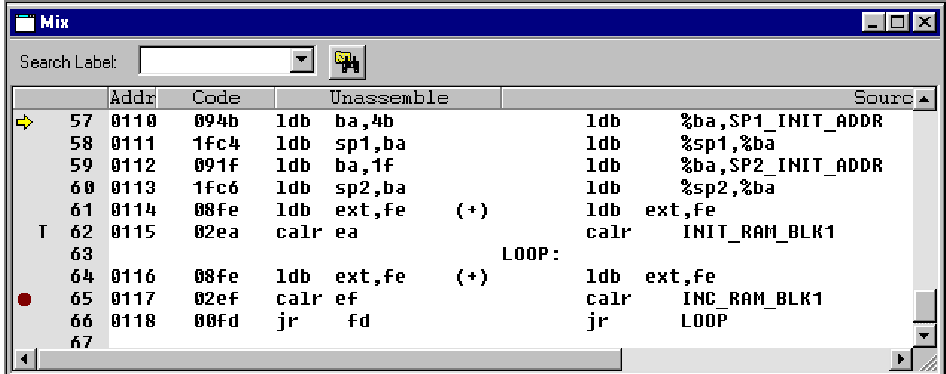
>m [<address>] ↵ (direct input mode)

<address>: Start address for display; hexadecimal or symbol (IEEE-695 format only)

Condition: $0 \leq$ address $\leq$ last program memory address

Display

(1) When [Source] window is opened



If <address> is not specified, display in the [Source] window is changed to the mix (unassemble & source) display mode. If <address> is specified, display in the [Source] window is changed to the mix (unassemble & source) display mode. At the same time, code is displayed beginning with <address>.

(2) When [Source] window is closed

The 16 lines of mix display are produced in the [Command] window. The system then waits for a command input.

If <address> is not specified, this display begins with the current PC (displayed in the [Register] window). If <address> is specified, the display begins with <address>.

```
>m↵
0110   094b   ldb  ba,4b                   ldb      %ba,SP1_INIT_ADDR
0111   1fc4   ldb  sp1,ba                  ldb      %sp1,%ba
0112   091f   ldb  ba,1f                   ldb      %ba,SP2_INIT_ADDR
0113   1fc6   ldb  sp2,ba                  ldb      %sp2,%ba
0114   08fe   ldb  ext,fe      (+)         ldb   ext,fe
0115   02ea   calr ea                      calr     INIT_RAM_BLK1
                                     LOOP:
0116   08fe   ldb  ext,fe      (*)         ldb      %ext,INC_RAM_BLK1@rh
0117   02ef   calr ef                      calr     INC_RAM_BLK1@rl
                               (-)         ldb      %ext,LOOP@rh

  :      :      :    :               :      :           :
```

(3) During log output

If the command execution result is output to a log file as specified by the ***log*** command, code is displayed in the [Command] window and its contents are output to the log file also.
If the [Source] window is closed, code is displayed in the same way as in (2) above.
If the [Source] window is open, the window is redisplayed. In this case, the same number of lines is displayed in the [Command] window as displayed in the [Source] window.

(4) Successive display

If you execute the ***m*** command after entering it from the keyboard, code can be displayed successively by entering the [Enter] key only until some other command is executed.
When you press the [Enter] key, the [Source] window is scrolled forward one screen.
When displaying code in the [Command] window, 16 lines of code following the previously displayed address are displayed (the same number of lines as displayed in the [Source] window if the ***m*** command is executed during log output).

Notes

- Source codes can be displayed only when an absolute object file that contains source debug information has been loaded.
- The display start address you specified must be within the range of the program memory area available with each microcomputer model.
  An error results if the input one is not a hexadecimal number or not a valid symbol.
  ```
  Error : invalid value    (no such symbol / symbol type error)
  ```
  An error results if the limit is exceeded.
  ```
  Error : Address out of range, use 0-0xXXXX
  ```

GUI utility

[View | Program | Mix Mode] menu item, [Mix] button

When this menu item or button is selected, the [Source] window opens or activates and displays the program from the current PC address.

[Mix] button

8.9.11 Symbol Information

sy (symbol list)

Function

This command displays a list of symbols in the [Command] window.

Format

(1) >sy [/a]↵ (direct input mode)
(2) >sy $<keyword> [/a]↵ (direct input mode)
(3) >sy #<keyword> [/a]↵ (direct input mode)

<keyword>: Search character string; ASCII character
Condition: 0 ≤ length of keyword ≤ 32

Examples

Format (1)

```
>sy↵
INC_RAM_BLK1                                        0007
INIT_RAM_BLK1                                       0000
RAM_BLK0                                            0000
RAM_BLK1                                            0004
BOOT@C:\E0C63\TEST\MAIN.S                           0110
LOOP@C:\E0C63\TEST\MAIN.S                           0116
NMI@C:\E0C63\TEST\MAIN.S                            0100
>
```

In format (1), all the defined symbols are displayed in alphabetical order. Global symbols are displayed first, then local symbols. Shown to right to each symbol is the address that is defined in it.

Format (2)

```
>sy $R↵
INC_RAM_BLK1                                        0007
INIT_RAM_BLK1                                       0000
RAM_BLK0                                            0000
RAM_BLK1                                            0004
>
```

In format (2), the debugger displays global symbols that contain the character string specified by <keyword>.

Format (3)

```
>sy #B↵
BOOT@C:\E0C63\TEST\MAIN.S                           0110
>
```

In format (3), the debugger displays local symbols that contain the character string specified by <keyword>.

When local symbols are displayed, @ and the source file name in which the symbol is defined are added.

Notes

- The symbol list will be sorted by letter order if no option is added. If the option is added, the symbol list will be sorted by address.
- The symbol list can only be displayed when the object file in IEEE-695 format has been read.
- The specification of keyword conforms to which defined for assembler tools.

GUI utility

None

8.9.12 Load File

lf (load file)

Function

This command loads an object file in IEEE-695 format into the debugger.

Format

(1) >lf <file name>↵ **(direct input mode)**

(2) >lf↵ **(guidance mode)**

File Name ? <file name>↵

>

<file name>: File name to be loaded (path can also be specified)

Examples

Format (1)

```
>lf test.abs↵
Loading file ... OK!
>
```

Format (2)

```
>lf↵
File name   ? test.abs
Loading file ... OK!
>
```

Notes

- An error results if the loaded file is linked with a different ICE parameter file than the one the debugger is using.

  ```
  Error : Different chip type, cannot load this file
  ```

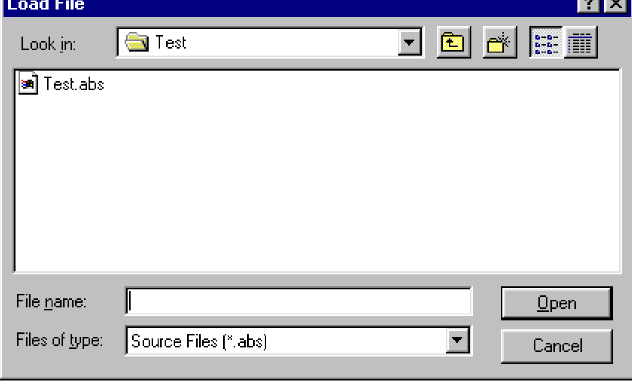
- Only an IEEE-695 format object file (generated by the linker) can be loaded by the *lf* command.
- If you want to use source display and symbols when debugging a program, the object file must be in IEEE-695 format that contains debug information loaded into the computer.
- If the [Source] window is opened when loading a file, its contents are updated. The program contents are displayed from the current PC address.
- If an error occurs when loading a file, portions of the file that have already been read will remain in the emulation memory.

GUI utility

[File | Load File ...] menu item, [Load File] button

When this menu item or button is selected, a dialog box appears allowing selection of an object file to be loaded.

[Load File] button



lo (load option)

Function

This command loads a Motorola-S format program, data or option file listed below into the debugger.

File	Name specification
Program file	~.hsa (5 high-order bits), ~.lsa (8 low-order bits)
Data file*	~.csa
Function option data file	~.fsa
Segment option data file*	~.ssa
Melody data file*	~.msa

* Not used in some microcomputer models

Format

(1) >lo <file name>↵ **(direct input mode)**

(2) >lo↵ **(guidance mode)**

File Name ...? <file name>↵

>

<file name>: File name to be loaded (path can also be specified)

Examples

Format (1)

```
>lo test.lsa↵
Loading file ... OK!
>
```

...Loads the program files test.lsa and test.hsa.

