FUJITSU SEMICONDUCTOR
CONTROLLER MANUAL

CM41-00313-3E

F^2MC-16 FAMILY

SOFTUNE^{TM} WORKBENCH

USER'S MANUAL

F^{2}MC-16 FAMILY

SOFTUNE^{TM} WORKBENCH

USER'S MANUAL

FUJITSU LIMITED

PREFACE

■ What is the SOFTUNE Workbench?

SOFTUNE Workbench is support software for developing programs for the F^2MC-16 family of microprocessors / microcontrollers.

It is a combination of a development manager, simulator debugger, emulator debugger, monitor debugger, and an integrated development environment for efficient development.

Note:F^2MC is the abbreviation of FUJITSU Flexible Microcontroller.

■ Purpose of this manual and target readers

This manual explains functions of SOFTUNE Workbench.

This manual is intended for engineers designing several kinds of products using SOFTUNE Workbench.

■ Trademarks

SOFTUNE is a trademark of FUJITSU LIMIITED.

REALOS (REAL time Operating System) is a trademark of FUJITSU LIMITED.
The company names and brand names herein are the trademarks or registered trademarks of their respective owners.
Microsoft, Windows and Windows Media are either registered trademarks of Microsoft Corporation in the United States and/or other countries.

■ Organization of This Manual

This manual consists of the following 2 chapters.

CHAPTER 1 “BASIC FUNCTIONS”

This chapter describes the basic functions on the SOFTUNE^{TM} Workbench.

CHAPTER 2 “DEPENDENCE FUNCTIONS”

This chapter describes the functions dependent on each debugger.

- The contents of this document are subject to change without notice.
  Customers are advised to consult with FUJITSU sales representatives before ordering.
- The information, such as descriptions of function and application circuit examples, in this document are presented solely for the purpose of reference to show examples of operations and uses of FUJITSU semiconductor device; FUJITSU does not warrant proper operation of the device with respect to use based on such information. When you develop equipment incorporating the device based on such information, you must assume any responsibility arising out of such use of the information. FUJITSU assumes no liability for any damages whatsoever arising out of the use of the information.
- Any information in this document, including descriptions of function and schematic diagrams, shall not be construed as license of the use or exercise of any intellectual property right, such as patent right or copyright, or any other right of FUJITSU or any third party or does FUJITSU warrant non-infringement of any third-party's intellectual property right or other right by using such information. FUJITSU assumes no liability for any infringement of the intellectual property rights or other rights of third parties which would result from the use of information contained herein.
- The products described in this document are designed, developed and manufactured as contemplated for general use, including without limitation, ordinary industrial use, general office use, personal use, and household use, but are not designed, developed and manufactured as contemplated (1) for use accompanying fatal risks or dangers that, unless extremely high safety is secured, could have a serious effect to the public, and could lead directly to death, personal injury, severe physical damage or other loss (i.e., nuclear reaction control in nuclear facility, aircraft flight control, air traffic control, mass transport control, medical life support system, missile launch control in weapon system), or (2) for use requiring extremely high reliability (i.e., submersible repeater and artificial satellite).
  Please note that FUJITSU will not be liable against you and/or any third party for any claims or damages arising in connection with above-mentioned uses of the products.
- Any semiconductor devices have an inherent chance of failure. You must protect against injury, damage or loss from such failures by incorporating safety design measures into your facility and equipment such as redundancy, fire protection, and prevention of over-current levels and other abnormal operating conditions.
- If any products described in this document represent goods or technologies subject to certain restrictions on export under the Foreign Exchange and Foreign Trade Law of Japan, the prior authorization by Japanese government will be required for export of those products from Japan.

READING THIS MANUAL

■ Configuration of Page

In each section of this manual, the summary about the section is described certainly, so you can grasp an outline of this manual if only you read these summaries.

And the title of upper section is described in lower section, so you can grasp the position where you are reading now.

■ Product Names

In this manual and this product, product name is designated as follows:

The Microsoft® Windows® 2000 Professional operating system is abbreviated to Windows 2000.

The Microsoft® Windows® XP Professional operating system is abbreviated to Windows XP.

CONTENTS

CHAPTER 1 Basic Functions 1
1.1 Workspace Management Function 2
1.2 Project Management Function 3
1.3 Project Dependence 5
1.4 Make/Build Function 6
1.4.1 Customize Build Function 7
1.5 Include Dependencies Analysis Function 9
1.6 Functions of Setting Tool Options 10
1.7 Error Jump Function 11
1.8 Editor Functions 13
1.9 Storing External Editors 14
1.10 Storing External Tools 16
1.11 Macro Descriptions Usable in Manager 17
1.12 Setting Operating Environment 21
1.13 Debugger Types 22
1.14 Memory Operation Functions 23
1.15 Register Operations 24
1.16 Line Assembly and Disassembly 25
1.17 Symbolic Debugging 26
1.17.1 Referring to Local Symbols 28
1.17.2 Referring to C Variables 29

CHAPTER 2 Dependence Functions 31
2.1 Simulator Debugger 32
2.1.1 Instruction Simulation 34
2.1.2 Memory Simulation 35
2.1.3 I/O Port Simulation 36
2.1.4 Interrupt Simulation 37
2.1.5 Reset Simulation 38
2.1.6 Power-Save Consumption Mode Simulation 39
2.1.7 STUB Function 40
2.2 Emulator Debugger (MB2141) 41
2.2.1 Setting Operating Environment 42
2.2.2 Notes on Commands for Executing Program 52
2.2.3 On-the-fly Executable Commands 54
2.2.4 On-the-fly Memory Access 55
2.2.5 Events 57
2.2.6 Control by Sequencer 65
2.2.7 Real-time Trace 74
2.2.8 Measuring Performance 92
2.2.9 Measuring Coverage 96
2.2.10 Measuring Execution Time Using Emulation Timer 100
2.2.11 Sampling by External Probe 101

2.3 Emulator Debugger (MB2147-01) 103
2.3.1 Setting Operating Environment 104
2.3.2 Notes on Commands for Executing Program 114
2.3.3 On-the-fly Executable Commands 116
2.3.4 Break 118
2.3.5 Control by Sequencer 119
2.3.6 Real-time Trace 123
2.3.7 Measuring Performance 141
2.3.8 Measuring Coverage 145
2.3.9 Real-time Monitoring 149
2.3.10 Measuring Execution Time Using Emulation Timer 150
2.3.11 Power-on Debugging 152
2.3.12 RAM Checker 153
2.4 Emulator Debugger (MB2147-05) 157
2.4.1 Setting Operating Environment 158
2.4.2 Notes on Commands for Executing Program 166
2.4.3 On-the-fly Executable Commands 168
2.4.4 Real-time Trace 169
2.4.5 Execution Cycle Measurement by Cycle Counter 180
2.5 Monitor Debugger 181
2.5.1 Resources Used by Monitor Program 182
2.6 Abortion of Program Execution (SIM, EML, MON) 183
2.6.1 Instruction Execution Breaks (SIM, EML) 184
2.6.2 Monitoring Data Breaks (MB2147-01 EML) 186
2.6.3 Data Access Breaks (SIM, EML) 187
2.6.4 Software Break (MON) 188
2.6.5 Sequential Break (EML) 189
2.6.6 Guarded Access Breaks (SIM, EML) 190
2.6.7 Trace-Buffer-Full Break (SIM, EML) 191
2.6.8 Performance-Buffer-Full Break (EML) 192
2.6.9 Task Dispatch Break (SIM, EML, MON) 193
2.6.10 System Call Break (SIM, EML, MON) 194
2.6.11 Forced Break (SIM, EML) 195

CHAPTER 1

Basic Functions

This chapter describes the basic functions on the SOFTUNE Workbench.

1.1 Workspace Management Function

1.2 Project Management Function

1.3 Project Dependence

1.4 Make/Build Function

1.5 Include Dependencies Analysis Function

1.6 Functions of Setting Tool Options

1.7 Error Jump Function

1.8 Editor Functions

1.9 Storing External Editors

1.10 Storing External Tools

1.11 Macro Descriptions Usable in Manager

1.12 Setting Operating Environment

1.13 Debugger Types

1.14 Memory Operation Functions

1.15 Register Operations

1.16 Line Assembly and Disassembly

1.17 Symbolic Debugging

1.1 Workspace Management Function

This section explains the workspace management function of SOFTUNE Workbench.

■ Workspace

SOFTUNE Workbench uses workspace as a container to manage two or more projects including subprojects.

For example, a project that creates a library and a project that creates a target file using the project can be stored in one workspace.

■ Workspace Management Function

To manage two or more projects, workspace manages the following information:

-Project

-Active project

-Subproject

■ Project

The operation performed in SOFTUNE Workbench is based on the project. The project is a set of files and procedures necessary for creation of a target file. The project file contains all data managed by the project.

■ Active Project

The active project is basic to workspace and undergoes [Make], [Build], [Compile/Assemble], [Start Debug], and [Update Dependence] in the menu. [Make], [Build], [Compile/Assemble], and [Update Dependence] affect the subprojects within the active project.

If workspace contains some project, it always has one active project.

■ Subproject

The subproject is a project on which other projects depend. The target file in the subproject is linked with the parent project of the subproject in creating a target file in the parent project.

This dependence consists of sharing target files output by the subproject, so a subproject is first made and built. If making and building of the subproject is unsuccessful, the parent project of the subproject will not be made and built.

The target file in the subproject is however not linked with the parent project when:

-An absolute (ABS)-type project is specified as a subproject.

-A library (LIB)-type project is specified as a subproject.

■ Restrictions on Storage of Two or More Projects

Only one REALOS-type project can be stored in one workspace.

1.2 Project Management Function

This section explains the project management function of SOFTUNE Workbench.

■ Project Management Function

The project manages all information necessary for development of a microcontroller system. Especially, its major purpose is to manage information necessary for creation of a target file.

The project manages the following information:

- Project configuration
- Active project configuration
- Information on source files, include files, other object files, library files
- Information on tools executed before and after executing language tools (customize build function)

■ Project Format

The project file supports two formats: a 'workspace project format,' and an 'old project format.'

The differences between the two formats are as follows:

Workspace project format

- Supports management of two or more project configurations
- Supports use of all macros usable in manager
- Does not support early Workbench versions(*)

Old project format

- Supports management of just one project configuration
- Limited number of macros usable in manager

  For details, see Section 1.11 Macro Descriptions Usable in Manager.
- Supports early Workbench versions(*)

When a new project is made, the workspace project format is used.

When using an existing project, the corresponding project format is used.

If a project made by an early Workbench version(*) is used, a dialog asking whether to convert the file to the workspace project format is displayed. For details, refer to Section 2.13 of SOFTUNE WORKBENCH Operation Manual.

To open a project file in the workspace project format with an early Workbench version(*), it is necessary to convert the file to the old project format. For saving the file in other project formats, refer to Section 4.2.7 Save As of SOFTUNE WORKBENCH Operation Manual.

*: F^{2}MC-16: V30L26 or earlier

■ Project Configuration

The project configuration is a series of settings for specifying the characteristics of a target file, and making, building, compiling and assembling is performed in project configurations.

Two or more project configurations can be created in a project. The default project configuration name is Debug. A new project configuration is created on the setting of the selected existing project configuration. In the new project configuration, the same files as those in the original project configuration are always used.

By using the project configuration, the settings of programs of different versions, such as the optimization level of a compiler and MCU setting, can be created within one project.

In the project configuration, the following information is managed:

- Name and directory of target file
- Information on options of language tools to create target file by compiling, assembling and linking source files
- Information on whether to build file or not
- Information on setting of debugger to debug target file

■ Active Project Configuration

The active project configuration at default undergoes [Make], [Build], [Compile/Assemble], [Start Debug], and [Update Dependence].

The setting of the active project configuration is used for the file state displayed in the SRC tab of project window and includes files detected in the Dependencies folder.

Note:

If a macro function newly added is used in old project format, the macro description is expanded at the time of saving in old project format. For the macro description newly added, refer to Section 1.11 Macro Descriptions Usable in Manager.

1.3 Project Dependence

This section explains the project dependence of SOFTUNE Workbench.

■ Project Dependence

If target files output by other projects must be linked, a subproject is defined in the project required in [Project]-[Project Dependence] menu. The subproject is a project on which other projects depend.

By defining project dependence, a subproject can be made and built to link its target file before making and building the parent project.

The use of project dependence enables simultaneous making and building of two or more projects developed in one workspace.

A project configuration in making and building a subproject in [Project]-[Project Configuration]-[Build Configuration] menu can be specified.

1.4 Make/Build Function

This section explains the make/build function of SOFTUNE Workbench.

■ Make Function

Make function generates a target file by compiling/assembling only updated source files from all source files registered in a project, and then joining all required object files.

This function allows compiling/assembling only the minimum of required files. The time required for generating a target file can be sharply reduced, especially, when debugging.

For this function to work fully, the dependence between source files and include files should be accurately grasped. To do this, SOFTUNE Workbench has a function for analyzing include dependence. To perform this function, it is necessary to understand the dependence of a source file and include file. SOFTUNE Workbench has the function for analyzing the include file dependence. For details, see Section 1.5.

■ Build Function

Build function generates a target file by compiling/assembling all source files registered with a project, regardless of whether they have been updated or not, and then by joining all required object files. Using this function causes all files to be compiled/assembled, resulting in the time required for generating the target file longer. Although the correct target file can be generated from the current source files.

The execution of Build function is recommended after completing debugging at the final stage of program development.

Note:

When executing the Make function using a source file restored from backup, the integrity between an object file and a source file may be lost. If this happens, executing the Build function again.

1.4.1 Customize Build Function

This section describes the SOFTUNE Workbench to set the Customize Build function.

■ Customize Build function

In SOFTUNE Workbench, different tools can be operated automatically before and after executing the Assembler, Compiler, Linker, Librarian, Converter, or Configurator started at Compile, Assemble, Make, or Build.

The following operations can be performed automatically during Make or Build using this function:

- starting the syntax check before executing the Compiler,
- after executing the Converter, starting the S-format binary Converter (m2bs.exe) and converting Motorola S-format files to binary format files.

■ Setting Options

An option follows the tool name to start a tool from SOFTUNE Workbench. The options include any file name and tool-specific options. SOFTUNE Workbench has the macros indicating that any file name and tool-specific options are specified as options.

If any character string other than parameters is specified, it is passed directly to the tool. For details about the parameters, see Section 1.11 Macro Descriptions Usable in Manager.

■ Macro List

The Setup Customize Build dialog provides a macro list for macro input. The build file, load module file, project file submenus indicate their sub-parameters specified.

The environment variable brackets must have any item; otherwise, resulting in an error.

Table 1.4-1 Macro List

Macro List	Macro Name
Build file	%(FILE)
Load module file	%(LOADMODULEFILE)
Project file	%(PRJFILE)
Workspace file	%(WSPFILE)
Project directory	%(PRJPATH)
Target file directory	%(ABSPATH)
Object file directory	%(OBJPATH)
List file directory	%(LSTPATH)
Project construction name	%(PRJCONFIG)
Environment variable	%(ENV[])
Temporary file	%(TEMPFILE)

Note:

When checking [Use the Output window], note the following:

- Once a tool is activated, Make/Build activated until the tool is terminated.
- The Output window must not be used with a tool using a wait state for user input while the tool is executing. The user can not perform input while the Output window is in use, so the tool cannot be terminated. To forcibly terminate the tool, select the tool on the Task bar and input Control - C, or Control - Z.

1.5 Include Dependencies Analysis Function

This section describes the function of the Include Dependencies Analysis of SOFTUNE Workbench.

■ Analyzing Include Dependencies

A source file usually includes some include files. When only an include file has been modified leaving a source file unchanged, SOFTUNE Workbench cannot execute the Make function unless it has accurate and updated information about which source file includes which include files.

For this reason, SOFTUNE Workbench has a built-in Include Dependencies Analysis function. This function can be activated by selecting the [Project] -[Include Dependencies] menu. By using this function, uses can know the exact dependencies, even if an include file includes another include file.

SOFTUNE Workbench automatically updates the dependencies of the compiled/assembled files.

Note:

When executing the [Project] - [Include Dependencies] command, the Output window is redrawn and replaced by the dependencies analysis result.

If the contents of the current screen are important (error message, etc.), save the contents to a file and then execute the Include Dependencies command.

1.6 Functions of Setting Tool Options

This section describes the functions to set options for the language tools activated from SOFTUNE Workbench.

■ Function of Setting Tool Options

To create a desired target file, it is necessary to specify options for the language tools such as a compiler, assembler, and linker. SOFTUNE Workbench stores and manages the options specified for each tool in project configurations.

Tool options include the options effective for all source files (common options) and the options effective for specific source files (individual options). For details about the option setting, refer to Section 4.5.5 of SOFTUNE WORKBENCH Operation Manual.

- Common options

  These options are effective for all source files (excluding those for which individual options are specified) stored in the project.

- Individual options

  These options are compile/assemble options effective for specific source files. The common options specified for source files for which individual options are specified become invalid.

■ Tool Options

SOFTUNE Workbench the macros indicating that any file name and directory name are specified as options.

If any character string other than parameters is specified, it is passed directly to the tool. For details about the parameters, see Section 1.11 Macro Descriptions Usable in Manager. For details about the tool options for each tool, see the manual of each tool.

■ Reference Section

Setup Project

Development Environment

1.7 Error Jump Function

This section describes the error jump function in SOFTUNE Workbench.

■ Error Jump Function

When an error, such as a compile error occurs, double-clicking the error message displayed in the Output window, opens the source file where the error occurred, and automatically moves the cursor to the error line. This function permits efficient removal of compile errors, etc.

The SOFTUNE Workbench Error Jump function analyzes the source file names and line number information embedded in the error message displayed in the Output window, opens the matching file, and jumps automatically to the line.

The location where a source file name and line number information are embedded in an error message, varies with the tool outputting the error.

An error message format can be added to an existing one or modified into an new one. However, the modify error message formats for pre-installed Fujitsu language tools are defined as part of the system, these can not be modified.

A new error message format should be added when working the Error Jump function with user register. To set Error Jump, execute the [Setup] - [Error Jump Setting] menu.

■ Syntax

An error message format can be described in Syntax. SOFTUNE Workbench uses macro descriptions as shown in the Table 1.7-1 to define such formats.

To analyze up to where %f, %h, and %* continue, SOFTUNE Workbench uses the character immediately after the above characters as a delimiter. Therefore, in Syntax, the description until a character that is used as a delimiter re-appears, is interpreted as a file name or a keyword for help, or is skipped over. To use % as a delimiter, describe as %%. The %[char] macro skips over as long as the specified character continues in parentheses. To specify "]" as a skipped character, describe it as "\]". Blank characters in succession can be specified with a single blank character.

Table 1.7-1 List of Special Characters String for Analyzing Error Message

Characters	Semantics
%f	Interpret as source file name and inform editor.
%l	Interpret as line number and inform editor.
%h	Become keyword when searching help file.
%*	Skip any desired character.
%[char]	Skip as long as characters in [ ] continues.

[Example]

*** %f(%l) %h: or, %[*] %f(%l) %h:

The first four characters are "*** ", followed by the file name and parenthesized page number, and then the keyword for help continues after one blank character.

This represents the following message:

***C :\Sample\sample.c(100)   E4062C: Syntax Error: near /int.

■ Reference Section

Setup Error Jump

1.8 Editor Functions

This section describes the functions of the SOFTUNE Workbench built-in standard editor.

■ Standard Editor

SOFTUNE Workbench has a built-in editor called the standard editor. The standard editor is activated as the Edit window in SOFTUNE Workbench. As many Edit windows as are required can be opened at one time.

The standard editor has the following functions in addition to regular editing functions.

- **Keyword marking function in C/assembler source file**

  Displays reserved words, such as if and for, in different color

- **Error line marking function**

  The error line can be viewed in a different color, when executing Error Jump.

- **Bookmark setup function**

  A bookmark can be set on any line, and instantaneously jumps to the line. Once a bookmark is set, the line is displayed in a different color.

- **Ruler, line number display function**

  The Ruler is a measure to find the position on a line; it is displayed at the top of the Edit window. A line number is displayed at the left side of the Edit window.

- **Automatic indent function**

  When a line is inserted using the Enter key, the same indent (indentation) as the preceding line is set automatically at the inserted line. If the space or tab key is used on the preceding line, the same use is set at the inserted line as well.

- **Function to display, Blank, Line Feed code, and Tab code**

  When a file includes a Blank, Line Feed code, and Tab code, these codes are displayed with special symbols.

- **Undo function**

  This function cancels the preceding editing action to restore the previous state. When more than one character or line is edited, the whole portion is restored.

- **Tab size setup function**

  Tab stops can be specified by defining how many digits to skip when Tab codes are inserted. The default is 8.

- **Font changing function**

  The font size for character string displayed in the Edit window can be selected.

■ Reference Section

Edit Window (The Standard Editor)

1.9 Storing External Editors

This section describes the function to set an external editor to SOFTUNE Workbench.

■ External Editor

SOFTUNE Workbench has a built-in standard editor, and use of this standard editor is recommended. However, another accustomed editor can be used, with setting it, instead of an edit window. There is no particular limit on which editor can be set, but some precautions (below) may be necessary. Use the [Setup] - [Editor setting] menu to set an external editor.

■ Precautions

- **Error jump function**

  The Error Jump cannot move the cursor to an error line if the external editor does not have a function to specify the cursor location when activated the external editor.

- **File save at compiling/assembling**

  SOFTUNE Workbench cannot control an external editor. Always save the file you are editing before compiling/assembling.

■ Setting Options

When activating an external editor from SOFTUNE Workbench, options must be added immediately after the editor name. The names of file to be opened by the editor and the initial location of the cursor (the line number). can be specified. SOFTUNE Workbench has a set of special parameters for specifying any file name and line number, as shown in the Table 1.9-1. If any other character string are described by these parameters, such characters string are passed as is to the editor.

%f (File name) is determined as follows:

1. If the focus is on the SRC tab of Project window, and if a valid file name is selected, the selected file name becomes the file name.
2. When a valid file name cannot be acquired by the above procedure, the file name with a focus in the built-in editor becomes the file name.

%x (project path) is determined as follows:

1. If a focus is on the SRC tab of project window and a valid file name is selected, the project path is a path to the project in which the file is stored.
2. If no path is obtained, the project path is a path to the active project.

Also file name cannot be given double-quotes in the expansion of %f macros.

Therefore, it is necessary for you to provide double-quotes for %f. Depending on the editor, there are line numbers to which there will be no correct jump if the entire option is not given double-quotes.

Table 1.9-1 List of Special Characters for Analyzing Error Message

Parameter	Semantics
%%	Means specifying % itself
%f	Means specifying file name
%l	Means specifying line number
%x	Means specifying project path

■ Example of Optional Settings

Table 1.9-2 Example of Optional Settings

Editor name	Argument
WZ Editor V4.0	%f /j%l
MIFES V1.0	%f+%l
UltraEdit32	%f/%l/1
TextPad32	%f(%l)
PowerEDITOR	%f -g%l
Codewright32	%f -g%l
Hidemaru for Win3.1/95	/j%l:1 %f
ViVi	/line=%l %f

■ Reference Section

Editor Setup

Note:

Regarding execution of error jump in Hidemaru:
To execute error jump in Hidemaru used as an external editor, use the [Others] - [Operating Environment] - [Exclusive Control] command, and then set "When opening the same file in Hidemaru" and "Opening two identical files is inhibited".

1.10 Storing External Tools

This section describes the function to set an external tool to SOFTUNE Workbench.

■ External Tools

A non-standard tool not attached to SOFTUNE Workbench can be used by setting it as an external tool and by calling it from SOFTUNE Workbench. Use this function to coordinate with a source file version control tool.

If a tool set as an external tool is designed to output the execution result to the standard output and the standard error output through the console application, the result can be specified to output the SOFTUNE Workbench Output window. In addition, the allow description of additional parameters each time the tool is activated.

To set an external tool, use the [Setup] - [Setting Tool] menu.

To select the title of a set tool, use the [Setup] - [Activating Tool] menu.

■ Setting Options

When activating an external tool from SOFTUNE Workbench, options must be added immediately after the external tool name. Specify the file names, and unique options, etc.

SOFTUNE Workbench has a set of special parameters for specifying any file name and unique tool options.

If any characters string described other than these parameters, such characters string are passed as is to the external tool.

For details about the parameters, see Section 1.11 Macro Descriptions Usable in Manager.

Note:

When checking [Use the Output window], note the following:

1. Once a tool is activated, neither other tools nor the compiler/assembler can be activated until the tool is terminated.
2. The Output window must not be used with a tool using a wait state for user input while the tool is executing. The user cannot perform input while the Output window is in use, so the tool cannot be terminated. To forcibly terminate the tool, select the tool on the Task bar and input Control - C, or Control - Z.

■ Reference Section

Setting Tools

Starting Tools

1.11 Macro Descriptions Usable in Manager

This section explains the macro descriptions that can be used in the manager of SOFTUNE Workbench.

■ Macros

SOFTUNE Workbench has special parameters indicating that any file name and tool-specific options are specified as options.

The use of these parameters as tool options eliminates the need for options specified each time each tool is started.

The type of macro that can be specified and macro expansion slightly vary depending on where to describe macros. The macros usable for each function are detailed below. For the macros that can be specified for ‘‘Error Jump’’ and ‘‘External Editors’’ see Sections 1.7 ‘‘Error Jump Function’’ and 1.9 ‘‘Storing External Editors’’.

■ Macro List

The following is a list of macros that can be specified in SOFTUNE Workbench.

The macros usable for each function are listed below.

- External tools:	Table 1.11-1 and Table 1.11-2
- Customize build:	Table 1.11-1 and Table 1.11-2
- Tool options:	Table 1.11-2

The directory symbol \ is added to the option directories in Table 1.11-1 but not to the macro directories in Table 1.11-2.

The sub-parameters in Table 1.11-3 can be specified in %(FILE), %(LOADMOUDLEFILE), %(PRJFILE).

The sub-parameter is specified in the form of %(PRJFILE[PATH]).

If the current directory is on the same drive, the relative path is used. The current directory is the workspace directory for %(PRJFILE), and %(WSPFILE), and the project directory for other than them.

Table 1.11-1 List of Macros That Can Be Specified 1

Parameter	Meaning
%f	Passed as full-path name of file. (*1)
%F	Passed as main file name of file. (*1)
%d	Passed as directory of file. (*1)
%e	Passed as extension of file. (*1)
%a	Passed as full-path name of load module file.
%A	Passed as main file name of load module file. (*2)
%D	Passed as directory of load module file. (*2)
%E	Passed as extension of load module file. (*2)
%x	Passed as directory of project file. (*2)
%X	Passed as main file name of project file. (*2)
%%	Passed as %.

Table 1.11-2 List of Macros That Can Be Specified 2

Parameter	Meaning
%(FILE)	Passed as full-path name of file. (*1)
%(LOADMODULEFILE)	Passed as full-path name of load module file. (*2)
%(PRJFILE)	Passed as full-path name of project file. (*2)
%(WSPFILE)	Passed as full-path name of workspace file. (*3)
%(PRJPATH)	Passed as directory of project file. (*2)
%(ABSPATH)	Passed as directory of target file. (*2)
%(OBJPATH)	Passed as directory of object file. (*2)
%(LSTPATH)	Passed as directory of list file. (*2)
%(PRJCONFIG)	Passed as project configuration name. (*2) (*3)
%(ENV [Environment variable])	Environment variable specified in environment variable brackets is passed.
%(TEMPFILE)	Temporary file is created and its full-path name is passed. (*4)

The macros in (*1) are determined as follows:

- Customize build

1. Source file before and after executing compiler and assembler
2. Target file before and after executing linker, librarian and converter
3. Configuration file before and after executing configuration

- Tool options

  Null character

- Others
  1. File as focus is on the SRC tab of project window and valid file name is selected
  2. File on which focus is in internal editor as no valid file name can be obtained in 1
  3. Null character if no valid file name can be obtained

The macros in (*2) are determined as follows:

- Customize build and tool options

  Information on configuration of project under building, making, compiling and assembling

- Others
  1. Information on active configuration of project in which file is stored as focus is on the SRC tab of project window and valid file name is selected
  2. Information on active configuration of active project if no valid file name can be obtained in 1

Only project files in the workspace project format can be used for macros indicated by (*3).

Data in the temporary file in (*4) can be specified only for customize build.

Table 1.11-3 List of Sub parameters 1

Sub parameter	Meaning
[PATH]	Directory of file
[RELPATH]	Relative Path of file
[NAME]	Main file name of file
[EXT]	Extension of file
[SHORTFULLNAME]	Full path name of short file
[SHORTPATH]	Directory of short file
[SHORTNAME]	Main file name of short file
[FOLDER]	Name of folder in which files are stored in the SRC tab of project window (Can be specified only in %(FILE).)(*)

The macro in (*) can be used only the project of workspace project format.

■ Examples of Macro Expansion

If the following workspace is opened, macro expansion is performed as follows:

Workspace : C:/Wsp/Wsp.wsp

Active project : C:/Wsp/Sample/Sample.prj

Active project configuration - Debug

Object directory : C:/Wsp/Sample/Debug/Obj/

Subproject : C:/Subprj/Subprj.prj

Active project configuration - Release

Object directory : C:/Subprj/Release/Obj/

Target file	:	C:/Subprj/Release/Abs/Subprj.abs

[Example] Macro expansion in external tools

Focus is on Subprj project file in the SRC tab of project window.

%a	:	C:/Subprj/Release/Abs/Subprj.abs
%A	:	SUBPRJ.abs
%D	:	C:/Subprj/Release/Abs/
%E	:	.abs
%(FILE[FOLDER])	:	Source Files/Common
%(PRJFILE)	:	C:Subprj/Subprj.prj

Focus is not in the SRC tab of project window.

%a	:	C:/Wsp/Sample/Debug/Abs/Sample.abs
%A	:	Sample.abs
%D	:	C:/Wsp/Sample/Debug/Abs/
%(PRJFILE)	:	C:/Wsp/Sample/Sample.prj

[Example] Macro expansion in customize build

Release configuration of Subprj project is built.

%(FILE)	:	C:/Subprj/LongNameFile.c
%(FILE[PATH])	:	C:/Subprj
%(FILE[RELPATH])	:	.
%(FILE[NAME])	:	LongNameFile
%(FILE[EXT])	:	.c
%(FILE[SHORTFULLNAME])	:	C:/Subprj/LongFi~1.c
%(FILE[SHORTPATH])	:	C:/Subprj
%(FILE[SHORTNAME])	:	LongFi~1
%(PRJFILE[RELPATH])	:	../Subprj
%(PRJPATH)	:	C:/Subprj
%(OBJPATH)	:	C:/Subprj/Release/Obj
%(PRJCONFIG)	:	Release
%(ENV[FETOOL])	:	C:/SOFTUNE
%(TEMPFILE)	:	C:/Subprj/Release/Opt/_fs1056.TMP

[Example] Macro expansion in tool options

Release configuration of Subprj project is built.

%(FILE)	:	
%(PRJFILE[RELPATH])	:	../Subprj
%(PRJPATH)	:	C:/Subprj
%(OBJPATH)	:	C:/Subprj/Release/Obj
%(PRJCONFIG)	:	Release
%(ENV[FETOOL])	:	C:/SOFTUNE

1.12 Setting Operating Environment

This section describes the functions for setting the SOFTUNE Workbench operating environment.

■ Operating Environment

Set the environment variables for SOFTUNE Workbench and some basic setting for the Project.

To set the operating environment, use the **[Setup]-[Development Environment Setting]** menu.

- **Environment Variables**

  Environment variables are variables that are referred to mainly using the language tools activated from SOFTUNE Workbench. The semantics of an environment variable are displayed in the lower part of the Setup dialog. However, the semantics are not displayed for environment variables used by tools added later to SOFTUNE Workbench.

  When SOFTUNE Workbench and the language tools are installed in a same directory, it is not especially necessary to change the environment variable setups.

- **Basic setups for Project**

  The following setups are possible.

- **Open the previously worked-on Project at start up**

  When starting SOFTUNE Workbench, it automatically opens the last worked-on Project.

- **Display options while compiling/assembling**

  Compile options or assemble options can be viewed in the Output window.

- **Save dialog before closing Project**

  Before closing the Project, a dialog asking for confirmation of whether or not to save the Project to the file is displayed. If this setting is not made, SOFTUNE Workbench automatically saves the Project without any confirmation message.

- **Save dialog before compiling/assembling**

  Before compiling/assembling, a dialog asking for confirmation of whether or not to save a source file that has not been saved is displayed. If this setting is not made, the file is saved automatically before compile/assemble/make/build.

- **Termination message is highlighted at Make/Build**

  At Compile, Assemble, Make, or Build, the display color of termination messages (Abort, No Error, Warning, Error, Fatal error, or Failing During start) can be changed freely by the user.

■ Reference Section

Development Environment

Note:

Because the environment variables set here are language tools for the SOFTUNE Workbench, the environment variables set on previous versions of SOFTUNE cannot be used. In particular, add the set values of [User Include Directory] and [Library Search Directory] to [Tool Options Settings].

1.13 Debugger Types

This section describes the types of SOFTUNE Workbench debuggers.

■ Type of Debugger

SOFTUNE Workbench integrates three types of debugger: a simulator debugger, emulator debugger, and monitor debugger. Any one can be selected depending on the requirement.

■ Simulator Debugger

The simulator debugger simulates the MCU operations (executing instructions, memory space, I/O ports, interrupts, reset, etc.) with software to evaluate a program.

It is used for evaluating an uncompleted system and operation of individual units, etc.

■ Emulator Debugger

The emulator debugger is software to evaluate a program by controlling an emulator from a host through a communications line (RS-232C, LAN, USB).