Format (2)

```
>lo↵
File name    ? test.fsa↵
Loading file ... OK!
>
```

...Loads a function option file.

Notes

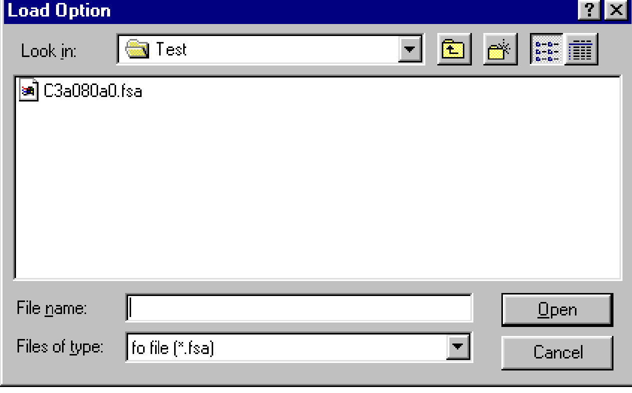
- The debugger determines the file type based on the specified file name. Therefore, the debugger cannot load a file not following to the name specification listed above, and an error will result.
  ```
  Error : invalid file name
  ```
- If an error occurs when loading a file, portions of the file that have already been read are left as they were loaded.

GUI utility

[File | Load Option ...] menu item, [Load Option] button

When this menu item or button is selected, a dialog box appears allowing selection of a hex file to be loaded.

[Load Option] button



8.9.13 Flash Memory Operation

lfl (load from flash memory)

Function

This command loads the memory contents from the flash memory of the ICE63 into the target memory. It therefore allows you to debug the program beginning from the contents previously saved to the flash memory up to latest one.

Format

(1) >lfl <content> [... <content>]↵ (direct input mode)

(2) >lfl↵ (guidance mode)

```
Read program  1. yes  2. no ...? <1 | 2>↵
     data     1. yes  2. no ...? <1 | 2>↵
     fog      1. yes  2. no ...? <1 | 2>↵
     sog      1. yes  2. no ...? <1 | 2>↵
     mla      1. yes  2. no ...? <1 | 2>↵
Loading ...
>
```

<content>: Data type; p (program) / d (data) / f (fog) / s (sog) / m (mla)

Examples

Format (1)

```
>lfl p↵                                    ...Loads program data.
Loading from flash memory ... done!
```

Format (2)

```
>lfl↵
Read program  1.yes  2.no ...? 1↵          ...Select the contents to be loaded.
     data     1.yes  2.no ...? 1↵
     fog      1.yes  2.no ...? 1↵
     sog      1.yes  2.no ...? 1↵
     mla      1.yes  2.no ...? 1↵
Loading from flash memory ... done!
>
```

Notes

- If the flash memory is protected against read/write, an error will result and memory contents will not be loaded into the target memory.
  `Error : flash ROM is protected`
- If the flash memory has been erased, an error will result and memory contents will not be loaded into the target memory.
  `Error : format error`

- If the flash memory and target memory are mapped differently (e.g., the parameter file used in the current debug differs from one that was used when the program was saved to the flash memory), an error will result and memory contents will not be loaded into the target memory.

```
Error : Map information is not the same
```

In this case, the system displays the map information of the target memory and the flash memory after showing the message above.

```
                      ICE   flash
---------------------------------
Chip name            63A08
Parameter version      02     00
Size of program      2000      0
        data RAM      800   8000
        data ROM     1000   7000
        ext. memory   100    700
        LCD           2C0    800
        IO             20     20
        FO             20     F0
        SO1             0   1000
        SO2           100   1000
        MLA           510   1000
```

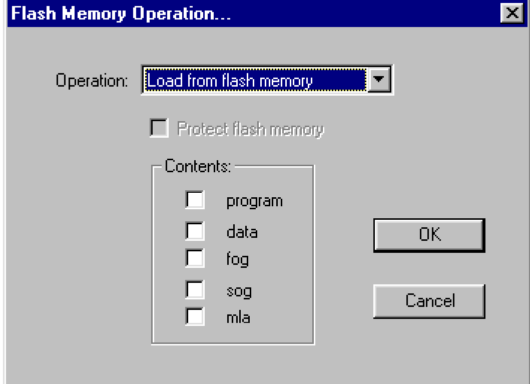
Redo the loading with the correct parameter file using the ***efl*** or ***sfl*** command.

- If an error occurs when loading data, portions of the data that have already been read into the target memory are left as they were loaded.

GUI utility

[File | Flash Memory Operation...] menu item

When this menu item is selected, a dialog box appears allowing selection of flash memory operations.



To execute the ***lfl*** command, select "Load from flash memory" from the [Operation] list box and select contents using the check boxes, then clock [OK].

sfl (save to flash memory)

Function

This command writes the contents of the target memory in the ICE63 into the flash memory.
Writing to the flash memory allows the ICE63 to be operated in free-run mode. Furthermore, the next debug session can be continued immediately from the current contents in the flash memory.

Format

```
(1) >sfl <content> [ ... <content>] [-p]↵                 (direct input mode)

(2) >sfl↵                                                 (guidance mode)
    Protect flash memory  1. yes  2. no ...? <1 | 2>↵
           Write program  1. yes  2. no ...? <1 | 2>↵
                 data     1. yes  2. no ...? <1 | 2>↵
                 fog      1. yes  2. no ...? <1 | 2>↵
                 sog      1. yes  2. no ...? <1 | 2>↵
                 mla      1. yes  2. no ...? <1 | 2>↵
    Saving ...
    >
```

<content>: Data type; p (program) / d (data) / f (fog) / s (sog) / m (mla)
-p: Protect option

Examples

Format (1)

```
>sfl p d f s m -p↵                          ...Saves all contents and sets prorect.
Please wait few minutes
Save to flash memory ... done!
>
```

Format (2)

```
>sfl↵
Protect flash memory    1.yes   2.no ...? 1↵     ... Protect is set.
        Write program   1.yes   2.no ...? 1↵     ... Write contents are selected.
              data      1.yes   2.no ...? 1↵
              fog       1.yes   2.no ...? 1↵
              sog       1.yes   2.no ...? 1↵
              mla       1.yes   2.no ...? 1↵
Please wait few minutes
Save to flash memory ... done!
>
```

* If you enter only the [Enter] key in the middle of guidance, the guidance is terminated and only the area you have selected up to that time is written into the flash memory.

Notes

- If the flash memory is write-protected, an error results and memory contents are not written to the flash memory.

  ```
  Error : flash ROM is protected
  ```

  The write-protect can be removed by erasing the flash memory with the ***efl*** command.

- If the flash memory has been erased, an error results, in which case you can choose to continue or stop processing.

```
Error : format error
Save with the map information,
 or quit the command ?  1.save  2.quit ...? 1
Protect flash memory    1.yes   2.no ...?
```

- If the flash memory and target memory are mapped differently, an error results. In this case, the system displays the map information of the memory and a message prompting you to choose to continue or stop processing.

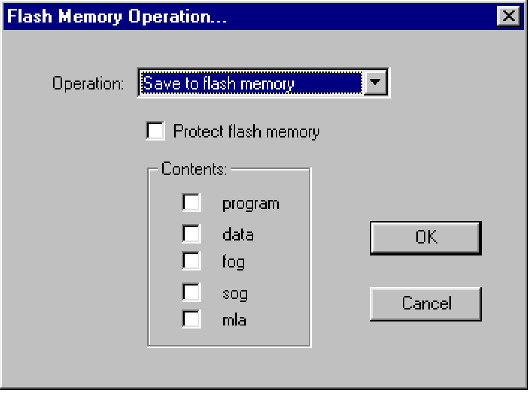
```
Error : Map infomation is not the same
                      ICE  flash
---------------------------------
Chip name            63A08
Parameter version      02    00
Size of program      2000     0
        data RAM      800  8000
        data ROM     1000  7000
        ext. memory   100   700
        LCD           2C0   800
        IO             20    20
        FO             20    F0
        SO1             0  1000
        SO2           100  1000
        MLA           510  1000
Save with the map information,
 or quit the command ?  1.save  2.quit ...? 1
Protect flash memory    1.yes   2.no ...?
```

- When shipped from the factory or erased by the ***efl*** command, all data in the flash memory is initialized to 0xff. When part of the data, such as a program, is written to the flash memory by the ***slf*** command, all other data in it remains unchanged (= 0xff). In this condition, the ICE63 cannot be operated in free-run mode. To operate the ICE63 in free-run mode, always make sure that after erasing the flash memory, all the data has been written into the flash memory.
  In the ICE63, furthermore, the default values for all option data are 0x00. Consequently, if you write to the flash memory before loading option data (***lo*** command), the data you have written to the flash memory is overwritten by 0x00.