Before using this debugger, the emulator must be initialized.

■ Monitor Debugger

The monitor debugger evaluates a program by putting it into an evaluation system and by communicating with a host. An RS-232C interface and an area for the debug program are required within the evaluation system.

For further information on the MCU-related items, see Chapter 2 and later in this manual.

1.14 Memory Operation Functions

This section describes the memory operation functions.

■ Functions for Memory Operations

- **Display/Modify memory data**

  Memory data can be display in the Memory window and modified.

- **Fill**

  The specified memory area can be filled with the specified data.

- **Copy**

  The data in the specified memory area can be copied to another area.

- **Compare**

  The data in the specified source area can be compared with data in the destination area.

- **Search**

  Data in the specified memory area can be searched.

  For further details of the above functions, refer to "3.11 Memory Window" in "SOFTUNE WORKBENCH Operation Manual".

- **Display/Modify C variables**

  The names of variables in a C source file can be displayed in the Watch window and modified.

- **Setting Watch point**

  By setting a watch point at a specific address, its data can be displayed in the Watch window.

  For further details of the above functions, refer to "3.13 Watch Window" in "SOFTUNE WORKBENCH Operation Manual".

1.15 Register Operations

This section describes the register operations.

■ Register Operations

The Register window is opened when the [View] - [Register] command is executed. The register and flag values can be displayed in the Register window.

For further details about modifying the register value and the flag value, refer to "4.4.4 Register" in "SOFTUNE WORKBENCH Operation Manual".

The name of the register and flag displayed in the Register window varies depending on each MCU in use. For the list of register names and flag names for the MCU in use, refer to the Operational Manual Appendix.

■ Reference Section

Register Window

1.16 Line Assembly and Disassembly

This section describes line assembly and disassembly.

■ Line Assembly

To perform line-by-line assembly (line assembly), right-click anywhere in the Disassembly window to display the short-cut menu, and select [Line Assembly]. For further details about assembly operation, refer to "4.4.3 Assembly" in "SOFTUNE WORKBENCH Operation Manual".

■ Disassembly

To display disassembly, use the [View]-[Disassembly] command. By default, disassembly can be viewed starting from the address pointed by the current program counter (PC). However, the address can be changed to any desired address at start-up.

Disassembly for an address outside the memory map range cannot be displayed. If this is attempted, "???" is displayed as the mnemonic.

■ Reference Section

Disassembly Window

1.17 Symbolic Debugging

The symbols defined in a source program can be used for command parameters (address). There are three types of symbols as follows:
- Global Symbol
- Static Symbol within Module (Local Symbol within Module)
- Local Symbol within Function

■ Types of Symbols

A symbol means the symbol defined while a program is created, and it usually has a type. Symbols become usable by loading the debug information file.

Furthermore, a type of the symbol in C is recognized and the command is executed.

There are three types of symbols as follows:

- **Global symbol**

  A global symbol can be referred to from anywhere within a program. In C, variables and functions defined outside a function without a static declaration are in this category. In assembler, symbols with a PUBLIC declaration are in this category.

- **Static symbol within module (Local symbol within module)**

  A static symbol within module can be referred to only within the module where the symbol is defined.

  In C, variables and functions defined outside a function with a static declaration are in this category. In assembler, symbols without a PUBLIC declaration are in this category.

- **Local symbol within function**

  A local symbol within a function exists only in C. A static symbol within a function and an automatic variable are in this category.

- **Static symbol within function**

  Out of the variables defined in function, those with static declaration.

- **Automatic variable**

  Out of the variables defined in function, those without static declaration and parameters for the function.

■ Setting Symbol Information

Symbol information in the file is set with the symbol information table by loading a debug information file. This symbol information is created for each module.

The module is constructed for each source file to be compiled in C, in assembler for each source file to be assembled.

The debugger automatically selects the symbol information for the module to which the PC belongs to at abortion of execution (Called "the current module"). A program in C also has information about which function the PC belongs to.

■ Line Number Information

Line number information is set with the line number information table in SOFTUNE Workbench when a debug information file is loaded. Once registered, such information can be used at anytime thereafter. Line number is defined as follows:

```
[Source File Name]    $Line Number
```

1.17.1 Referring to Local Symbols

This section describes referring to local symbols and Scope.

■ Scope

When a local symbol is referred to, Scope is used to indicate the module and function to which the local symbol to be referred belongs.

SOFTUNE Workbench automatically scopes the current module and function to refer to local symbols in the current module with preference. This is called the Auto-scope function, and the module and function currently being scoped are called the Current Scope.

When specifying a local variable outside the Current Scope, the variable name should be specified by the module and function to which the variable belongs. This method of specifying a variable is called a symbol path name or a Search Scope.

■ Moving Scope

As explained earlier, there are two ways to specify the reference to a variable: by adding a Search Scope when specifying the variable name, and by moving the Current Scope to the function with the symbol to be referred to. The Current Scope can be changed by displaying the Call Stack dialog and selecting the parent function. For further details of this operation, refer to "4.6.7 Stack" in "SOFTUNE WORKBENCH Operation Manual". Changing the Current Scope as described above does not affect the value of the PC.

By moving the current scope in this way, you can search a local symbol in parent function with precedence.

■ Specifying Symbol and Search Procedure

A symbol is specified as follows:

```
[[Module Name] [\Function Name] \] Symbol Name
```

When a symbol is specified using the module and function names, the symbol is searched. However, when only the symbol name is specified, the search is made as follows:

1. Local symbols in function in Current Scope
2. Static symbols in module in Current Scope
3. Global symbols

If a global symbol has the same name as a local symbol in the Current Scope, specify "\" or "::" at the start of global symbol. By doing so, you can explicitly show that is a global symbol.

An automatic variable can be referred to only when the variable is in memory. Otherwise, specifying an automatic variable causes an error.

1.17.2 Referring to C Variables

C variables can be specified using the same descriptions as in the source program written in C.

■ Specifying C Variables

C variables can be specified using the same descriptions as in the source program. The address of C variables should be preceded by the ampersand symbol "&". Some examples are shown in the Table 1-17-1.

Table 1.17-1 Examples of Specifying Variables

Example of Variables		Example of Specifying Variables	Semantics
Regular Variable	int data;	data	Value of data
Pointer	char *p;	*p	Value pointed to by p
Array	char a[5];	a[1]	Value of second element of a

Structure	struct stag {
	char c;
	int i;
	};
	struct stag st;
	struct stag *stp;
		st.c
		stp- >c
			Value of member c of st
			Value of member c of the structure to which stp points

Union	union utag {
	char c;
	int i;
	} uni;
		uni.i	Value of member i of uni

Address of variable	int data;	&data	Address of data

Reference type	int i;
	int &ri = i;
		ri	Same as i

■ Notes on C Symbols

The C compiler outputs symbol information with "_" prefixed to global symbols. For example, the symbol main outputs symbol information _main. However, SOFTUNE Workbench permits access using the symbol name described in the source to make the debug of program described by C language easier.

Consequently, a symbol name described in C and a symbol name described in assembler, which should both be unique, may be identical.

In such a case, the symbol name in the Current Scope normally is preferred. To refer to a symbol name outside the Current Scope, specify the symbol with the module name.

If there are duplicated symbols outside the Current Scope, the symbol name searched first becomes valid. To refer to another one, specify the symbol with the module name.

CHAPTER 2
Dependence Functions

This chapter describes the functions dependent on each Debugger.

2.1 Simulator Debugger

2.2 Emulator Debugger (MB2141)

2.3 Emulator Debugger (MB2147-01)

2.4 Emulator Debugger (MB2147-05)

2.5 Monitor Debugger

2.6 Abortion of Program Execution (SIM, EML, MON)

2.1 Simulator Debugger

This section describes the functions of the simulator debugger for the F^{2}MC-16 Family

■ Simulator Debugger

The simulator debugger simulates the MCU operations (executing instructions, memory space, I/O ports, interrupts, reset, etc.) with software to evaluate a program.

It is used to evaluate an uncompleted system, the operation of single units, etc.

There are 2 types of simulator debuggers.

- Normal simulator debugger (normal)
- High-speed simulator debugger (fast)

This high-speed simulator debugger provides substantial reductions in simulation time due to a dramatic review of normal simulator debugger's processing methods.

The high-speed simulator debugger can be instruction processing performance for 10MIPS when it is operated by PC equipped with Pentium4 2.0GHz.

External I/F for simulator are equipped to high-speed simulator debugger to create peripheral simulation modules.

Please refer to "Appendix I External I/F DLL for Simulator" in "SOFTUNE Workbench Operation Manual".

■ Operating Condition of High-speed Simulator Debugger

The high-speed simulator debugger requires much more RAM space on the host PC than that of normal simulator debugger.

The required RAM size depends largely on your program size.

For the required available RAM space, see the table below:

Basic use	Fs907s.exe	20MB
CODE size of target program	per 64 KB	6MB
DATA size of target program	per 64 KB	1.5MB

Insufficient RAM space will lead to an extreme decrease in simulation speed.

Target program size

CODE	XX(KB)
DATA	YY(KB)

Required RAM space (MB) = 20 + (XX / 64) * 6 + (YY / 64) * 1.5

However, RAM space larger than the above may be needed depending on program allocation. Consecutive areas should be reserved as much as possible.

Example: Program with 1 MB of CODE and DATA sizes

Required RAM space (MB) = 20 + (1024 / 64) * 6 + (1024 / 64) * 1.5 = 140MB

■ Simulation Range

The simulator debugger simulates the MCU operations (instruction operations, memory space, I/O ports, interrupts, reset, power-save consumption mode, etc.) Peripheral I/Os, such as a timer, DMAC and serial I/O, other than the CPU core of the actual chip are not supported as peripheral resources. I/O space to which peripheral I/Os are connected is treated as memory space. There is a method for simulating interrupts like timer interrupts, and data input to memory like I/O ports. For details, see the sections concerning I/O port simulation and interrupt simulation.

- Instruction simulation
- Memory simulation
- I/O port simulation (Input port)
- I/O port simulation (Output port)
- Interrupt simulation
- Reset simulation
- Power-save consumption mode simulation

2.1.1 Instruction Simulation

This section describes the instruction simulation executed by SOFTUNE Workbench.

■ Instruction Simulation

This simulates the operations of all instructions supported by the F^{2}MC-16/16L/16LX/16H/16F. It also simulates the changes in memory and register values due to such instructions.

2.1.2 Memory Simulation

This section describes the memory simulation executed by SOFTUNE Workbench.

■ Memory Simulation

The simulator debugger must first secure memory space to simulate instructions because it simulates the memory space secured in the host machine memory.

The following operation is required.

- To secure the memory area, either use the [Setup] - [Memory Map] command, or the SET MAP command in the Command window.
- Load the file output by the Linkage Editor (Load Module File) using either the [Debug] - [Load target file] command, or the LOAD/OBJECT command in the Command window.

■ Simulation Memory Space

Memory space access attributes can be specified byte-by-byte using the [Setup] - [Memory Map] command. The access attribute of unspecified memory space is Undefined.

■ Memory Area Access Attributes

Access attributes for memory area can be specified as shown in Table 2.1-1. A guarded access break occurs if access is attempted against such access attribute while executing a program. When access is made by a program command, such access is allowed regardless of the attribute, CODE, READ or WRITE. However, access to memory in an undefined area causes an error.

Table 2.1-1 Types of Access Attributes

Attribute	Semantics
CODE	Instruction operation enabled
READ	Data read enabled
WRITE	Data write enabled
undefined	Attribute undefined (access prohibited)

2.1.3 I/O Port Simulation

The output to I/O ports can be recorded in the specified buffer or file.This section describes I/O port simulation executed by SOFTUNE Workbench.

■ I/O Port Simulation (Input Port)

There are two types of simulations in I/O port simulation: input port simulation, and output port simulation. Input port simulation has the following types:

- Whenever a program reads the specified port, data is input from the pre-defined data input source.
- Whenever the instruction execution cycle count exceeds the specified cycle count, data is input to the port.

To set an input port, use the [Setup] - [Debug Environment] - [I/O Port] command, or the SET IMPORT command in the Command window.

Up to 4096 port addresses can be specified for the input port. The data input source can be a file or a terminal. After reading the last data from the file, the data is read again from the beginning of the file. If a terminal is specified, the input terminal is displayed at read access to the set port.

A text file created by an ordinary text editor, or a binary file containing direct code can be used as the data input file. When using a text file, input the input data inside commas (,). When using a binary file, select the binary radio button in the input port dialog.

■ I/O Port Simulation (Output Port)

At output port simulation, whenever a program writes data to the specified port, writing is executed to the data output destination.

To set an output port, either use the [Setup] - [Debug Environment] - [I/O Port] command, or the SET OUTPORT command in the Command window.

Up to 4096 port addresses can be set as output ports. Select either a file or terminal (Output Terminal window) as the data output destination.

A destination file must be either a text file that can be referred to by regular editors, or a binary file. To output a binary file, select the Binary radio button in the Output Port dialog.

Note:

The following method is not supported by high-speed simulator debugger.

- Whenever the instruction execution cycle count exceeds the specified cycle count, data is input to the port.

Furthermore the setting of memory map is necessary to set I/O port. When deleting memory map, I/O port is also deleted.

2.1.4 Interrupt Simulation

This section describes the interrupt simulation executed by SOFTUNE Workbench.

■ Interrupt Simulation

Simulate the operation of the MCU (including intelligent I/O service) in response to an interrupt request. Note that intelligent I/O service does not support any end request from the resource.

Provisions for the causes of interrupts and interrupt control registers are made by referencing data in the install file read at simulator start up.

*Intelligent I/O service provides automatic data transfer between I/O and memory. This function allows exchange of data between memory and I/O, which was done previously by the interrupt handling program, using DMA (Direct Memory Access). (For details, refer to the user manual for each model.)

The methods of generating interrupts are as follows:

- Execute instructions for the specified number of cycles while the program is running (during execution of executable commands) to generate interrupts corresponding to the specified interrupt numbers and cancel the interrupt generating conditions.
- Continue to generate interrupts each time the number of instruction execution cycles exceeds the specified number of cycles.

The method of generating interrupts is set by the [Setup]-[Debug environment]-[Interrupt] command. If interrupts are masked by the interrupt enable flag when the interrupt generating conditions are established, the interrupts are generated after they are unmasked.

MCU operation in response to an interrupt request is also supported for the following exception handling:

- Execution of undefined instructions
- Address error in program access

  (Program access to internal RAM area and internal I/O area)
- Stack area error (only for 16F)

Note:

When an external interrupt is generated while under an interrupt mask at high-speed simulator debugger, that interrupt factor is eliminated.

2.1.5 Reset Simulation

This section describes the reset simulation executed by SOFTUNE Workbench.

■ Reset Simulation

The simulator debugger simulates the operation when a reset signal is input to the MCU using the [Debug]-[Reset MCU] command and initializes the registers. The function for performing reset processing by operation of MCU instructions (writing to RST bit in standby control register) is also supported. In this case, the reset message (Reset) is displayed on the status bar.

2.1.6 Power-Save Consumption Mode Simulation

This section describes the low power-save consumption SOFTUNE Workbench mode simulation executed by SOFTUNE Workbench.

■ Power-Save Consumption Mode Simulation

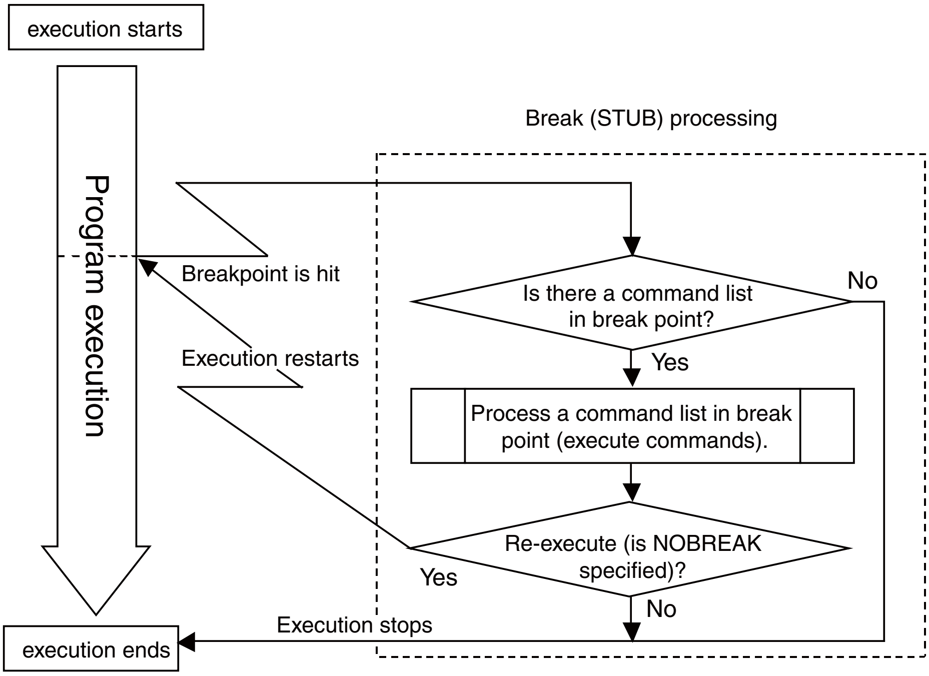
The MCU enters the power-save consumption mode in accordance with the MCU instruction operation (Write to SLEEP bit or STOP bit of standby control register). Once in the sleep mode or stop mode, a message ("sleep" for sleep mode, "stop" for stop mode) is displayed on the Status Bar. The loop keeps running until either an interrupt request is generated, or the [Debug] - [Abort] command is executed. Each cycle of the loop increments the count by 1. During this period, I/O port processing can be operated. Writing to the standby control register using a command is not prohibited.

2.1.7 STUB Function

This section describes the STUB function which executes commands automatically when the breakpoint hit occurs.

■ STUB Function

The STUB function is supported so that a series of commands in the command list can automatically be executed when a specified breakpoint is hit. The use of this function enables spot processing, such as simple I/O simulation, external interrupt generation, and memory reprogramming, without changing the main program. This function is effective only when the simulator debugger is used.


execution starts
Program execution
Break (STUB) processing
Breakpoint is hit
Execution restarts
Is there a command list in break point?
No
Yes
Process a command list in break point (execute commands).
Re-execute (is NOBREAK specified)?
Yes
No
Execution stops
execution ends

■ Setting

The STUB function can be set by the following commands.

- Dialog
  1. Breakpoint Set Dialog - [Code] tab
  2. Breakpoint Set Dialog - [Data] tab
- Command
  1. SET BREAK
  2. SET DATABREAK

2.2 Emulator Debugger (MB2141)

This section explains the functions of the emulator debuggers for the MB2141.

■ Emulator Debugger

When choosing the emulator debugger from the setup wizard, select one of the following emulators. Select the MB2141.

MB2141

MB2147-01

MB2147-05

The emulator debugger for the MB2141 is software that controls an emulator from a host computer via a communications line (RS-232C or LAN) to evaluate programs.

The following series can be debugged:

When MB2141-506 pod used

F^2MC-16/16H

F^2MC-16F

F^2MC-16L

F^2MC-16LX

When MB2141-507 pod used

F^2MC-16F

F^2MC-16L

F^2MC-16LX

Before using the emulator, the emulator must be initialized. For father details, refer to "Appendix B. Downloading Monitor Program", and "Appendix C. Setting LAN Interface" in "SOFTUNE WORKBENCH Operation Manual".

For further details, refer to "Appendix B Download Monitor Program", and "Appendix C Setting up LAN Interface" in "SOFTUNE WORKBENCH Operation Manual".

2.2.1 Setting Operating Environment

This section explains the operating environment setup.

■ Setting Operating Environment

For the emulator debugger for the MB2141, it is necessary to set the following operating environment. Predefined default settings for all these setup items are enabled at startup. Therefore, setup is not required when using the default settings. Adjusted settings can be used as new default settings from the next time.

- MCU operation mode
- Debug area
- Memory mapping
- Timer minimum measurement unit

2.2.1.1 MCU Operation Mode

There are two MCU operation modes as follows:

- **Debugging Mode**
- **Native Mode**

■ Setting MCU Operation Mode

Set the MCU operation mode.

There are two operation modes: the debugging mode, and the native mode. Choose either one using the SET RUNMODE command.

At emulator start-up, the MCU is in the debugging mode.

When the MCU operation mode is changed, all the following are initialized:

- Data break points
- Event condition settings
- Sequencer settings
- Trace measurement settings and trace buffer
- Performance measurement settings and measured result

■ Debugging Mode

All the operations of evaluation chips can be analyzed, but their operating speed is slower than that of mass-produced chips.

■ Native Mode

Evaluation chips have the same timing as mass-produced chips to control the operating speed. Note that the restrictions the shown in Table 2.2-1 are imposed on the debug functions.

Table 2.2-1 Restrictions on debug functions in native mode

Applicable series	Restrictions on debug functions
F^2MC-16/16H	- Memory mapping setting is disabled and each area is accessed to the MCU specifications. - Traces cannot be disassembled.
Common to all series	- When a data read access occurs on the MCU internal bus, the internal bus access information is not sampled and stored in the trace buffer. - Even when a data break or event (data access condition) is set for data on the MCU internal bus, it may not become a break factor or sequencer-triggering factor. - The coverage function may fail to detect an access to data on the MCU internal bus.

■ MCU Operation Speed

To support a broader range of MCU operation speeds, the emulator adjusts control of the MCU according to the MCU operation speed.

Normally, set the low-speed operation mode. If the F^{2}MC-16H/16F series is operated at high speed and malfunctions occur, change the setting to the high-speed operation mode.

Also, to start at low speed and then change to high speed because of the gear setting, etc., use the SET RUNMODE command to change the setting.

2.2.1.2 Debug Area

Set the intensive debugging area out of the whole memory space. The area functions are enhanced.

■ Setting Debug Area

There are two debug areas: DEBUG1, and DEBUG2. A continuous 512KB area (8 banks) is set for each area.

Set the debug area using the SET DEBUG command.

Setting the debug area enhances the break points/data break points and the coverage measurement function.

- **Enhancement of Break Points**

  Up to six break points (not including temporary break points set using GO command) can be set when the debug area has not been set yet.

  When setting the debug area as the CODE attribute, up to 65535 break points can be set if they are within the area. At this time, up to six break points can be set for an area other than the debug area, but the total count of break points must not exceed 65535.

- **Enhancement of Data Break Points**

  Up to six data break points can be set when the debug area has not been set yet.

  When setting the debug area of the data attribute (READ, WRITE), up to 65535 data break points can be set if they are within the area and have the same attribute. At this time, up to six data break points can be set for an area other than the area or for a different attribute, but the total number of data break points must not exceed 65535.

- **Enhancement of Coverage Measurement Function**

  Setting the debug area enables the coverage measurement function. In coverage measurement, the measurement range can be specified only within the area specified as the debug area.

  The attributes for the debug area are "Don't care" as long as it is being used for coverage measurement. The coverage measurement attribute can be set, regardless of the debug area attributes.

2.2.1.3 Memory Area Types

A unit in which memory is allocated is called an area. There are seven different area types.

■ Memory Area Types

A unit to allocate memory is allocated is called an area. There are seven different area types as follows:

- **User Memory Area**

  Memory space in the user system is called the user memory area and this memory is called the user memory. Up to eight user memory areas can be set with no limit on the size of each area.

  Access attributes can be set for each area; for example, CODE, READ, etc., can be set for ROM area, and READ, WRITE, etc. can be set for RAM area. If the MCU attempts access in violation of these attributes, the MCU operation is suspended and an error is displayed (guarded access break).

  To set the user memory area, use the SET MAP command. The F^{2}MC-16/16H only allows this setup in the debugging mode.

- **Emulation Memory Area**

  Memory space substituted for emulator memory is called the emulation memory area, and this memory is called emulation memory.

  As emulation memory area, Using MB2145-506 emulation pod, up to seven areas (including mirror area and internal ROM area described below) each with a maximum size of 64 KB can be set. An area larger than 64 KB can be set, but the areas are managed internally in 64 KB units.

  Using MB2145-507 emulation pod, up to seven areas (including mirror area and internal ROM area described below) each with a maximum size of 512 KB can be set.

  To set the emulation memory area, use the SET MAP command. Attributes are set as for user memory area.

Note:

Even if the MCU internal resources are set as emulation memory area, access is made to the internal resources. The F^{2}MC-16/16H only allows this setup in the debugging mode.

- **Mirror Area**

  The mirror area is a region in the emulator memory that makes copies of user memory accesses. The memory in this area is called a mirror region.

  The mirror area is used while it overlaps with a user memory area or undefined area. It is implemented by the emulation memory. Up to five mirror areas can be defined including emulation memory areas.

  Mirror areas are used to reference the user memory during on-the-fly execution (For further details, see 2.2.4 On-the-fly Memory Access).

  Mirror areas can be set using the SET MAP command. If the memory contents copy option is selected when a mirror area is set, the contents of the mirror area are always the same contents as the user memory.

Note:

When the F^2MC-16/16H is used, mirror area setup can be performed only in the debugging mode.

- **Internal ROM Area**

  The area where the emulator internal memory is substituted for internal ROM is called the internal ROM area, and this memory is called the internal ROM memory.

  Only one internal ROM area with a size up to 128 KB can be specified. The internal ROM area is capable to set by the "Setup Map" dialog opening by "Debugger Memory Map... " from "Setup".

Note:

The internal memory area, it is set a suitable area automatically by the selected MCU.

- **Internal ROM Image Area (F^2MC-16L, F^2MC-16LX, F^2MC-16F only)**

  Some types of MCUs have data in a specific area of internal ROM appearing to 00 bank. This specific area is called the internal ROM image area.

  The internal ROM image area is capable to set by the "Debugger Setup Map" dialog opening by "Memory Map... " from "Setup". This area attribute is automatically set to READ/CODE. The same data as in the internal ROM area appears in the internal ROM image area.

  Note that the debug information is only enabled for either one (one specified when linked). To debug only the internal ROM image area, change the creation type of the load module file.

Note:

The internal memory area, it is set a suitable area automatically by the selected MCU.

- **Internal Instruction RAM Area (F^2MC-16H only)**

  Some types of MCUs have the internal instruction RAM, and this area is called the internal instruction RAM area.

  The internal instruction RAM area, it is capable to set by the "Internal Instruction RAM area" tab in the "Setup CPU Information" dialog (select menu "project"-"setup project..." , select the "MCU" tab, and push the "Set CPU Information..." button). The size must be specified to either H'100, H'200, H'400, H'800, H'1000, H'2000 or H'4000 bytes.

Note:

The internal memory area, it is set a suitable area automatically by the selected MCU.

- **Undefined Area**

  A memory area that does not belong to any of the areas described above is part of the user memory area. This area is specifically called the undefined area.

The undefined area can be set to either NOGUARD area, which can be accessed freely, or GUARD area, which cannot be accessed. Select either setup for the whole undefined area. If the area attribute is set to GUARD, a guarded access error occurs if access to this area is attempted.

Note:

The F^2MC-16/16H only allows this setup in the debugging mode.

2.2.1.4 Memory Mapping

Memory space can be allocated to the user memory, the emulation memory, etc., and the attributes of these areas can be specified.
However, the MCU internal resources are not dependent on this mapping setup and access is always made to the internal resources.

■ Access Attributes for Memory Areas

The access attributes shown in Table 2.2-2 can be specified for memory areas.

A guarded memory access break occurs if access is attempted in violation of these attributes while executing a program.

When access to the user memory area and the emulation memory area is made using program commands, such access is allowed regardless of the CODE, READ, WRITE attributes. However, access to memory with the GUARD attribute in the undefined area, causes an error.

Table 2.2-2 Types of Access Attributes

Area	Attribute	Description
User Memory Emulation Memory	CODE	Instruction Execution Enabled
	READ	Data Read Enabled
	WRITE	Data Write Enabled
Undefined	GUARD	Access Disabled
	NOGUARD	No check of access attribute

When access is made to an area without the WRITE attribute by executing a program, a guarded access break occurs after the data has been rewritten if the access target is the user memory. However, if the access target is the emulation memory, the break occurs before rewriting. In other words, write-protection (memory data cannot be overwritten by writing) can be set for the emulation memory area by not specifying the WRITE attribute for the area.

This write-protection is only enabled for access made by executing a program, and is not applicable to access by commands.

■ Creating and Viewing Memory Map

Use the following commands for memory mapping.

SET MAP:	Set memory map.
SHOW MAP:	Display memory map.
CANCEL MAP:	Change memory map setting to undefined.

[Example]

```
 >SHOW MAP
address                 attribute           type
000000 .. FFFFFF        noguard
The rest of setting area numbers
user = 8     emulation = 5
 >SET MAP/USER H'0..H'1FF
 >SET MAP/READ/CODE/EMULATION H'FF0000..H'FFFFFF
 >SET MAP/USER H'8000..H'8FFF
 >SET MAP/MIRROR/COPY H'8000..H'8FFF
 >SET MAP/GUARD
 >SHOW MAP
address                 attribute           type
000000 .. 0001FF        read write          user
000200 .. 007FFF        guard
008000 .. 008FFF        read write          user
009000 .. FEFFFF        guard
FF0000 .. FFFFFF        read write code     emulation
mirror address area
008000 .. 008FFF        copy
The rest of setting area numbers
user = 6     emulation = 3
 >
```

■ Internal ROM Area Setting

The [Map Setting] dialog box is displayed using [Environment] - [Debugger Memory Map] command. You can set the internal ROM area using the [Internal ROM Area] tab after the [Map Adding] dialog box is displayed by clicking on the [Setting] button. Two areas can be set. Both ones require empty Emulation area to be set. The region size by (Empty space of the emulation area) x (one area size) can be set.

Specify the internal ROM area from the ending address H'FFFFFF (fixed) for area 1. Also, it is possible to delete the internal ROM area.

2.2.1.5 Timer Minimum Measurement Unit

The timer minimum measurement unit affects the sequencer, the emulation timer and the performance measurement timer.

■ Setting Timer Minimum Measurement Unit

Choose either 1 µs or 100 ns as the timer minimum measurement unit for the emulator for measuring time.

The minimum measurement unit for the following timers is changed depending on this setup.

-Timer values of sequencer (timer conditions at each level)

-Emulation timer

-Performance measurement timer

Table 2.2-3 shows the maximum measurement time length of each timer when 1 µs or 100 ns is selected as the minimum measurement unit.

When the minimum measurement unit is changed, the measurement values of each timer are cleared as well. The default setting is 1 µs.

Table 2.2-3 Maximum Measurement Time Length of Each Timer

	1 µs selected	100 ns selected
Sequencer timer	About 16 seconds	About 1.6 seconds
Emulation timer	About 70 minutes	About 7 minutes
Performance measurement timer	About 70 minutes	About 7 minutes

Use the following commands to control timers.

SET TIMERSCALE command: Sets minimum measurement unit for timers

SHOW TIMERSCALE command: Displays status of minimum measurement unit setting for timers

[Example]

```
>SET TIMERSCALE/100N
>SHOW TIMERSCALE
Timer scale : 100ns
>
```

2.2.2 Notes on Commands for Executing Program

When using commands to execute a program, there are several points to note.

■ Notes on GO Command

For the GO command, two break points that are valid only while executing commands can be set. However, it is required to be careful in setting these break points.

- **Invalid Break Points**

  No break occurs when a break point is set at the instruction immediately after the following instructions.

F^2MC-16L/16LX/16/16H	PCB NCC SPB MOV ILM,#imm8 OR CCR,#imm8	DTB ADB CNR AND CCR,#imm8 POPW PS
F^2MC-16F	PCB NCC SPB	DTB ADB CNR

- No break occurs when break point set at address other than starting address of instruction.
- No break occurs when both following conditions met at one time.
- Instruction for which break point set starts from odd-address,
- Preceding instruction longer than 2 bytes length, and break point already set at last 1-byte address of preceding instruction (This "already-set" break point is an invalid break point that won't break, because it has been set at an address other than the starting address of an instruction).

- **Abnormal Break Point**

  Setting a break point at the instruction immediately after string instructions listed below, may cause a break in the middle of the string instruction without executing the instruction to the end.

F^2MC-16L/16LX/16/16H	MOVS SECQ WBTS MOVSWI SECQWI MOVSD SECQD FILS FILSW	MOVSW SECQW MOVSI SECQI WBTC MOVSWD SECQWD FILSI FILSWI
F^2MC-16F	MOVS SECQ WBTS MOVSWI SECQWI MOVSD SECQD FILS FILSW MOVM	MOVSW SECQW MOVSI SECQI WBTC MOVSWD SECQWD FILSI FILSWI MOVMW

■ Notes on STEP Command

- **Exceptional Step Execution**

When executing the instructions listed in the notes on the GO command as invalid break points and abnormal break points, such instructions and the next instruction are executed as a single instruction. Furthermore, if such instructions are continuous, then all these continuous instructions and the next instruction are executed as a single instruction.

- **Step Execution that won't Break**

Note that no break occurs after step operation when both the following conditions are met at one time.

- When step instruction longer than 2 bytes and last code ends at even address
- When break point already set at last address (This "already-set" break point is an invalid break point that won't break, because it has been set at an address other than the starting address of an instruction.)

■ Controlling Watchdog Timer

It is possible to select "No reset generated by watchdog timer counter overflow" while executing a program using the GO, STEP, CALL commands.

Use the ENABLE WATCHDOG, DISABLE WATCHDOG commands to control the watchdog timer.

- ENABLE WATCHDOG : Reset generated by watchdog timer counter overflow
- DISABLE WATCHDOG : No reset generated by watchdog timer counter overflow

The start-up default in this program is "Reset generated by watchdog timer counter overflow".