GUI utility

[File | Flash Memory Operation...] menu item

When this menu item is selected, a dialog box appears allowing selection of flash memory operations.



To execute the ***sfl*** command, select "Save to flash memory" from the [Operation] list box and select contents to be saved using the check boxes, then clock [OK]. The -p option can be specified using the [Protect flash memory] check box.

efl (erase flash memory)

Function

This command erases the contents of the ICE63's flash memory (including map information) and removes its protect function.

Format

efl↵ **(direct input mode)**

Example

```
>efl↵
Clear flash memory ... done!
>
```

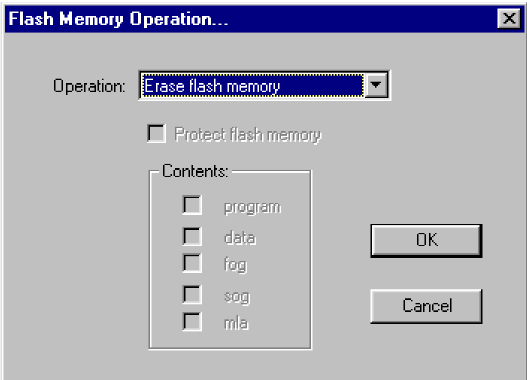
Note

When erased by the ***efl*** command, all data in the flash memory is initialized at 0xff. Even when part of the data, such as a program, is thereafter written to the flash memory by the ***slf*** command, all other data remains unchanged (= 0xff). In this condition, the ICE63 cannot be operated in free-run mode. In order for the ICE63 to be operated in free-run mode, always make sure that after erasing the flash memory, all the data has been written into the flash memory.
In the ICE63, furthermore, the default values for all option data are 0x00. Consequently, if you write to the flash memory before loading option data (***lo*** command), the data you have written to the flash memory is overwritten by 0x00.

GUI utility

[File | Flash Memory Operation...] menu item

When this menu item is selected, a dialog box appears allowing selection of flash memory operations.



To execute the ***efl*** command, select "Erase flash memory" from the [Operation] list box and then clock [OK].

8.9.14 Trace

tm (trace mode)

Function

This command sets and displays a trace mode. It allows you to set the following three trace modes and a trace trigger point (when a specified address is executed, the TRGOUT pin outputs a pulse).

1. Normal trace mode
   The data written to the trace memory is always the latest trace information.
2. Single-delay trigger trace mode
   One of the following three trace sampling areas can be specified with respect to the trace trigger point:
   • Start: Trace information is a sample beginning from the trace trigger point.
   • Middle: Trace information is a sample from before and after the trace trigger point.
   • End: Trace information is a sample all the way up to the trace trigger point.
3. Address-area trace mode
   The execution process is traced as instructions inside or outside a specified address range are executed. This address range can be specified in up to four locations.

Format

(1) >tm <mode> <trigger> [<option>] [<addr1> <addr2> [... <addr7> <addr8>]↵ (direct input mode)

(2) >tm↵ (guidance mode)

Current type setting
1. normal 2. single delay 3. address area . . . ? <1 | 2 | 3>↵
Trigger address ? : <trigger>↵
...... (guidance depends on the above selection, see examples)
>

<mode>:	Trace mode; -n (normal), -s (single delay), or -a (address area)
<trigger>:	Trace trigger address; hexadecimal or symbol (IEEE-695 format only)
<option>:	For single-delay trace mode: s (start) / m (middle) / e (end)
	For addres-area trace mode: i (in area) / o (out area)
<addr1–8>:	Address ranges; hexadecimal or symbol (IEEE-695 format only)
Condition:	0 ≤ trigger, addr1–8 ≤ last program memory address

Examples

Format (1)

```
>tm -n 116↵                          ... Sets normal trace mode and sets trigger point to 0x0116.
```

Format (2)

```
>tm↵
Normal mode
Trigger Address : 0
1.normal  2.single delay  3.address area ...? 1↵  ... [1. normal] is selected.
Trigger address ? :116↵                           ... Trigger address is input.
>tm↵
Normal mode
Trigger Address : 0116
1.normal  2.single delay  3.address area ...? 2↵  ... [2. single delay] is selected.
Trigger address ? :116↵                           ... Trigger address is input.
1.start  2.middle  3.end ...? 2↵                  ... Trace sampling area is selected.
>
```

```
>tm↵
Single delay mode
Trigger Address : 0116
Position: Middle
1.normal  2.single delay  3.address area ...? 3↵  ... [2. address area] is selected.
Trigger address ? :116↵                           ... Trigger address is input.
1.in area  2.out area ...? 1↵                      ... In/out is selected.
Start address ? 110↵                               ... Address range is input
End address ? 200↵                                     in up to 4 locations.
Start address ? ↵                                  ... Terminated by [Enter] key.
>
```

If you enter the [Enter] key only, the command will be canceled.
However, if more than one pair of addresses is specified after selecting the address-area trace mode (one pair of addresses is specified in the above example), the range of specified addresses will be set as the trace area.

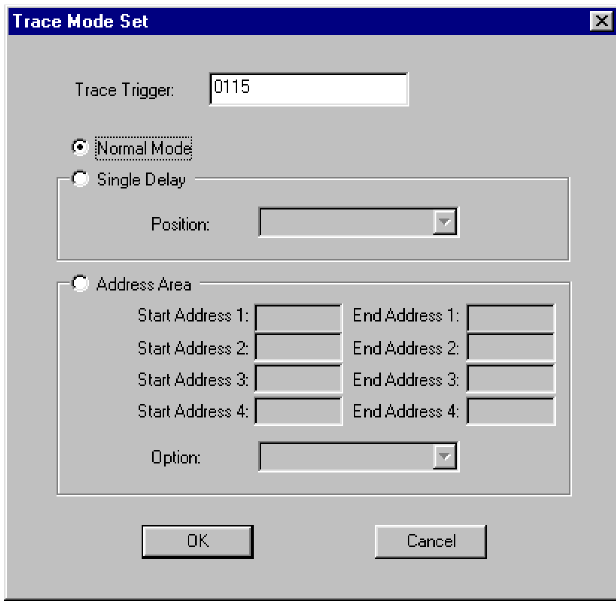
Notes

- The trigger addresses set here are marked by the letter "T" at the beginning of the address lines displayed in the [Source] window.
- The address you specified must be within the range of the program memory area available with each microcomputer model.
  An error results if the input one is not a hexadecimal number or not a valid symbol.
  `Error : invalid value    (no such symbol / symbol type error)`
  An error results if the limit is exceeded.
  `Error : Address out of range, use 0-0xXXXX`

GUI utility

[Trace | Trace Mode Set ...] menu item

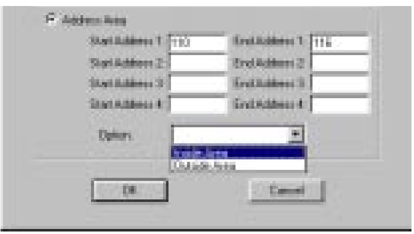
When this menu item is selected, a dialog box appears allowing selection of a trace mode.



Normal trace mode

Select a trace mode using the radio button.
Enter addresses and/or select an option and then clock [OK].

Single-delay trace mode



Address-area trace mode

td (trace data display)

Function

This command displays the trace information that has been sampled into the ICE63's trace memory.