[Example]

```
>DISABLE WATCHDOG
>GO
```

2.2.3 On-the-fly Executable Commands

Certain commands can be executed even while executing a program. This is called "on-the-fly" execution.

■ On-the-fly Executable Commands

Certain commands can be executed on-the-fly. If an attempt is made to execute a command that cannot be executed on-the-fly, an error "MCU is busy" occurs. Table 2.2-4 lists major on-the-fly executable functions. For further details, refer to "SOFTUNE WORKBENCH Command Reference Manual".

Meanwhile, on-the-fly execution is enabled only when executing the MCU from the menu or the tool button. On-the-fly commands cannot be executed when executing the GO command, etc., from the Command window.

Table 2.2-4 Major Functions Executable in On-the-fly Mode

Function	Restrictions	Major Commands
MCU reset	-	RESET
Displaying MCU execution status	-	SHOW STATUS
Displaying execution time measurement value (Timer)	-	SHOW TIMER
Memory operation (Read/Write)	Emulation memory only operable Read only enabled in mirror area	ENTER EXAMINE COMPARE FILL MOVE DUMP SEARCH MEMORY SHOW MEMORY SET MEMORY
Line assembly, Disassembly	Emulation memory only enabled Mirror area, Disassembly only enabled	ASSEMBLE DISASSEMBLE
Load, Save program	Emulation memory only enabled Mirror area, save only enabled	LOAD SAVE
Displaying coverage measurement data	-	SHOW COVERAGE
Displaying event	Disabled in performance mode	SHOW EVENT

2.2.4 On-the-fly Memory Access

While on-the-fly, the area mapped to the emulation memory is Read/Write enabled, but the area mapped to the user memory area is Read-only enabled.

■ Read/Write Memory while On-the-fly

The user memory cannot be accessed while on-the-fly (when execute the MCU). However, the emulation memory can be accessed. (The using cycle-steal algorithm eliminates any negative effect on the MCU speed.)

This emulator allows the user to use part of the emulation memory as a mirror area. The mirror area holds a copy of the user memory. Using this mirror area makes the Read-only enabled function available while on-the-fly.

Each memory area operates as follows:

- **User Memory Area**

  Access to the user memory is permitted only when the operation is suspended by a break.

- **Emulation Memory Area**

  Access to the emulation memory is permitted regardless of whether the MCU is suspended, or while on-the-fly.

- **Mirror Area**

  The emulation memory with the MIRROR setting can be set up for the user memory area to be referred to while on-the-fly. This area is specifically called the mirror area.

  As shown in Figure 2.2-1, the mirror area performs access to the user memory while the MCU is stopped, and such access is reflected simultaneously in the emulation memory specified as the mirror area. (Read access is also reflected in the emulation memory specified as the mirror area).

  In addition, as shown in Figure 2.2-2, access to the user memory by the MCU is reflected "as it is" in the emulation memory of the mirror area.

  While on-the -fly, the user memory cannot be accessed. However, the emulation memory specified as the mirror area can be read instead. In other words, identical data to that of the user memory can be read by accessing the mirror area

  However, at least one time access must be allowed before the emulation memory of the mirror area has the same data as the user memory. The following copy types allow the emulation memory of the mirror area to have the same data as the user memory.

  (1) Copying all data when setting mirror area

  When, /Copy is specified with the mirror area set using the SET MAP command, the whole area is specified, as the mirror area is copied.

  (2) Copying only required portion using memory access commands

  Data in the specified portion can be copied by executing a command that accesses memory. The following commands access memory.

- **Memory operation commands**

  SET MEMORY, SHOW MEMORY, EXAMINE, ENTER,
  COMPARE, FILL, MOVE, SEARCH MEMORY, DUMP,
  COPY, VERIFY
- **Data load/save commands**

  LOAD, SAVE

Figure 2.2-1 Access to Mirror Area while MCU Suspended

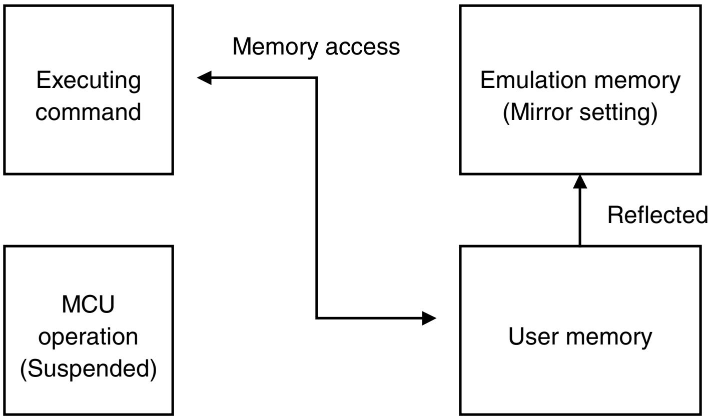
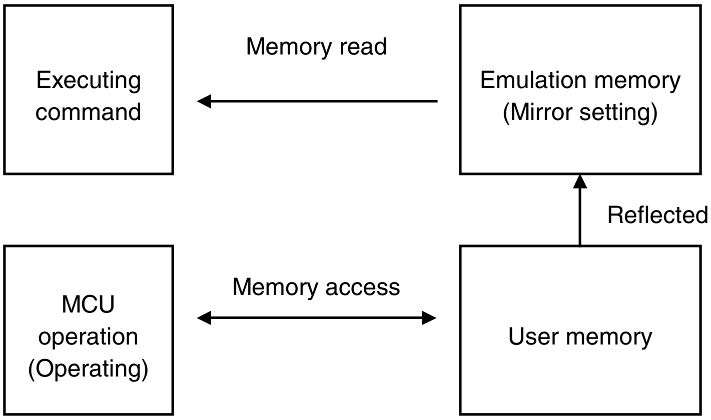


Figure 2.2-2 On-the-fly Access to Mirror Area



Note:

Memory access by a bus master other than the MCU is not reflected in the mirror area.

2.2.5 Events

The emulator can monitor the MCU bus operation, and generate a trigger at a specified condition called an event.
In this emulator, event triggers are used in order to determine which function event triggers are used accounting to event modes for the following functions;

- **Sequencer**
- **Sampling condition for multi-trace**
- **Measuring point in performance measurement**

■ Setting Events

A sequencer trigger can be generated under specified conditions by monitoring the operation of the MCU bus. This function is called an event.

The event provides code (/CODE) data access (/READ/WRITE).

Up to eight events can be set. Sharing hardware with trace triggers, however, the maximum settable count of events is actually as follows:

Current maximum count of events set
= 8 - (current set count of trace triggers + current set count of data monitoring breaks)

Table 2.2-5 shows the conditions that can be set for events.

Table 2.2-5 Conditions for Setting Events

Condition	Description
Address	Memory location (Address bit masking enabled)
Data	8-bit data (data bit masking enable) NOT specified enable
Status	Select from among dada read, data write, instruction execution and data modify.
External	8-bit data (bit masking enable)

Note:

In instruction execution, an event trigger is generated only when an instruction is executed. This status cannot be specified concurrently with other status.

Use the following commands to set an event.

SET EVENT:	Sets event
SHOW EVENT:	Display event setup status
CANCEL EVENT:	Deletes event
ENABLE EVENT:	Enable event

DISABLE EVENT: Disable event

[Example]

```
>SET EVENT  1,func1
>SET EVENT/WRITE 2,data[2],!d=h'10
>SET EVENT/MODIFY 3,102
```

An event can be set in the Event window as well.

■ Event Modes

There are three event modes as listed below. To determine which function event triggers are used for, select one using the SET MODE command. The default is normal mode.

The event value setting are made for each mode, so switching the event mode changes the event settings as well.

- **Normal Mode**

  Event triggers used for sequencer.

  Since the sequencer can perform control at 8 levels, it can control sequential breaks, time measurement and trace sampling. Real-time tracing in the normal mode is performed by single trace (tracing function that samples program execution continuously).

- **Multi Trace Mode**

  Event triggers used for multitracing (trace function that samples data before and after event trigger occurrence).

- **Performance Mode**

  Event triggers are used for performance measurement to measure time duration between two event trigger occurrences and count of event trigger occurrences.

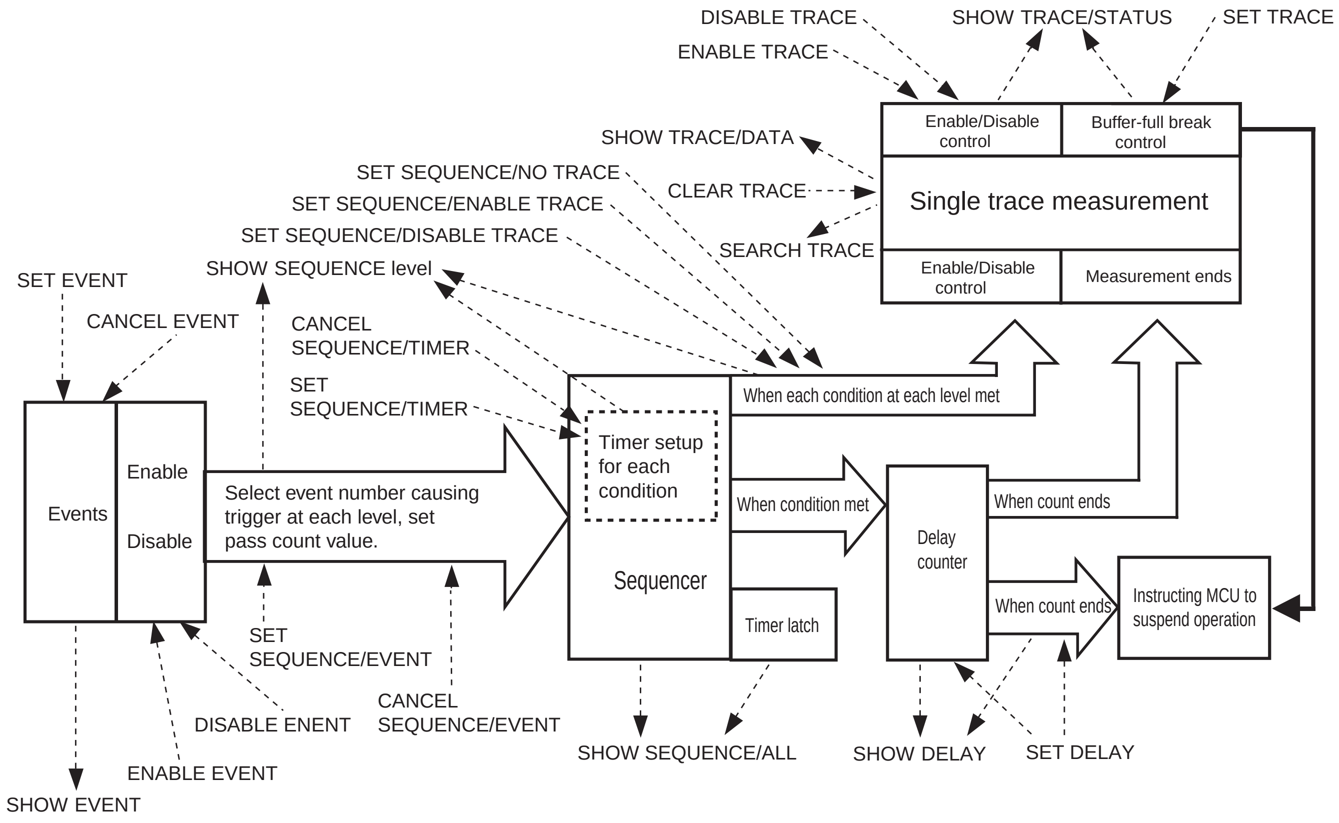
2.2.5.1 Operation in Normal Mode

As shown in the figure below, the event trigger set in the normal mode performs input to the sequencer. In the sequencer, either branching to any level, or terminating the sequencer, can be specified as an operation at event trigger occurrence. This enables debugging (breaks, limiting trace, measuring time) while monitoring program flow.

■ Operation in Normal Mode

The termination of sequencer triggers the delay counter. When the delay counter reaches the specified count, sampling for the single trace terminates. A break normally occurs at this point, but if necessary, the program can be allowed to run on without a break.

Figure 2.2-3 Operation in Normal Mode



■ Event-related Commands in Normal Mode

Since the real-time trace function in the normal mode is actually the single trace function, the commands can be used to control.

Table 2.2-6 shows the event-related commands that can be used in the normal mode.

Table 2.2-6 Event-related Commands in Normal Mode

Mode	Usable Command	Function
Normal Mode	SET EVENT SHOW EVENT CANCEL EVENT ENABLE EVENT DISABLE EVENT	Set event Displays event setup status Delete event Enables event Disables event
	SET SEQUENCE SHOW SEQUENCE CANCEL SEQUENCE ENABLE SEQUENCE DISABLE SEQUENCE	Sets sequencer Displays sequencer setup status Cancels sequencer Enables sequencer Disables sequencer
	SET DELAY SHOW DELAY	Sets delay count Displays delay count setup status
	SET TRACE SHOW TRACE SEARCH TRACE ENABLE TRACE DISABLE TRACE CLEAR TRACE	Sets trace buffer-full break Displays trace data Searches trace data Enables trace function Disables trace function Clears trace data

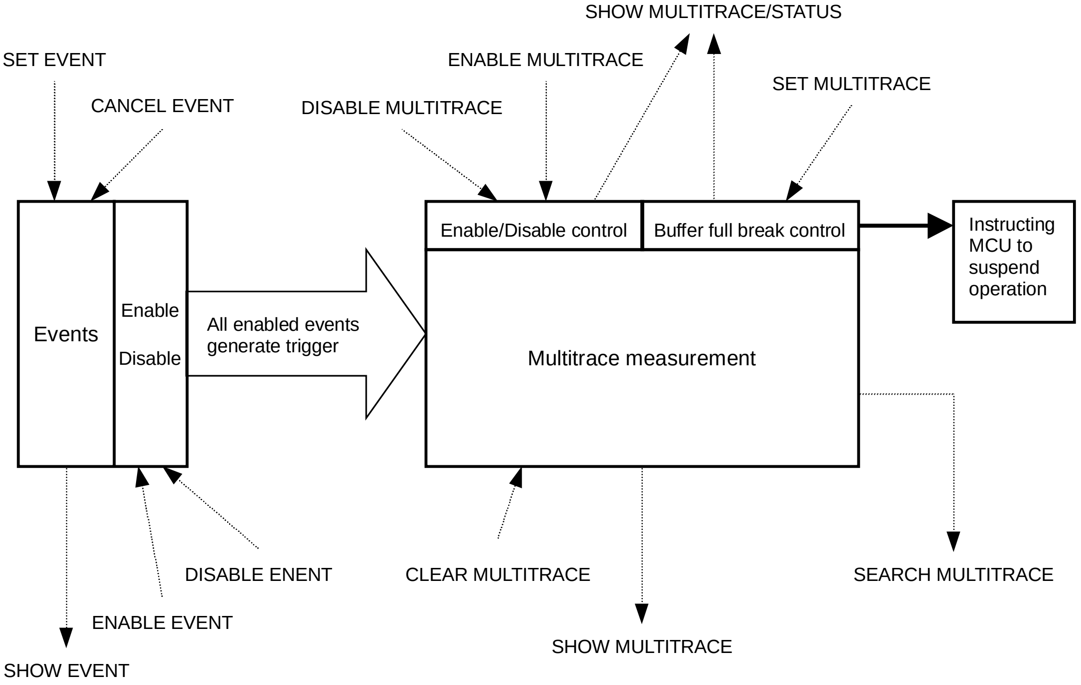
2.2.5.2 Operation in Multi Trace Mode

When the multitrace mode is selected as the event mode, the real-time trace function becomes the multitrace function, and events are used as triggers for multitracing.

■ Operation in Multi Trace Mode

Multitracing is a trace function that samples data before and after an event trigger occurrence. When the multitrace mode is selected as the event mode, the real-time trace function becomes the multitrace function, and events are used as triggers for multitracing.

Figure 2.2-4 Operation in Multi Trace Mode



■ Event-related Commands in Multi Trace Mode

Table 2.2-7 shows the event-related commands that can be used in the multi-race mode.