Format

(1) >td [<cycle>]↵ **(direct input mode)**

(2) >td↵ **(guidance mode)**
Start point ?: (ENTER from the latest) <num>↵
(Trace data is displayed)
>

<cycle>: Start cycle number of trace data; decimal (from 0 to 8,191)

Display

The following lists the contents of trace information:

trace cycle:	Trace cycle (decimal). The last information taken into the trace memory becomes 00001.
fetch addr:	Fetch address (hexadecimal).
fetch code disasm:	Fetch code (hexadecimal) and disassembled content.
register:	Values of A, B, X, and Y registers after cycle execution (hexadecimal).
flag:	States of E, I, C, and Z flags after cycle execution (binary).
data:	Accessed data memory address (hexadecimal), read/write (denoted by r or w at the beginning of data), and data (1-digit hexadecimal for 4-bit access; 4-digit hexadecimal for 16-bit access).
SP:	Stack access (1 for SP1 access; 2 for SP2 access).
trace in:	Input to TRCIN pin (denoted by L when low-level signal is input).

(1) When [Trace] window is opened:

When the ***td*** command is input without <cycle>, the [Trace] window redisplays the latest data; when the ***td*** command is input with <cycle>, the trace data starting from <cycle> is displayed in the [Trace] window.
The display contents of the [Trace] window is updated after an execution of the target program.
All trace data can be displayed by scrolling the window.

```
Trace
trace fetch    fetch                register     flag   data         trace
cycle addr   code disasm          A  B   X    Y    EICZ addr data  SP    in
00011 000A   1990 adc   [%x]+,0x00  F  1  0005 AAAA 0000 0004 wC
00010 ----   ---- ----------        F  1  0006 AAAA 0000 0005 r9
00009 000B   1990 adc   [%x]+,0x00  F  1  0006 AAAA 0000 0005 w9
00008 ----   ---- ----------        F  1  0007 AAAA 0001 0006 r0
00007 000C   1980 adc   [%x],0x00   F  1  0007 AAAA 0001 0006 w0
00006 ----   ---- ----------        F  1  0007 AAAA 0000 0007 r3
00005 000D   1FF8 ret               F  1  0007 AAAA 0000 0007 w3
00004 0118   00FD jr    0xfd        F  1  0007 AAAA 0000 012C rAAAA 1
00003 0116   08FE ldb   %ext,0xfe   F  1  0007 AAAA 0000 ---- --
00002 0117   02EF calr  0xef        F  1  0007 AAAA 1000 ---- --
00001 0007   0800 ldb   %ext,0x00   F  1  0007 AAAA 0000 0128 w0118 1
```

(2) When [Trace] window is closed:

When the ***td*** command is input without <cycle>, the debugger displays 11 lines of the latest trace data in the [Command] window. When the ***td*** command is input with <cycle>, the debugger displays 11 lines of the trace data from <cycle> in the [Command] window.

```
>td↵
Start point ?:(ENTER from the latest)↵
trace fetch    fetch                  register    flag   data         trace
cycle addr  code disasm           A B   X    Y    EICZ addr data  SP   in
00011 0118  00FD jr   0xfd        F 1 0007 AAAA 0000 012C rAAAA 1
00010 0116  08FE ldb  %ext,0xfe   F 1 0007 AAAA 0000 ---- --
00009 0117  02EF calr 0xef        F 1 0007 AAAA 1000 ---- --
00008 0007  0800 ldb  %ext,0x00   F 1 0007 AAAA 0000 0128 w0118 1
00007 0008  0A04 ldb  %xl,0x04    F 1 0007 AAAA 1000 ---- --
00006 0009  1911 add  [%x]+,0x01  F 1 0004 AAAA 0000 ---- --
00005 ----  ---- ----------       F 1 0005 AAAA 0000 0004 rD
00004 000A  1990 adc  [%x]+,0x00  F 1 0005 AAAA 0000 0004 wE
00003 ----  ---- ----------       F 1 0006 AAAA 0000 0005 r5
00002 000B  1990 adc  [%x]+,0x00  F 1 0006 AAAA 0000 0005 w5
00001 ----  ---- ----------       F 1 0007 AAAA 0000 0006 rE
>td 10↵
trace fetch    fetch                  register    flag   data         trace
cycle addr  code disasm           A B   X    Y    EICZ addr data  SP   in
00020 0009  1911 add  [%x]+,0x01  F 1 0004 AAAA 0000 ---- --
00019 ----  ---- ----------       F 1 0005 AAAA 0000 0004 rC
00018 000A  1990 adc  [%x]+,0x00  F 1 0005 AAAA 0000 0004 wD
00017 ----  ---- ----------       F 1 0006 AAAA 0000 0005 r5
00016 000B  1990 adc  [%x]+,0x00  F 1 0006 AAAA 0000 0005 w5
00015 ----  ---- ----------       F 1 0007 AAAA 0000 0006 rE
00014 000C  1980 adc  [%x],0x00   F 1 0007 AAAA 0000 0006 wE
00013 ----  ---- ----------       F 1 0007 AAAA 0000 0007 r4
00012 000D  1FF8 ret              F 1 0007 AAAA 0000 0007 w4
00011 0118  00FD jr   0xfd        F 1 0007 AAAA 0000 012C rAAAA 1
00010 0116  08FE ldb  %ext,0xfe   F 1 0007 AAAA 0000 ---- --
>
```

(3) During log output

When the command execution result is being output to a log file as specified by the ***log*** command, the trace data is displayed in the [Command] window and its contents are also output to the log file.
If the [Trace] window is closed, data is displayed in the same way as in (2) above.
If the [Trace] window is open, its contents are redisplayed. In this case, the same number of lines are displayed in the [Command] window as displayed in the [Trace] window.

(4) Successive display

When you execute the ***td*** command, the trace data can be displayed successively by entering the [Enter] key only until some other command is executed.
When you input the [Enter] key, the [Trace] window is scrolled forward one screen.
When displaying data in the [Command] window, 11 lines of data preceding the previously displayed cycle are displayed in the [Command] window (the same number of lines as displayed in the [Trace] window if the command is executed during log output).
The direction of display is such that each time you input the [Enter] key, data on older execution cycles is displayed (FORWARD). This direction can be reversed (BACKWARD) by entering the [B] key. To return the display direction to FORWARD, input the [F] key. If the [Trace] window is open, the direction in which the window is scrolled is also changed.

```
>td 100↵                    ... Started display in FORWARD.
   (Data on cycle Nos. 110 to 100 is displayed.)
>b↵                         ... Changed to BACKWARD.
   (Data on cycle Nos. 99 to 89 is displayed.)
>↵                          ... Continued display in BACKWARD.
   (Data on cycle Nos. 88 to 78 is displayed.)
>f↵                         ... Changed back to FORWARD.
   (Data on cycle Nos. 99 to 89 is displayed.)
>
```

Notes

- Specify the trace cycle No. within the range of 0 to 8,191. An error results if this limit is exceeded.
  `Error : Address out of range, use 0-8191`
- The trace memory receives new data until a break occurs. When the trace memory is filled, old data is overwritten by new data.
- For reasons of the ICE63's operation timing, the trace data at the boundary of operations, such as in the fetch cycle at which trace starts or the execution cycle at which trace ends, will not always be stored in memory.

GUI utility

[View | Trace] menu item

When this menu item is selected, the [Trace] window opens and displays the latest trace data.

ts (trace search)

Function

This command searches trace information from the trace memory under a specified condition. The search condition can be selected from three available conditions:

1. Search by executed address
   In this mode, you can specify a program memory address. The debugger searches the cycle in which the specified address is executed.
2. Search for a specified memory read cycle
   In this mode, you can specify a data memory address. The debugger searches the cycle in which data is read from the specified address.
3. Search for a specified memory write cycle
   In this mode, you can specify a data memory address. The debugger searches the cycle in which data is written to the specified address.

Format

(1) >ts <option> <address>↵ (direct input mode)

(2) >ts↵ (guidance mode)

1. pc address 2. data read address 3. data write address ...? <1 | 2 | 3>↵
Search address ?: <address>↵
(Search result is displayed)
>

<option>: Condition type (program address, data read address or data write address); pc/dr/dw
<address>: Search address; hexadecimal or symbol (IEEE-695 format only)

Display

The search results are displayed in the [Trace] window if it is opened; otherwise, the results are displayed in the [Command] window in the same way as for the ***td*** command.