Table 2.2-7 Event-related Commands in Multi Trace Mode

Mode	Usable Command	Function
Multi Trace Mode	SET EVENT SHOW EVENT CANCEL EVENT ENABLE EVENT DISABLE EVENT	Sets event Displays event setup status Deletes event Enables event Disables event
	SET MULTITRACE SHOW MULTITRACE SEARCH MULTITRACE ENABLE MULTITRACE DISABLE MULTITRACE CLEAR MULTITRACE	Sets trace buffer-full break Displays trace data Searches trace data Enables trace function Disables trace function Clears trace data

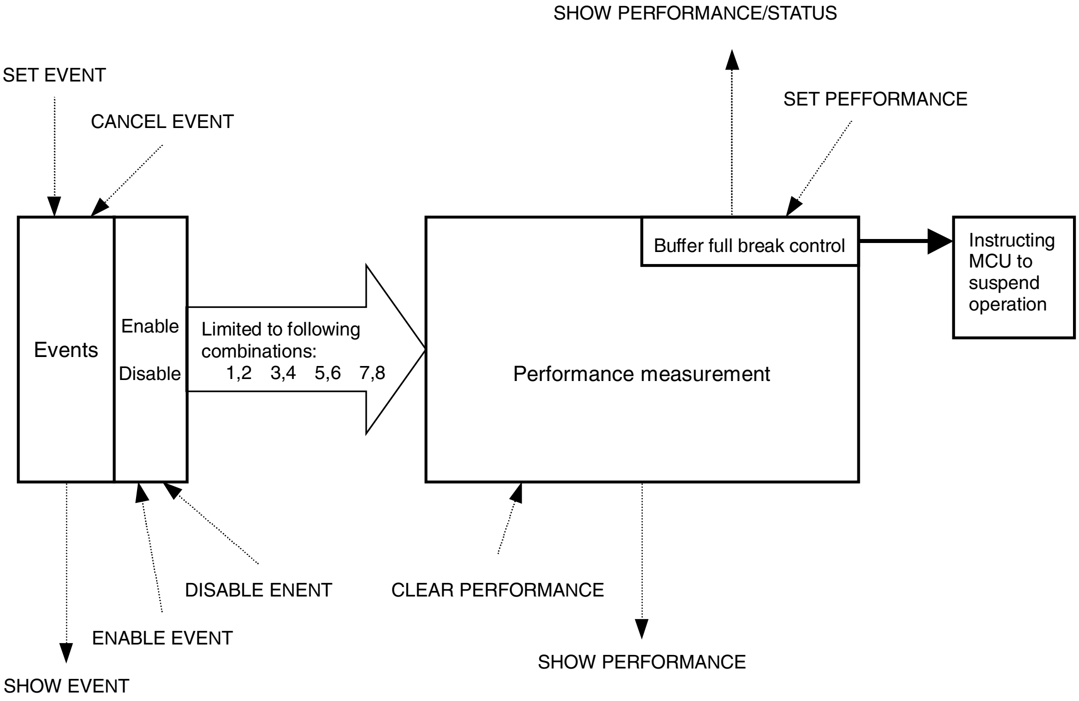
2.2.5.3 Operation in Performance Mode

Event triggers set in the performance mode are used to measure performance. The time duration between two event occurrences can be measured and the event occurrences can be counted.

■ Operation in Performance Mode

The event triggers that are set in the performance mode are used to measure performance. The time duration between two event occurrences can be measured and the event occurrences can be counted.

Figure 2.2-5 Operation in Performance Mode



■ Event-related Commands in Performance Mode

Table 2.2-8 shows the event-related commands that can be used in the performance mode.

Table 2.2-8 Event-related Commands in Performance Mode

Mode	Usable Command	Function
Performance Mode	SET EVENT SHOW EVENT CANCEL EVENT ENABLE EVENT DISABLE EVENT	Sets event Displays event setup status Deletes event Enables event Disables event
	SET PERFORMANCE SHOW PERFORMANCE CLEAR PERFORMANCE	Sets performance Displays performance setup status Clears performance measurement data

2.2.6 Control by Sequencer

This emulator has a sequencer to control events. By using this sequencer, sampling of breaks, time measurement and tracing can be controlled while monitoring program flow (sequence). A break caused by this function is called a sequential break.
To use this function, set the event mode to normal mode using the SET MODE command.
Use the SET EVENT command to set events.

■ Control by Sequencer

As shown in Table 2.2-9, controls can be made at 8 different levels.

At each level, 8 events and 1 timer condition (9 conditions in total) can be set.

A timer condition is met when the timer count starts at entering a given level and the specified time is reached.

For each condition, the next operation can be specified when the condition is met. Select any one of the following.

- Move to required level.
- Terminate sequencer.

The conditions set for each level are determined by OR. Therefore, if any one condition is met, the sequencer either moves to the required level, or terminates. In addition, trace sampling suspend/resume can be controlled when a condition is met.

Table 2.2-9 Sequencer Specifications

Function	Specifications
Level count	8 levels
Conditions settable for each level	8 event conditions (1 to 16777216 times pass count can be specified for each condition.) 1 timer condition (Up to 16 s. in µs units or up to 1.6 s. in 100 ns units can be specified.*)
Operation when condition met	Branches to required level or terminates sequence. Controls trace sampling.
Other function	Timer latch enable at level branching
Operation when sequencer terminates	Starts delay counter

*:The minimum measurement unit for Timer value can be set to either 1 µs or 100 ns using the SET TIMERSCALE command.

2.2.6.1 Setting Sequencer

The sequencer operates in the following order:

(1) The sequencer starts from level 1 simultaneously with the start of program executing.

(2) Depending on the setting at each level, branching to the required level is performed when the condition is met.

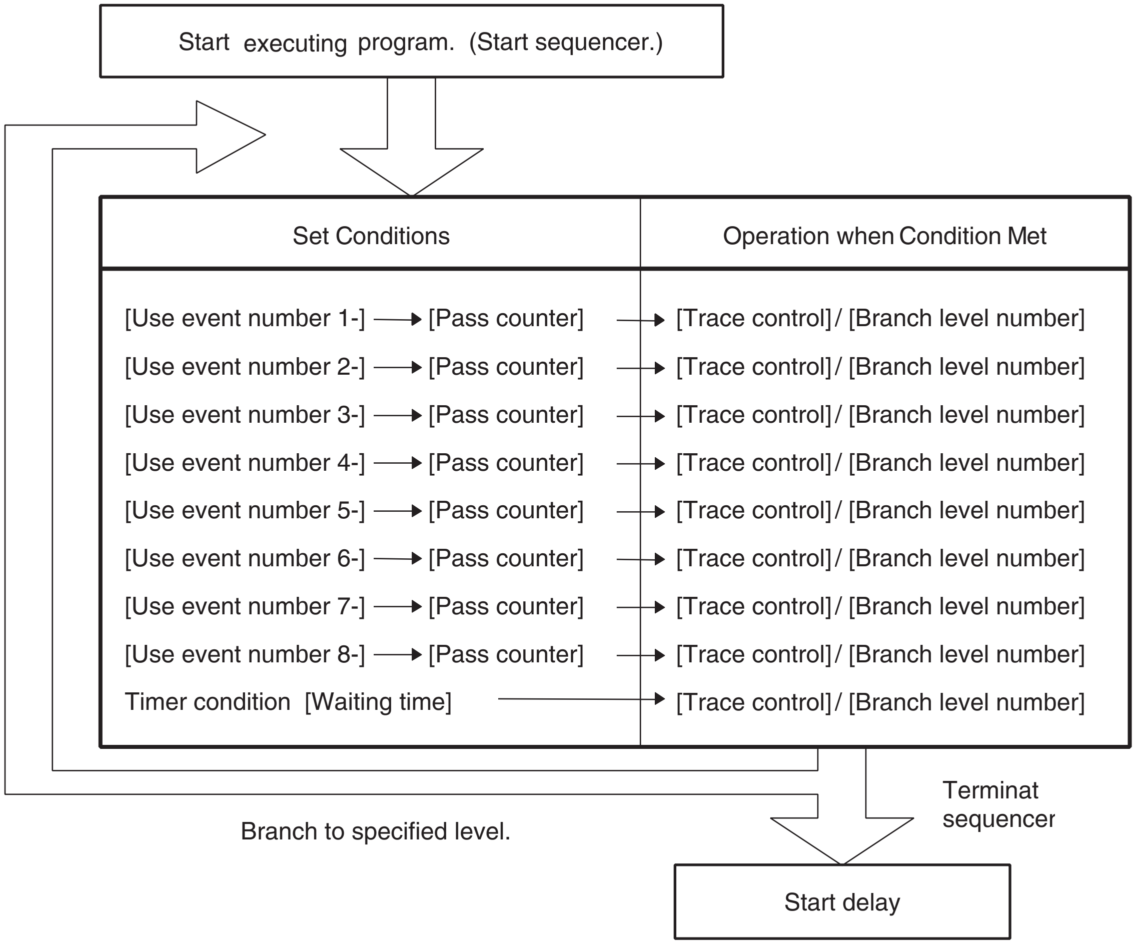
(3) When sequencer termination is specified, the sequencer terminates when the condition is met.

(4) When the sequencer terminates, the delay counter starts counting.

■ Setting Sequencer

Figure 2.2-6 shows the sequencer operation.

Figure 2.2-6 Operation of Sequencer


Start executing program. (Start sequencer.)
Branch to specified level.
Terminat sequencer
Start delay

Set Conditions	Operation when Condition Met
[Use event number 1-] → [Pass counter]	→ [Trace control]/ [Branch level number]
[Use event number 2-] → [Pass counter]	→ [Trace control]/ [Branch level number]
[Use event number 3-] → [Pass counter]	→ [Trace control]/ [Branch level number]
[Use event number 4-] → [Pass counter]	→ [Trace control]/ [Branch level number]
[Use event number 5-] → [Pass counter]	→ [Trace control]/ [Branch level number]
[Use event number 6-] → [Pass counter]	→ [Trace control]/ [Branch level number]
[Use event number 7-] → [Pass counter]	→ [Trace control]/ [Branch level number]
[Use event number 8-] → [Pass counter]	→ [Trace control]/ [Branch level number]
Timer condition [Waiting time]	→ [Trace control]/ [Branch level number]

[Setup Examples]

- Terminate sequencer when event 1 occurs.

  >SET SEQUENCE/EVENT 1,1,J=0

- Terminate sequencer when event 2 occurs 16 times.

  >SET SEQUENCE/EVENT 1,2,16,J=0

- Terminate sequencer when event 2 occurs after event 1 occurred. However, do not terminate sequencer if event 3 occurs between event 1 and event 2.

  >SET SEQUENCE/EVENT 1,1,J=2

  >SET SEQUENCE/EVENT 2,2,J=0

  >SET SEQUENCE/EVENT 2,3,J=1

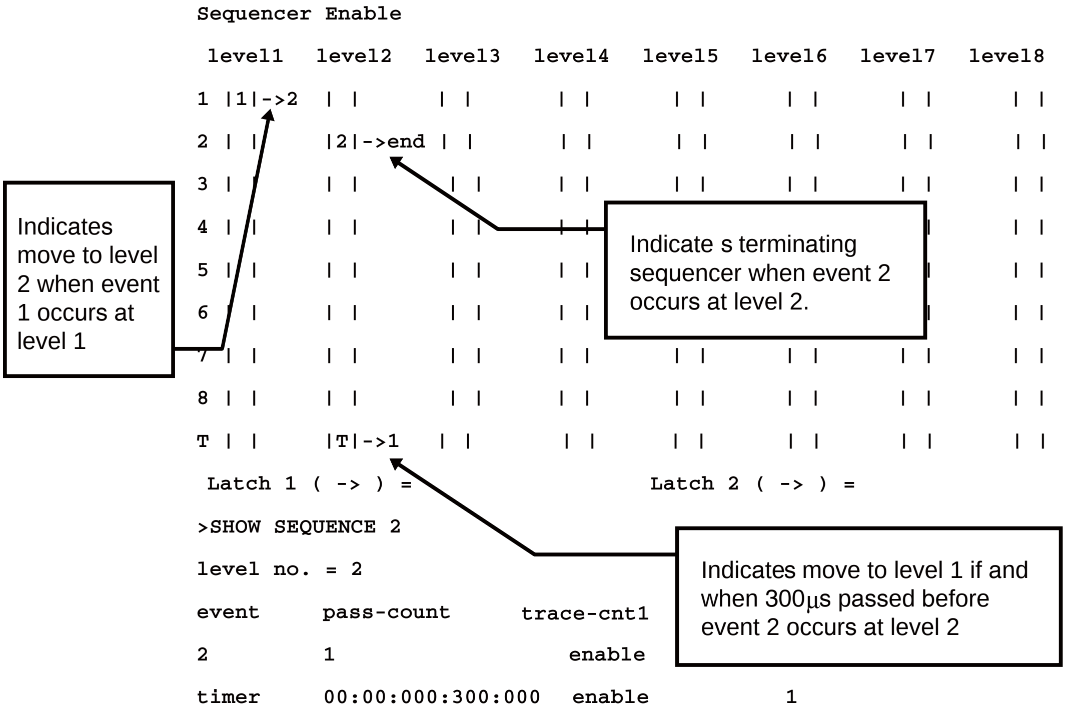
- Terminate sequencer if and when event 2 occurs less than 300 μs after event 1 occurred.

  >SET SEQUENCE/EVENT 1,1,J=2

  >SET SEQUENCE/EVENT 2,2,J=0

  >SET SEQUENCE/TIMER 2,300,J=1

  >SHOW SEQUENCE



```
Sequencer Enable
 level1   level2   level3   level4   level5   level6   level7   level8
1 |1|->2  | |      | |       | |      | |      | |      | |      | |
2 | |     |2|->end | |       | |      | |      | |      | |      | |
3 | |     | |       | |      | |      | |      | |      | |      | |
4 | |     | |       | |      | |                         |      | |
5 | |     | |       | |      | |                         |      | |
6 | |     | |       | |      | |                         |      | |
7 | |     | |       | |      | |      | |      | |      | |      | |
8 | |     | |       | |      | |      | |      | |      | |      | |
T | |     |T|->1   | |       | |     | |      | |      | |      | |
 Latch 1 ( -> ) =                  Latch 2 ( -> ) =
>SHOW SEQUENCE 2
level no. = 2
event     pass-count      trace-cnt1
2         1                   enable
timer     00:00:000:300:000   enable          1
```

Indicates move to level 2 when event 1 occurs at level 1

Indicate s terminating sequencer when event 2 occurs at level 2.

Indicates move to level 1 if and when 300μs passed before event 2 occurs at level 2

2.2.6.2 Break by Sequencer

A program can suspend program execution when the sequencer terminates. This break is called a sequential break.

■ Break by Sequencer

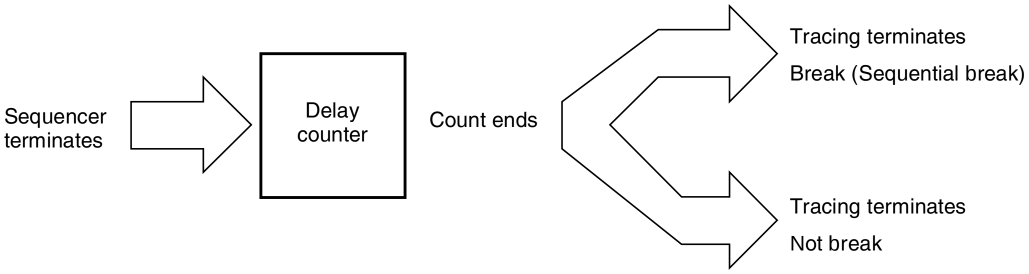
A program can suspend program execution when the sequencer terminates. This break is called a sequential break.

As shown in Figure 2.2-7, the delay count starts when the sequencer terminates, and after delay count ends, either "break" or "not break but tracing only terminates" is selected as the next operation.

To make a break immediately after the sequencer terminates, set delay count to 0 and specify "Break after delay count terminates". Use the SET DELAY command to set the delay count and the operation after the delay count.

The default is delay count 0, and Break after delay count.

Figure 2.2-7 Operation when sequencer terminates



[Examples of Delay Count Setups]

- Break when sequencer terminates.

  >SET DELAY/BREAK 0

- Break when 100-bus-cycle tracing done after sequencer terminates.

  >SET DELAY/BREAK 100

- Terminate tracing, but do not break when sequencer terminates.

  >SET DELAY/NOBREAK 0

- Terminate tracing, but do not break when 100-bus-cycle tracing done after sequencer terminates.

  >SET DELAY/NOBREAK 100

2.2.6.3 Trace Sampling Control by Sequencer

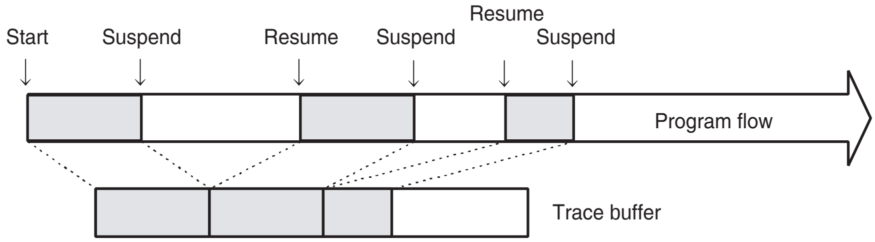
When the event mode is in the normal mode, real-time trace executing tracing called single trace.
If the trace function is enabled, single trace samples all the data from the start of executing a program until the program is suspended.

■ Trace Sampling Control by Sequencer

Sets up suspend/resume trace sampling for each condition at each level of the sequencer. Figure 2.2-8 shows the trace sampling flow.

For example, it is possible to suspend trace sampling when event 1 occurs, and then resume trace sampling when event 2 occurs. Trace data sampling can be restricted.