Format (1)
```
>ts pc 116↵
Trace searching ... Done!
trace fetch    fetch                register    flag   data        trace
cycle addr  code disasm           A B   X     Y    EICZ addr data  SP   in
00010 0116  08FE ldb  %ext,0xfe   F 1 0007 AAAA 0000 ---- --
>
```

Format (2)
```
>ts↵
1.pc address  2.data read address  3.data write address ...? 1↵
Search address ?:116↵
Trace searching ... Done!
trace fetch    fetch                register    flag   data        trace
cycle addr  code disasm           A B   X     Y    EICZ addr data  SP   in
00010 0116  08FE ldb  %ext,0xfe   F 1 0007 AAAA 0000 ---- --
>
```

When command execution results are being output to a log file by the ***log*** command, the search results are displayed in the [Command] window as well as output to the log file even when the [Trace] window is opened.

Note

The address specified for search must be within the range of the program/data memory area available for each microcomputer model.

An error results if the input one is not a hexadecimal number or not a valid symbol.

```
Error : invalid value     (no such symbol / symbol type error)
```

An error results if the limit is exceeded for program memory address.

```
Error : Address out of range, use 0-0xXXXX
```

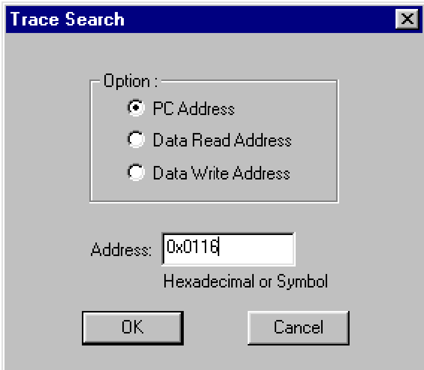
An error results if the limit is exceeded for data memory address.

```
Error : Address out of range, use 0-0xFFFF
```

GUI utility

[Trace | Trace Search ...] menu item

When this menu item is selected, a dialog appears for setting a search condition.



Select a option using the radio button and enter an address in the text box, then click [OK].

tf (trace file)

Function

This command saves the specified range of the trace information displayed in the [Trace] window by the *td* or *ts* command to a file.

Format

(1) >tf [<cycle1> [<cycle2>]] <file name>↵ (direct input mode)

(2) >tf↵ (guidance mode)

```
Start cycle number  (max  8191)  ? : <cycle1>↵
End cycle number    (min     0)  ? : <cycle2>↵
File Name    ? : <file name>↵
>
```

<cycle1>: Start cycle number; decimal (max 8,191)
<cycle2>: End cycle number; decimal (min 0)
<file name>: Output file name (path can also be specified)

Examples

Format (1)

```
>tf trace.trc↵
8191-8000
8000-7000
    :
1000-   1
OK!
>
```

... Saves all trace information extracted by the *td* command.

Format (2)

```
>tf↵
Start cycle number (max 8191) ? :1000↵
End cycle number   (min    0) ? :1↵
File name    ? :test.trc↵
1000-   1
OK!
>
```

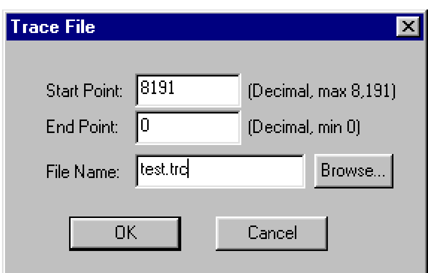
Notes

- If an existing file is specified, the file is overwritten with the new data.
- The default value of <cycle1> is the last location, and the default value of <cycle2> is "1".

GUI utility

[Trace | Trace File ...] menu item

When this menu item is selected, a dialog box appears allowing specification of the parameters.



Enter a start cycle number, end cycle number and a file name, then click [OK].
To save all the trace information, leave the [Start cycle number] and [End cycle number] boxes blank.
The file name can be selected using a standard file selection dialog box that appears by clicking [Browse...].

8.9.15 Coverage

CV (coverage)

Function

This command displays coverage information (addresses where the program is executed).
The coverage information is displayed in the [Command] window.

Format

>cv [<address1> [<address2>]]↵ (direct input mode)

<address1>: Start address; hexadecimal or symbol (IEEE-695 format only)
<address2>: End address; hexadecimal or symbol (IEEE-695 format only)
Condition: 0 ≤ address1 ≤ address2 ≤ last program memory address

Examples

```
>cv 100 1ff↵              ... Displays the executed addresses within the range from 0x100 to 0x1ff.
Coverage Infomation:
    1: 0100..0102
    2: 0110..0118
>cv↵                      ... Displays all the executed addresses.
Coverage Infomation:
    1: 0000..000d
    2: 0100..0102
    3: 0110..0118
>
```

Notes

- If the *cv* command is input without <address1> and <address2>, coverage information in all address is displayed; if both <address1> and <address2> are specified, coverage information within the specified address range is displayed; if just <address1> is specified, the end address is treated as the maximum program address and coverage information within that range is displayed.
- The addresses specified here must be within the range of the program memory area available with each microcomputer model.
  An error results if the input one is not a hexadecimal number or a valid symbol.
  `Error : invalid value     (no such symbol / symbol type error)`
  An error results if the limit is exceeded.
  `Error : Address out of range, use 0-0xXXXX`
- An error results if the start address is larger than the end address.
  `Error : end address < start address`

GUI utility

None

CVC (coverage clear)

Function
This command clears the coverage information.

Format
>cvc↵ (direct input mode)

GUI utility
None

8.9.16 Command File

com (execute command file)

Function

This command reads a command file and executes the debug commands written in that file. You can execute the commands successively, or set an interval between each command execution.

Format

(1) >com <file name> [<interval>]↵ **(direct input mode)**

(2) >com↵ **(guidance mode)**

File name ? <file name>↵
Execute commands 1. successively 2. with wait ...? <1 | 2>↵
Interval (0 - 256 seconds) : <interval>↵ (appears only when "2. With wait" is selected)
>*(Display execution progress)*

<file name>: Command file name (path can also be specified)
<interval>: Interval (wait seconds) between each command; decimal (0–256)

Examples

Format (1)

```
>com batch1.cmd↵
>.....                    ... Commands in "batch1.com" are executed successively.
```

Format (2)

```
>com↵
File name   ? test.cmd↵
Execute commands   1. successively   2. with wait   ...? 2↵
Wait time (0 - 256 seconds) : 2↵
>.....                    ... 2 sec. of interval is inserted after each command execution.
```

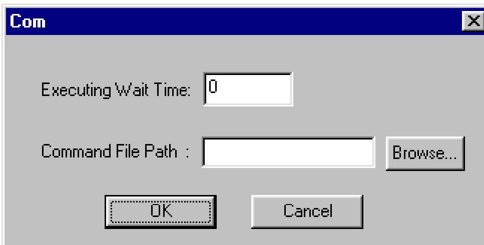
Notes

- Any contents other than commands cannot be written in the command file.
- An error results if the file you specified does not exist.
  `Error : Cannot open file`
- Another command file can be read from a command file. However, the nesting of command files is limited to a maximum of 5 levels. An error results if a *com* (or *cmw*) command at the sixth level is encountered, the commands in the file specified by that *com* (or *cmw*) command will not be executed, but the subsequent execution of the commands in upper level files will be executed continuously.
  `Error : Maximum nesting level(5) is exceeded, cannot open file`
- If you specify an interval more than 256 seconds, it is set to 256 by default.
- Use the hot key ([CTRL]+[Q]) to stop executing a command file.

GUI utility

[Run | Command File ...] menu item

When this menu item is selected, a dialog box appears allowing selection of a command file.



Enter an interval and a file name, then click [OK]. The file name can be selected using a standard file selection dialog box that appears by clicking [Browse...].

cmw (execute command file with wait)

Function

This command reads a command file and executes the debug commands written in that file at predetermined time intervals.
The execution interval of each command can be set in a range of 1 to 256 seconds (in 1-second increments) using the ***md*** command. In the initial debugger settings, the execution interval is 1 second.

Format

(1) >cmw <file name>↵ **(direct input mode)**

(2) >cmw↵ **(guidance mode)**
File name ? <file name>↵
>*(Display execution progress)*

<file name>: Command file name (path can also be specified)

Examples

Format (1)
```
>cmw batch1.cmd↵
>.....
```

Format (2)
```
>cmw↵
File name   ? test.cmd↵
>.....
```

Notes

- Any contents other than commands cannot be written in the command file.
- An error results if the file you specified does not exist.
  ```
  Error : Cannot open file
  ```
- Another command file can be read from a command file. However, the nesting of command files is limited to a maximum of 5 levels. An error results if a ***cmw*** (or ***com***) command at the sixth level is encountered, the commands in the file specified by that ***cmw*** (or ***com***) command will not be executed, but the subsequent execution of the commands in upper level files will be executed continuously.
  ```
  Error : Maximum nesting level(5) is exceeded, cannot open file
  ```
- If the ***cmw*** command is written in the command file that you want to be read by the ***com*** command, all other commands following that command in the file (even when a ***com*** command is included) will be executed at predetermined time intervals.
- Use the hot key ([CTRL]+[Q]) to stop executing a command file.

GUI utility

None
However, the same function as the ***cmw*** can be executed using [Command File...] in the [Run] menu (see the ***com*** command).

rec (record commands to a file)

Function

This command records all debug commands following this command to a specified command file.

Format

(1) >rec <file name>↵ **(direct input mode)**

(2) >rec↵ **(guidance mode)** ...See Examples for guidance.

<file name>: Command file name (path can also be specified)

Examples

(1) First rec execution after debugger starts up

```
>rec↵
File name   ? sample.cmd↵
1. append   2. clear and open   ...? 2↵       ...Displayed If the file is already exists.
>
```

(2) "rec" command input in the second and following sessions

```
>rec↵
Set to record off mode.        ...Record function toggles when rec is input.
.....
>rec↵
Set to record on mode.
```

Notes

- In record on mode, besides the commands directly input in the [Command] window, the commands executed by selecting from a menu or with a tool bar button (except the [Help] menu commands) are also displayed in the [Command] window, and output to the specified file.
  If you modify the register value or data memory contents by direct editing in the [Register] or [Data] window, or set breakpoints in the [Source] window by double-clicking the mouse, the corresponding commands are also displayed in the [Command] window, and output to the specified file.
- At the first time, you should specify the file name to which all debug commands following the *rec* command will be output.
- Once an output command file is opened, the recording is suspended and resumed (toggled) every time you input the *rec* command. This toggle operation remains effective until you terminate the debugger. If you want to record following commands to another file, you can use format (1) to specify the file name, then current output file is closed and all following commands will be recorded in the newly specified file.
- If you want to execute some commands frequently, you can record them to a file at the first execution, and then use the *com* or *cmw* command to execute that command file you made.

GUI utility

[Option | Record ...] menu item

When this menu item is selected, a standard file selection dialog box appears for specifying a command recording file. If the recording function has been activated, a dialog box appears allowing selection of either record-off mode or record-on mode. A new recording file can also be specified using the [New...] button.

8.9.17 log

log (log)

Function

This command saves the input commands and the execution results to a file.

Format

(1) >log <file name>↵ **(direct input mode)**

(2) >log↵ **(guidance mode)** ...See Examples for guidance.

<file name>: Log file name (path can also be specified)

Examples

(1) First log execution after debugger starts up

```
>log↵
File name   ? debug1.log↵
1. append   2. clear and open   ...? 2↵       ...Displayed If the file is already exists.
>
```

(2) "log" command input in the second and following sessions

```
>log↵
Set to log off mode.           ...Logging function toggles when log is input.
.....
>log↵
Set to log on mode.
```

Notes

- In log on mode, the contents displayed in the [Command] window are written as displayed directly to the log file.
  The commands executed by selecting from a menu or with a tool bar button are displayed in the [Command] window. However, the [Help] menu and button commands are not displayed. If you modify the register value or data memory contents by direct editing in the [Register] or [Data] window, or set breakpoints in the [Source] window by double-clicking the mouse, the corresponding commands and the execution results are also displayed in the [Command] window, and output to the specified file.

  The displayed contents of the [Source], [Data], [Trace] or [Register] window produced by command execution are displayed in the [Command] window as well. The on-the-fly information is also displayed. However, the updated contents of each window after some execution, as well as the contents of each window scrolled by scroll bar or arrow keys, are not displayed.

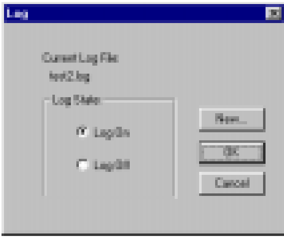
- At the first time, you should specify the file name to which all following debug commands and execution results will be output.

- Once a log file is open, log output is suspended and resumed (toggled) every time you input the ***log*** command. This toggle operation remains effective until you terminate the debugger. If you want to specify a new log file, you can use format (1) to specify the file name, then current log file is closed and following commands and results will be output to the newly specified file.

GUI utility

[Option | Log ...] menu item

When this menu item is selected, a standard file selection dialog box appears for specifying a log file.
If the logging function has been activated, a dialog box appears allowing selection of either log-off mode or log-on mode. A new log file can also be specified using the [New...] button.



8.9.18 Map Information

ma (map information)

Function

This command displays the map information that is set by a parameter file.

Format

>ma↵ **(direct input mode)**

Example

After the command is input, the system displays the chip name, version of the parameter file, and map information in each area. When you input the [Enter] key here, the system goes on and displays the map information in the I/O area and LCD area.

```
>ma↵
Chip name               : 63A08
Parameter file version  : 02
Program area            : 0000 - 1FFF
Data ram area           : 0000 - 07FF
Data rom area           : 8000 - 8FFF
LCD area                : F000 - F2BF
External memory area    : F800 - F8FF
IO area                 : FF00 - FFFF
Size of FO area         : 32
Size of SO1 area        : 0
Size of SO2 area        : 256
Size of MLA area        : 1296
>↵
IO Area
     01234567 89ABCDEF 01234567 89ABCDEF 01234567 89ABCDEF 01234567 89ABCDEF
FF00 mmmm-mmm --mmmmmm -------- -------- -mmm-mmm -------- mmmmmmmm ------mm
FF40 -mmm-mmm -mmm-mmm -----mmm -------- -----mmm ----mmmm ----mmmm -mmm-mmm
FF80 mmmmmmmm mmmmmmmm mmmmmmmm mmmmmmmm mmmmmmmm mmmmmmmm mmmmmmmm mmmmmmmm
FFC0 mmmmmmmm ----mmmm -------- -------- mmmmmmmm -------- mmmmmmmm --------
>↵
LCD Area
     01234567 89ABCDEF 01234567 89ABCDEF 01234567 89ABCDEF 01234567 89ABCDEF
F000 mmmmmmmm mmmmmmmm mmmmmmmm mmmmmmmm mmmmmmmm mmmmmmmm mmmmmmmm mmmmmmmm
F040 mmmmmmmm mmmmmmmm mmmmmmmm mmmmmmmm mmmmmmmm mmmmmmmm mmmmmmmm mmmmmmmm
F080 mmmmmmmm mmmmmmmm -------- -------- -------- -------- -------- --------
F0C0 -------- -------- -------- -------- -------- -------- -------- --------
F100 mmmmmmmm mmmmmmmm mmmmmmmm mmmmmmmm mmmmmmmm mmmmmmmm mmmmmmmm mmmmmmmm
F140 mmmmmmmm mmmmmmmm mmmmmmmm mmmmmmmm mmmmmmmm mmmmmmmm mmmmmmmm mmmmmmmm
F180 mmmmmmmm mmmmmmmm -------- -------- -------- -------- -------- --------
F1C0 -------- -------- -------- -------- -------- -------- -------- --------
F200 -m-m-m-m -m-m-m-m -m-m-m-m -m-m-m-m -m-m-m-m -m-m-m-m -m-m-m-m -m-m-m-m
F240 -m-m-m-m -m-m-m-m -m-m-m-m -m-m-m-m -m-m-m-m -m-m-m-m -m-m-m-m -m-m-m-m
F280 -m-m-m-m -m-m-m-m -------- -------- -------- -------- -------- --------
>
```

* When displaying the map information of the I/O and LCD areas, the mapped addresses are marked by the letter "m".

GUI utility

None

8.9.19 Mode Setting

md (mode)

Function

This command sets the debugger modes described below.