Figure 2.2-8 Trace Sampling Control (1)

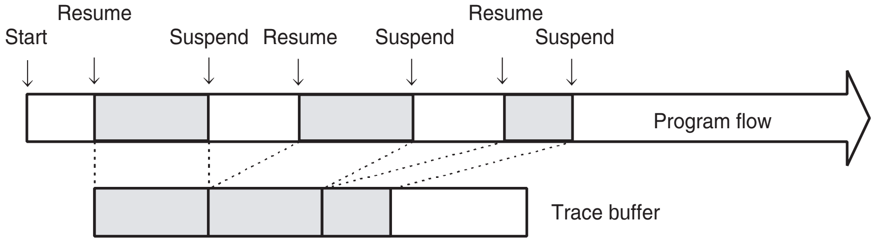


As shown in Figure 2.2-9, trace sampling can be disabled during the period from the start of a program execution until the first condition occurs. For this setup, use the GO command or the SET GO command.

[Example]

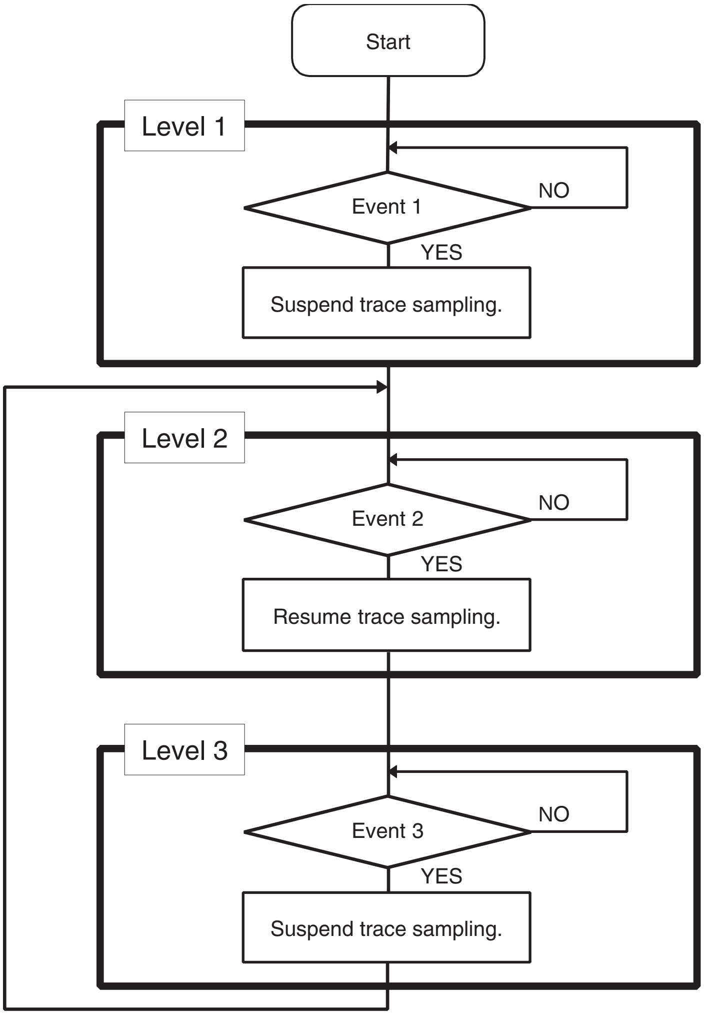
```
>GO/DISABLETRACE
>SET GO/DISABLETRACE
>GO
```

Figure 2.2-9 Trace Sampling Control (2)



[Setup Example]

Suspend trace sampling when event 1 occurs, and then resume at event 2 and keep sampling data until event 3 occurs.



```
>SET SEQUENCE/EVENT/DISABLETRACE 1,1,J=2
>SET SEQUENCE/EVENT/ENABLETRACE 2,2,J=3
>SET SEQUENCE/EVENT/DISABLETRACE 3,3,J=2
```

2.2.6.4 Time Measurement by Sequencer

Time can be measured using the sequencer. A time measurement timer called the emulation timer is used for this purpose. When branching is made from a specified level to another specified level, a timer value is specified. Up to two emulation timer values can be fetched. This function is called the timer latch function.

■ Time Measurement by Sequencer

The time duration between two given points in a complex program flow can be measured using the timer latch function.

The timing for the timer latch can be set using the SET SEQUENCE command; the latched timer values can be displayed using the SHOW SEQUENCE command.

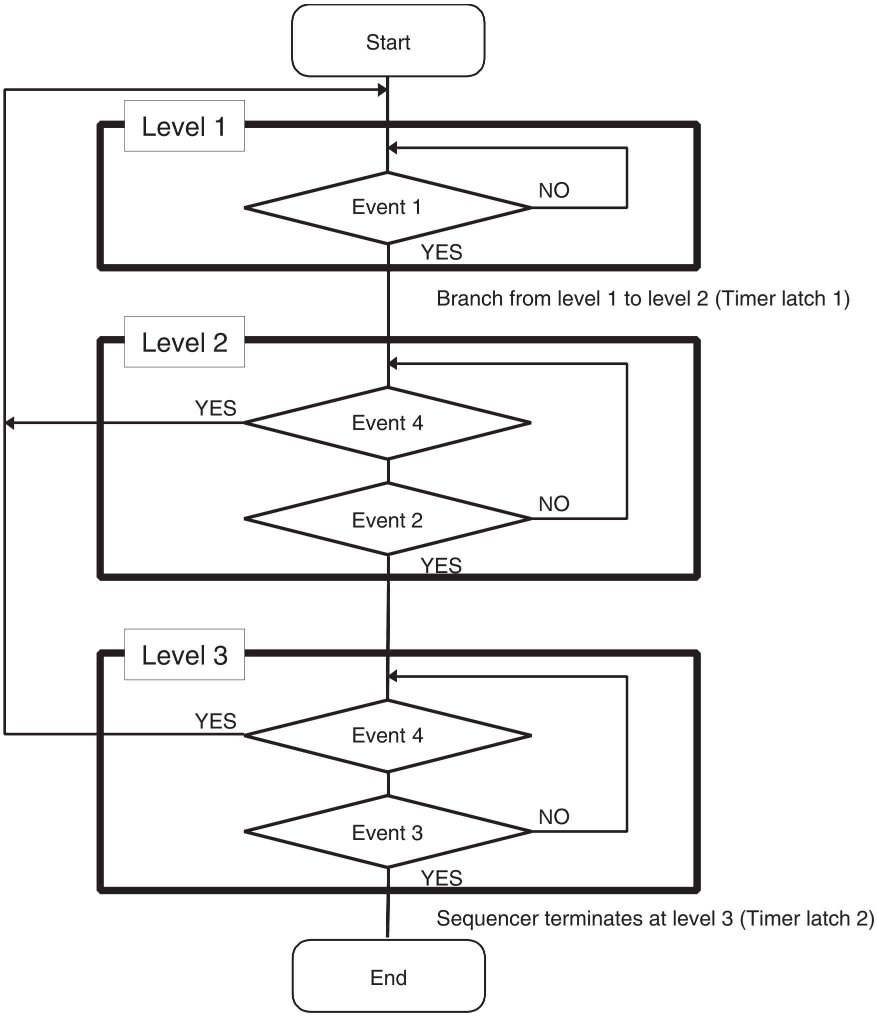
When a program starts execution, the emulation timer is initialized and then starts counting. Select either 1 μs or 100 ns as the minimum measurement unit for the emulation timer. Set the measurement unit using the SET TIMESCALE command.

When 1 us is selected, the maximum measured time is about 70 minutes; when 100 ns is selected, the maximum measured time is about 7 minutes. If the timer overflows during measurement, a warning message is displayed when the timer value is displayed using the SHOW SEQUENCE command.

2.2.6.5 Sample Flow of Time Measurement by Sequencer

In the following sample, when events are executed in the order of Event 1, Event 2 and Event 3, the execution time from the Event 1 to the Event 3 is measured. However, no measurement is made if Event 4 occurs anywhere between Event 1 and Event 3.

■ Sample Flow of Time Measurement by Sequencer


Start
Level 1
Event 1
NO
YES
Branch from level 1 to level 2 (Timer latch 1)
Level 2
YES
Event 4
Event 2
NO
YES
Level 3
YES
Event 4
Event 3
NO
YES
Sequencer terminates at level 3 (Timer latch 2)
End

```
>SET SEQUENCE/EVENT  1,1,J=2

>SET SEQUENCE/EVENT  2,4,J=1

>SET SEQUENCE/EVENT  2,2,J=3

>SET SEQUENCE/EVENT  3,4,J=1

>SET SEQUENCE/EVENT  3,2,J=0

>SET SEQUENCE/LATCH  1,1,2

>SET SEQUENCE/LATCH  2,3,0


>SHOW SEQUENCE

Sequencer Enable

 level1  level2   level3   level4   level5   level6   level7   level8

1 |1|#>2  | |      | |      | |      | |      | |      | |      | |

2 | |      |2|->3   | |      | |      | |      | |      | |      | |

3 | |      | |      |3|#end  | |      | |      | |      | |      | |

4 | |      |4|->1   |4|->1   | |      | |      | |      | |      | |

5 | |      | |      | |      | |      | |      | |      | |      | |

6 | |      | |      | |      | |      | |      | |      | |      | |

7 | |      | |      | |      | |      | |      | |      | |      | |

8 | |      | |      | |      | |      | |      | |      | |      | |

T | |      |T|->1   | |      | |      | |      | |      | |      | |

 Latch 1 (1->2) = 00m02s060ms379.0µs  Latch 2 (3->E) = 00m16s040ms650.0us
```

Indicates that, if event 3 occurs at level 3, the sequencer terminates and let the timer latched.

Indicates that, if event 1 occurs at level 1, move to level 2 and let the timer latched.

Indicate time values of timer latch 1 and timer latch 2. The time value, deducting the value of the timer latch 1 from the value of the timer latch 2, represents the execution time.
Time is displayed in the following format.

00 m	00 s	000 ms	000.0 us
-	-	-	-
minutes	seconds	milliseconds	microseconds

2.2.7 Real-time Trace

While execution a program, the address, data and status information, and the data sampled by an external probe can be sampled in machine cycle units and stored in the trace buffer. This function is called real-time trace.
In-depth analysis of a program execution history can be performed using the data recorded by real-time trace.
There are two types of trace sampling: single trace, which traces from the start of executing the program until the program is suspended, and multitrace, which starts tracing when an event occurs.

■ Trace Buffer

The data recorded by sampling in machine cycle units, is called a frame.

The trace buffer can store 32K frames (32768). Since the trace buffer has a ring structure, when it becomes full, it automatically returns to the start to overwrite existing data.

■ Trace Data

Data sampled by the trace function is called trace data.

The following data is sampled:

- Address
- Data
- Status Information
    - Access status: Read/Write/Internal access, etc.
    - Device status: Instruction execution, Reset, Hold, etc.
    - Queue status: Count of remaining bytes of instruction queue, etc.
    - Data valid cycle information: Data valid/invalid

      (Since the data signal is shared with other signals, it does not always output data. Therefore, the trace samples information indicating whether or not the data is valid.)
- External probe data
- Sequencer execution level

■ Data Not Traced

The following data does not leave access data in the trace buffer.

- **Data after tool hold**

  The F^2MC-16/16L/16LX/16H/16F family execute the following operation immediately after a break, etc., lets MCU suspend (a tool hold). This data is not displayed because it is deleted from the trace buffer.

  - Access to address 100
  - Access to FFFFDC to FFFFFF

- **Portion of access data while native mode.**

  When operating in the native mode, the F^2MC-16/16L/16LX/16H/16F family of chips sometime performs simultaneous multiple bus operations internally. However, in this emulator, monitoring of the internal ROM bus takes precedence. Therefore, other bus data being accessed simultaneously may not be sampled (in the debugging mode, all operations are sampled).

2.2.7.1 Single Trace

The single trace traces all data from the start of executing a program until the program is aborted.

■ Function of Single Trace

The single trace is enabled by setting the event mode to normal mode using the SET MODE command.

The single trace traces all data from the start of executing a program until the program is suspended.

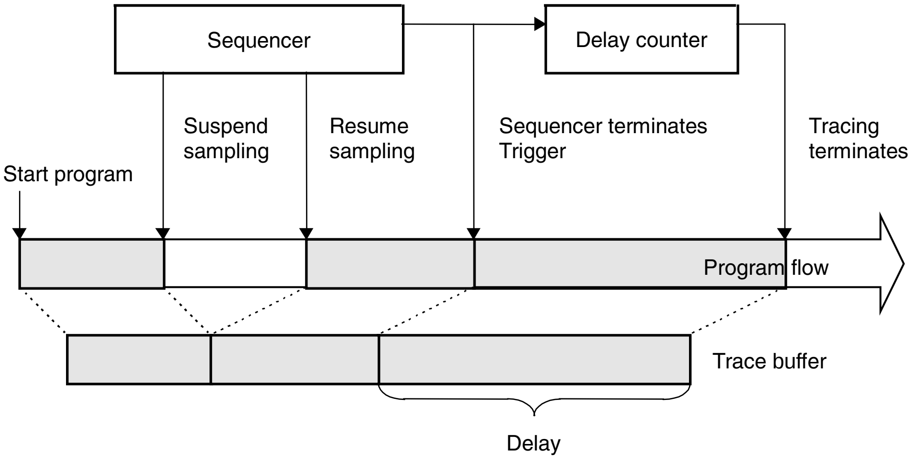
If the real-time trace function is enabled, data sampling continues execution to record the data in the trace buffer while the GO, STEP, CALL commands are being executed.

As shown in Figure 2.2-10, suspend/resume trace sampling can be controlled by the event sequencer. Since the delay can be set between the sequencer terminating the trigger and the end of tracing, the program flow after an given event occurrence can be traced. The delay count is counted in pass cycle units, so it matches the sampled trace data count. However, nothing can be sampled during the delay count if trace sampling is suspended when the sequencer is terminated.

After the delay count ends, a break occurs normally due to the sequential break, but tracing can be terminated without a break.

Furthermore, a program can be allowed to break when the trace buffer becomes full. This break is called a trace-buffer-full break.

Figure 2.2-10 Sampling in Single Trace



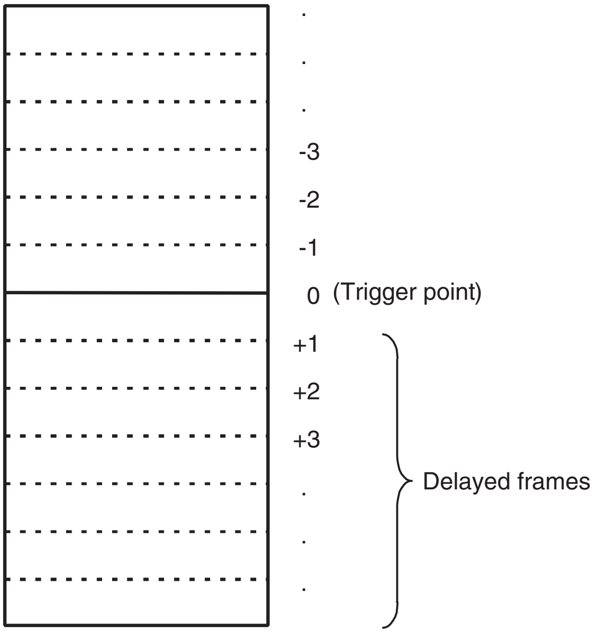
■ Frame Number and Step Number in Single Trace

The sampled trace data is numbered in frame units. This number is called the frame number.

When displaying trace data, the starting location in the trace buffer can be specified using the frame number. The trace data at the point where the sequencer termination trigger occurs is numbered 0; trace data sampled before reaching the trigger point is numbered negatively, and the data sampled after the trigger point is numbered positively (See Figure 2.2-11).

If there is no sequencer termination trigger point available, the trace data sampled last is numbered 0.

Figure 2.2-11 Frame Number in Single Trace



This program can analyze the single trace result and sort the buffer data in execution instruction units (only when the MCU execution mode is the debugging mode).

In this mode, the following information is grouped as one unit, and each information unit is numbered. This number is called the step number.

- Execution instruction mnemonic information
- Data access information
- Device status information

The step number at the sequencer termination trigger is numbered 0; information sampled before reaching the trigger point is numbered negatively, and information sampled after the trigger point is numbered positively.

If there is no sequencer termination trigger point, the information sampled last is numbered 0.

2.2.7.2 Setting Single Trace

The following settings (1) to (4) are required before executing single trace. Once these settings have been made, trace data is sampled when a program is executed.

(1) Set event mode to normal mode.

(2) Enable trace function.

(3) Set events, sequencer, and delay count.

(4) Set trace-buffer-full break.

■ Setting Single Trace

The following settings are required before executing single trace. Once these settings have been made, trace data is sampled when a program is executed.

(1) Set event mode to normal mode.

Use SET MODE command to make this setting.