1. **Displaying on-the-fly information**
   You can choose the display interval of the on-the-fly information from 0 to 5 (times) per second. When 0 is chosen, the on-the-fly information will not be displayed.

2. **Measurement mode for the execution cycle counter**
   This mode can be selected from the actual execution-time measurement mode (indicated in microseconds) or the bus cycle mode (indicated in terms of the number of cycles executed).

3. **Interrupt mode for step execution**
   You can choose to enable or disable interrupts during single-stepping.

4. **Single-step display mode**
   You can choose to display the execution results of each step or only the last step during single-step operation. The register values are updated when their contents are displayed in the [Register] window; they are displayed in the [Command] window if the [Register] window is closed.
   If the [Source] window is open, the displayed lines are marked with an arrow as they are executed according to the setting of this mode.

5. **Mode of execution cycle counter**
   This can be selected from hold mode or reset mode. In reset mode, the counter value is reset to 0 each time you enter a program execution command (including execution by the [Enter] key).
   The value of the execution cycle counter is also reset when you execute a ***gr*** command, switch this mode or the counter measurement mode, or execute an ***rst*** command.

6. **Illegal instruction check mode**
   When loading a program file into the computer using the ***lf*** or ***lo*** command, you can choose whether or not you want illegal instructions to be checked.
   This check is disabled when rewriting the program memory with a ***pe*** or ***pf*** command.

7. **cmw command wait time**
   A ***cmw*** command wait time can be set in the range of 1 to 256 seconds (in 1-second increments).

Default values of debugger modes

Mode	Default setting
On-the-fly function	Twice per second
Counter measurement mode	Bus cycle
Interrupt at stepping	Not allowed
Step display	Each step
Execution cycle counter reset	Hold
Illegal instruction check	Checked
cmw wait time	1 second

Format

(1) >md <option> <num> [... <option> <num>]↵ (direct input mode)

(2) >md↵ (guidance mode)

```
Current settings
On the fly interval  0 - 5 times/sec             ...? Current setting : <0 ... 5>↵
Counter unit         1.t ime      2. cycle       ...? Current setting : <1 | 2>↵
Interrupt at step    1. allowed   2. not allowed ...? Current setting : <1 | 2>↵
Step display mode    1. each      2. last        ...? Current setting : <1 | 2>↵
Counter mode         1. reset     2. hold        ...? Current setting : <1 | 2>↵
Illegal instruction  1. check     2. no check    ...? Current setting : <1 | 2>↵
Cmw wait time        1 - 256 s                   ...? Current setting : <1 ... 256>↵
>
```

<option>:	<num>:	
-f (on the fly interval)	0–5 times/sec	
-u (couter unit)	1. Time	2. Cycle
-i (interrupt at step)	1. Allowed	2. Not allowed
-s (step display mode)	1. Each	2. Last
-c (counter mode)	1. Reset	2. Hold
-il (illegal instruction)	1. Check	2. No check
-cm (cmw wait time)	1–256 sec	

Examples

```
>md -u 1↵      ...Sets the execution cycle counter in time measurement mode.
>md↵
On the fly interval : 2 times/sec
Counter unit        : time
Interrupt at step   : not allowed
Step display mode   : each
Counter mode        : hold
Illegal instruction : check
Cmw wait time       : 1 s

On the fly interval 0 - 5 times/sec          ...? 2 times/sec : 5↵
Counter unit        1.time     2.cycle       ...? time        : 2↵
Interrupt at step   1.allowed  2.not allowed ...? not allowed : 1↵
Step display mode   1.each     2.last        ...? each        : 2↵
Counter mode        1.reset    2.hold        ...? hold        : ↵
Illegal instruction 1.check    2.no check    ...? check       : ↵
Cmw wait time       1 - 256 s                ...?   1       s : 3↵
>
```

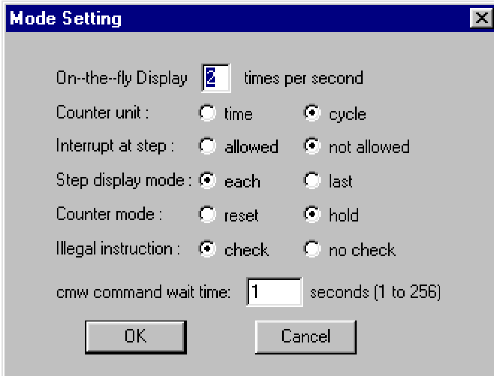
Notes

- The actual interval of the on-the-fly display is obtained from the expression below.
(1 [sec] / Count set) + (Overhead of the PC, RS232C interface and ICE63 [sec]) = display interval [sec]
The overhead varies depending on the performance of the PC and baud rate of the RS232C interface.
Be aware that there is a 0.05 sec to 0.1 sec overhead in this system.
- In guidance mode, the following keyboard inputs have special meaning:
"q↵" … Command is terminated. (finish inputting and start execution)
"^↵" … Return to previous item.
"↵" … Input is skipped. (keep current value)

GUI utility

[Option | Mode Setting…] menu item

When this menu item is selected, a dialog box appears allowing selection of each mode.



Select the mode using the check boxes or enter the number interval settings, and then click [OK].

8.9.20 Quit

q (quit)

Function

This command quits the debugger.

Format

>q↵ **(direct input mode)**

GUI utility

[File | Exit] menu item

Selecting this menu item terminates the debugger.

8.9.21 Help

? (help)

Function

This command displays the input format of each command.

Format

(1) ? (direct input mode)
(2) ? <n> (direct input mode)
(3) ? <command> (direct input mode)

<n>: Command group number; decimal
<command>: Command name
Condition: 1 ≤ n ≤ 6

Examples

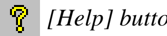
When you input the command in Format 1 or 2, the system displays a list of commands classified by function. Use the command in Format 3 if you want to display the input format of each individual command.

```
>?↵
group 1: program, data & register ... pe,pf,pm,a(as) / dd,de,df,dm,dw / od / rd,rs
group 2: execution & break .......... g,gr,s,n,rst / bp,bc(bpc),bd,bdc,br,brc,bs,bsc,bsp,bl,bac
group 3: source & symbol ............ u,sc,m / sy
group 4: file & flash rom ........... lf,lo / lfl,sfl,efl
group 5: trace & coverage ........... tm,td,ts,tf / cv,cvc
group 6: others ..................... com,cmw,rec,log / ma,md,q,?
 Type "? <group #>" to show group or type "? <command>" to get usage of the command
>? 1↵
group 1: program, data & register
pe (program enter), pf (program fill), pm (program move), a/as (inline assemble),
dd (data dump), de (data enter), df (data fill), dm (data move), dw (data watch),
od (option dump),
rd (register display), rs (register set)
 Type "? <command>" to get usage of the command
>? pe↵
pe (program enter): change program memory
usage: pe [address]                   ... change program with guidance
       pe address code1 [... code8]   ... change program with specified code
>
```

GUI utility

[Help | Contents…] menu item, [Help] button

When this menu item or button is selected, the [Help] window opens to show help topics.



[Help] button

8.10 Status/Error/Warning Messages

1. ICE status messages

Status message	Content of message
Break by PC break	Break caused by PC breakpoint
Break by data break	Break caused by data break condition
Break by register break	Break caused by register break condition
Break by sequential break	Break caused by sequential break condition
Key Break	Break caused by pressing [ESC] key or [Key break] button
Break by accessing no map program area	Break caused by accessing undefined program-memory area
Break by accessing no map data area	Break caused by accessing undefined data-memory area
Break by accessing ROM area	Break caused by writing to data ROM area
Out of SP1 area	Break caused by accessing outside SP1 stack area
Out of SP2 area	Break caused by accessing outside SP2 stack area
Break by external break	Break caused by signal input to ICE63's BRKIN pin

2. ICE error messages

Error message	Content of message
communication error	Communication error other than time-out (overrun, framing, or BCC error)
CPU is running	Target is running.
ICE is busy	ICE63 is busy processing a job.
ICE is free run mode	ICE63 is operating in free-run mode.
ICE is maintenance mode	ICE63 is placed in maintenance mode.
no map area, XXXX	No-map area is specified for accessing.
not defined ID, XXXX	ICE63's respond ID is invalid.
on tracing	System is tracing execution data.
reset time out	CPU cannot be reset (for more than 1 second).
target down	PRC board does not operate correctly or remains reset.
Time Out!	Communication time-out

3. Flash memory error messages

Error message	Content of message
flash memory error, XXXX	Writing or erasing flash memory has failed at XXXX.
flash ROM is protected	Flash memory is protected against access.
format error	Flash memory is not mapped.
Map information is not the same	Map information loaded from parameter file does not match that in the parameter file.
verify error, XXXX	Verify error has occurred when data was written to flash memory.

4. Command error/warning messages

Error message	Content of message (Commands involved)
Address out of range, use 0–0xXXXX	The specified program memory address is out of range. (a/as, pe, pf, pm, sc, m, u, g, gr, bp, bc, bs, tm, ts, cv)
Address out of range, use 0–0xFFFF	The specified data memory address is out of range. (dd, de, df, dm, dw, bd, ts)
Cannot load program/ROM data, check ABS file	Failed to load program/ROM data; some file other than IEEE-695 executable format was specified. (lf)
Cannot open file	The file cannot be opened. (lf, lo, com, cmw, log, rec)
Data out of range, use 0–0xF	The specified number is out of the data range. (de, df)
Different chip type, cannot load this file	A different ICE parameter is used in the file. (lf)
end address < start address	The start address is larger than the end address. (pf, pm, df, dm, bd, cv)
error file type (extension should be CMD)	The specified file extension is invalid. (com, cmw)
FO address out of range, use 0–0xEF	FO address is invalid. (od)
illegal code	The input code is not available. (pe, pf)
illegal mnemonic	The input mnemonic is invalid for E0C63000. (a/as)
Incorrect number of parameters	The parameter number is incorrect. (All commands)
Incorrect option, use -f/-u/-i/-s/-c/-il/-cm	An invalid mode setting option was specified. (md)
Incorrect r/w option, use r/w/*	An illegal R/W option was specified. (bd)
Incorrect register name, use A/B/X/Y/F	An invalid register name was specified. (br)
Incorrect register name, use PC/A/B/X/Y/F/SP1/SP2/EXT/Q	The specified register name is invalid. (rs)

Error message	Content of message (Commands involved)
Input address does not exist	Attempt is made to clear a break address that has not been set. (bp)
invalid command	This is an invalid command. (All commands)
invalid data pattern	The input data pattern is invalid. (bd, br)
invalid file name	The file name (extension) is invalid. (lo)
invalid value	The input data, address or symbol is invalid. (All commands)
Maximum nesting level(5) is exceeded, cannot open file	Nesting of the com/cmw command exceeds the limit. (com, cmw)
MLA address out of range, use 0–0xFFF	MLA address is invalid. (od)
no such symbol	There is no such symbol. (All symbol support commands)
no symbol information	No symbol information is available since the ".ABS" file has not been loaded. (sy)
Number of passes out of range, use 0–4095	The specified pass count for sequential break is out of range. (bs)
Number of steps out of range, use 0–65535	The specified step count is out of range. (s, n)
SO address out of range, use 0–0x1FFF	SO address is invalid. (od)
SP1 address out of range, use 0–0x3FF	The specified SP1 address is out of range. (bsp)
SP2 address out of range, use 0–0xFF	The specified SP2 address is out of range. (bsp)
symbol type error	The specified symbol type (program/data) is incorrect. (All symbol support commands)

Warning message	Content of message (Commands involved)
Break address already exists	Attempt is made to set an already-set break address. (bp)
Identical break address input	Input command contains identical address.
round down to multiple of 4	Watch data address is invalid. (dw)

EPSON International Sales Operations

AMERICA

EPSON ELECTRONICS AMERICA, INC.

- HEADQUARTERS -
1960 E. Grand Avenue
El Segundo, CA 90245, U.S.A.
Phone: +1-310-955-5300 Fax: +1-310-955-5400

- SALES OFFICES -

West
150 River Oaks Parkway
San Jose, CA 95134, U.S.A.
Phone: +1-408-922-0200 Fax: +1-408-922-0238

Central
101 Virginia Street, Suite 290
Crystal Lake, IL 60014, U.S.A.
Phone: +1-815-455-7630 Fax: +1-815-455-7633

Northeast
301 Edgewater Place, Suite 120
Wakefield, MA 01880, U.S.A.
Phone: +1-781-246-3600 Fax: +1-781-246-5443

Southeast
3010 Royal Blvd. South, Suite 170
Alpharetta, GA 30005, U.S.A.
Phone: +1-877-EEA-0020 Fax: +1-770-777-2637

EUROPE

EPSON EUROPE ELECTRONICS GmbH

- HEADQUARTERS -
Riesstrasse 15
80992 Muenchen, GERMANY
Phone: +49-(0)89-14005-0 Fax: +49-(0)89-14005-110

- GERMANY -
SALES OFFICE
Altstadtstrasse 176
51379 Leverkusen, GERMANY
Phone: +49-(0)217-15045-0 Fax: +49-(0)217-15045-10

- UNITED KINGDOM -
UK BRANCH OFFICE
2.4 Doncastle House, Doncastle Road
Bracknell, Berkshire RG12 8PE, ENGLAND
Phone: +44-(0)1344-381700 Fax: +44-(0)1344-381701

- FRANCE -
FRENCH BRANCH OFFICE
1 Avenue de l' Atlantique, LP 915 Les Conquerants
Z.A. de Courtaboeuf 2, F-91976 Les Ulis Cedex, FRANCE
Phone: +33-(0)1-64862350 Fax: +33-(0)1-64862355

ASIA

- CHINA -
EPSON (CHINA) CO., LTD.
28F, Beijing Silver Tower 2# North RD DongSanHuan
ChaoYang District, Beijing, CHINA
Phone: 64106655 Fax: 64107320

SHANGHAI BRANCH
4F, Bldg., 27, No. 69, Gui Jing Road
Caohejing, Shanghai, CHINA
Phone: 21-6485-5552 Fax: 21-6485-0775

- HONG KONG, CHINA -
EPSON HONG KONG LTD.
20/F., Harbour Centre, 25 Harbour Road
Wanchai, HONG KONG
Phone: +852-2585-4600 Fax: +852-2827-4346
Telex: 65542 EPSCO HX

- TAIWAN, R.O.C. -
EPSON TAIWAN TECHNOLOGY & TRADING LTD.
10F, No. 287, Nanking East Road, Sec. 3
Taipei, TAIWAN, R.O.C.
Phone: 02-2717-7360 Fax: 02-2712-9164
Telex: 24444 EPSONTB

HSINCHU OFFICE
13F-3, No. 295, Kuang-Fu Road, Sec. 2
HsinChu 300, TAIWAN, R.O.C.
Phone: 03-573-9900 Fax: 03-573-9169

- SINGAPORE -
EPSON SINGAPORE PTE., LTD.
No. 1 Temasek Avenue, #36-00
Millenia Tower, SINGAPORE 039192
Phone: +65-337-7911 Fax: +65-334-2716

- KOREA -
SEIKO EPSON CORPORATION KOREA OFFICE
50F, KLI 63 Bldg., 60 Yoido-Dong
Youngdeungpo-Ku, Seoul, 150-010, KOREA
Phone: 02-784-6027 Fax: 02-767-3677

- JAPAN -
SEIKO EPSON CORPORATION
ELECTRONIC DEVICES MARKETING DIVISION

Electronic Device Marketing Department
IC Marketing & Engineering Group
421-8, Hino, Hino-shi, Tokyo 191-8501, JAPAN
Phone: +81-(0)42-587-5816 Fax: +81-(0)42-587-5624

ED International Marketing Department I (Europe & U.S.A.)
421-8, Hino, Hino-shi, Tokyo 191-8501, JAPAN
Phone: +81-(0)42-587-5812 Fax: +81-(0)42-587-5564

ED International Marketing Department II (Asia)
421-8, Hino, Hino-shi, Tokyo 191-8501, JAPAN
Phone: +81-(0)42-587-5814 Fax: +81-(0)42-587-5110

In pursuit of **“Saving” Technology**, Epson electronic devices.
Our lineup of semiconductors, liquid crystal displays and quartz devices
assists in creating the products of our customers’ dreams.
Epson IS energy savings.

■ Electronic devices information on Epson WWW server

http://www.epson.co.jp

First issue AUGUST 1997, Printed JULY 1999 in Japan Ⓜ B