(2) Enable trace function.

Use the ENABLE TRACE command. To disable the function, use the DISABLE TRACE command. The default is Enable.

(3) Set events, sequencer, and delay count.

Trace sampling can be controlled by setting the sequencer for events. If this function is not needed, there is no need of this setting.

To set events, use the SET EVENT command. To set the sequencer, use the SET SEQUENCE command.

Furthermore, set the delay count between sequencer termination and trace ending, and the break operation (Break or Not Break) when the delay count ends. If the data after event occurrence is not required, there is no need of this setting.

If Not Break is set, the trace terminates but no break occurs. To check trace data on-the-fly, use this setup by executing the SET DELAY command.

Note:

When the sequencer termination causes a break (sequential break), the last executed machine cycle is not sampled.

(4) Set trace-buffer-full break.

The program can be allowed to break when the trace buffer becomes full. Use the SET TRACE command for this setting. The default is Not Break. Display the setup status using the SHOW TRACE/STATUS command.

Table 2.2-10 lists trace-related commands that can be used in the single trace function.

Table 2.2-10 Trace-related Commands That Can Be Used in The Single Trace Function

Usable Command	Function
SET EVENT SHOW EVENT CANCEL EVENT ENABLE EVENT DISABLE EVENT	Sets events Displays event setup status Deletes event Enables event Disables event
SET SEQUENCE SHOW SEQUENCE CANCEL SEQUENCE ENABLE SEQUENCE DISABLE SEQUENCE	Sets sequencer. Displays sequencer setting status Cancels sequencer Enables sequencer Disables sequencer
SET DELAY SHOW DELAY	Sets delay count value, and operation after delay Displays delay count setting status
SET TRACE SHOW TRACE SEARCH TRACE ENABLE TRACE DISABLE TRACE CLEAR TRACE	Set traces-buffer-full break Displays trace data Searches trace data Enables trace function Disables trace function Clears trace data

2.2.7.3 Multi trace

The multi trace samples data where an event trigger occurs for 8 frames before and after the event trigger.

■ Multi Trace Function

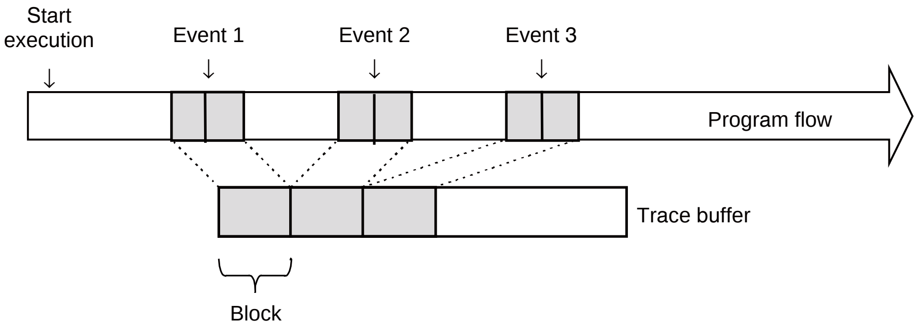
Execute multitrace by setting the event mode to the multitrace mode using the SET MODE command.

The multitrace samples data where an event trigger occurs for 8 frames before and after the event trigger.

It can be used for tracing required only when a certain variable access occurs, instead of continuous tracing.

The trace data sampled at one event trigger (16 frames) is called a block. Since the trace buffer can hold 32K frames, up to 2048 blocks can be sampled. Multi trace sampling terminates when the trace buffer becomes full. At this point, a executing program can be allowed to break if necessary.

Figure 2.2-12 Multi Trace Sampling



■ Multi Trace Frame Number

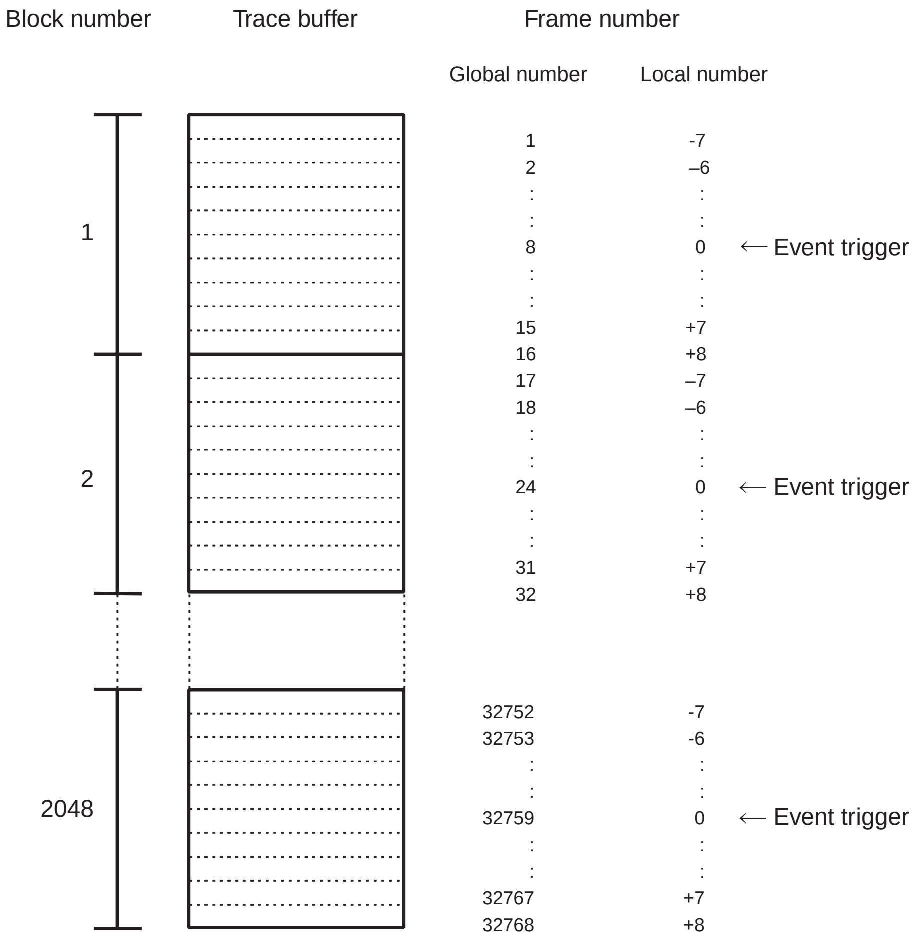
Sixteen frames of data are sampled each time an event occurs. This data unit is called a block, and each sampled block is numbered starting from 0. This is called the block number.

A block is a collection of 8 frames of sampled data before and after the event trigger occurs. At the event trigger is 0, trace data sampled before reaching the event trigger point is numbered negatively, and trace data sampled after the event trigger point is numbered positively. These frame numbers are called local numbers (See Figure 2.2-13).

In addition to this local number, there is another set of frame numbers starting with the oldest data in the trace buffer. This is called the global number. Since the trace buffer can hold 32K frames, frames are numbered 1 to 32758 (See Figure 2.2-13).

To specify which frame data is displayed, use the global number or block and local numbers.

Figure 2.2-13 Frame Number in Multi Trace



2.2.7.4 Setting Multi Trace

Before executing the multitrace, the following settings must be made. After these settings, trace data is sampled when a program is executed.

(1) Set event mode to multitrace mode.

(2) Enable trace function.

(3) Set event.

(4) Set trace-buffer-full break.

■ Setting Multi Trace

Before executing the multitrace, the following settings must be made. After these settings, trace data is sampled when a program is executed.

(1) Set event mode to multitrace mode.

Use the SET MODE command for this setting.

(2) Enable trace function.

Use the ENABLE MULTITRACE command. To disable the function, use the DISABLE MULTITRACE command.

(3) Set event.

Set an event that sampling. Use the SET EVENT command for this setting.

(4) Set trace-buffer-full break.

To break when the trace buffer becomes full, set the trace-buffer-full break. Use the SET MULTITRACE command for this setting.

Table 2.2-11 shows the list of trace-related commands that can be used in multitrace mode.

Table 2.2-11 Trace-related Commands That Can Be Used in Multi Trace Mode

Usable Command	Function
SET EVENT SHOW EVENT CANCEL EVENT ENABLE EVENT DISABLE EVENT	Sets events Displays event setup status Deletes event Enables event Disables event
SET MULTITRACE SHOW MULTITRACE SEARCH MULTITRACE ENABLE MULTITRACE DISABLE MULTITRACE CLEAR MULTITRACE	Sets trace-buffer-full break Displays trace data Searches trace data Enables multitrace Disables multitrace Clears trace data

2.2.7.5 Displaying Trace Data Storage Status

It is possible to Displays how much trace data is stored in the trace buffer. This status data can be read by specifying /STATUS to the SHOW TRACE command in the single trace mode, and to the SHOW MULTITRACE command in the multitrace mode.

■ Displaying Trace Data Storage Status

It is possible to Displays how much trace data is stored in the trace buffer. This status data can be read by specifying/STATUS to the SHOW TRACE command in the single trace mode, and to the SHOW MULTITRACE command in the multitrace.

Frame numbers displayed in the multitrace mode is the global number.

[Example]

- In Single Trace

```
 >SHOW TRACE/STATUS
en/dis      = enable                Trace function enabled
buffer full = nobreak               Buffer full break function disabled
sampling    = end                   Trace sampling terminates
flame no. = -00120 to  00050        Frame -120 to 50 store data
step no.  = -00091 to  00022        Step -91 to 22 store data
 >
```

- In Multi trace

```
 >SHOW MULTITRACE/STATUS
en/dis      = enable                Multi trace function enabled
buffer full = nobreak               Buffer full break function disabled
sampling    = end                   Trace sampling terminates
block no. = 1 to 5                  Block 1 to 5 store data
frame no. = 00001 to 00159          Frame 1 to 159 store data
                                    (Global number)
```

2.2.7.6 Specify Displaying Trace Data Start

It is possible to specify from which data in the trace buffer to display. To do so, specify a frame number with the SHOW TRACE command in the single trace mode, or specify either a global number, or a block number and local number with the SHOW MULTITRACE command in the multitrace mode. A range can also be specified.

■ Specifying Displaying Trace Data Start

It is possible to specify from which data in the trace buffer to displays. To do this, specify a frame number with the SHOW TRACE command in the single trace, and specify either a global number, or a block number and local number with the SHOW MULTITRACE command in the multitrace. A range can also be specified.

[Example]

- In Single Trace Mode

```
>SHOW TRACE/CYCLE -6          Start displaying from frame -6
>SHOW TRACE/CYCLE -6..10      Display from frame -6 to frame 10
>SHOW TRACE -6                Start displaying from step -6
>SHOW TRACE -6..10            Displays from step -6 to step 10
```

Note:

A step number can only be specified when the MCU execution mode is set to the debugging mode.

- In Multi trace

```
>SHOW MULTITRACE/GLOBAL 500       Start displaying from frame 500 (Global number)
>SHOW MULTITRACE/LOCAL 2          Displaying block number 2
>SHOW MULTITRACE/LOCAL 2,-5..5    Display from frame -5 to frame 5 of block number 2
```

2.2.7.7 Display Format of Trace Data

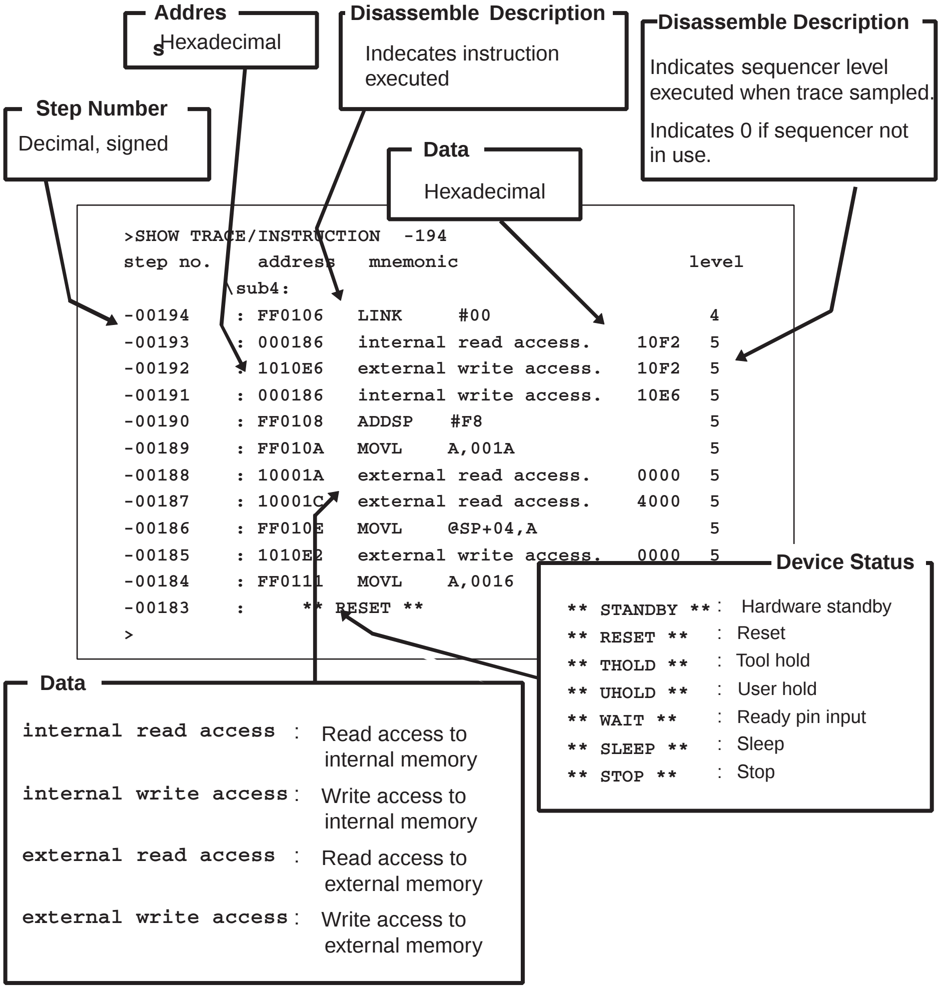
A display format can be chosen by specifying a command identifier with the SHOW TRACE command in the single trace, and with the SHOW MULTITRACE command in the multitrace.
The source line is also displayed if "Add source line" is selected using the SET SOURCE command.
There are three formats to display trace data:

- Display in instruction execution order	**(Specify /INSTRUCTION.)**
- Display all machine cycles	**(Specify /CYCLE.)**
- Display in source line units	**(Specify /SOURCE.)**

■ Display in instruction Execution Order (Specify /INSTRUCTION.)

Trace sampling is performed at each machine cycle, but the sampling results are difficult to Display because they are influenced by pre-fetch, etc. This is why the emulator has a function to allow it to analyze trace data as much as possible. The resultant data is displayed after processes such as eliminating pre-fetch effects, analyzing execution instructions, and sorting in instruction execution order are performed automatically. However, this function can be specified only in the single trace while in the debugging mode.

In this mode, data can be displayed in the following format.



```
>SHOW TRACE/INSTRUCTION  -194
step no.    address   mnemonic                     level
            sub4:
-00194    : FF0106   LINK     #00                    4
-00193    : 000186   internal read access.    10F2   5
-00192    : 1010E6   external write access.   10F2   5
-00191    : 000186   internal write access.   10E6   5
-00190    : FF0108   ADDSP   #F8                     5
-00189    : FF010A   MOVL    A,001A                  5
-00188    : 10001A   external read access.    0000   5
-00187    : 10001C   external read access.    4000   5
-00186    : FF010E   MOVL    @SP+04,A                5
-00185    : 1010E2   external write access.   0000   5
-00184    : FF0111   MOVL    A,0016
-00183    :     ** RESET **
>
```

Addres
s
Hexadecimal
Disassemble Description
Indecates instruction executed
Disassemble Description
Indicates sequencer level executed when trace sampled.
Indicates 0 if sequencer not in use.
Step Number
Decimal, signed
Data
Hexadecimal
Device Status
** STANDBY ** : Hardware standby
** RESET ** : Reset
** THOLD ** : Tool hold
** UHOLD ** : User hold
** WAIT ** : Ready pin input
** SLEEP ** : Sleep
** STOP ** : Stop
Data
internal read access : Read access to internal memory
internal write access : Write access to internal memory
external read access : Read access to external memory
external write access : Write access to external memory

■ Displaying All Machine Cycles (Specify /CYCLE.)

Detailed information at all sampled machine cycles can be displayed. In this mode, both single trace and multitrace data can be displayed in almost identical formats. (In the multitrace mode, the local frame number and block number are added.)

In this mode, data can be displayed in the following format. For further details, see the descriptions of the SHOW TRACE, and SHOW MULTITRACE commands. In this mode, source is not displayed regardless of the setup made using the SET SOURCE command.

[Example]

```
 >>SHOW TRACE/CYCLE -587
frame no.   address   data   a-status   d-status   Qst   dfg   level   ext-probe
-00587      :FF0106   0106   ---        -------    FLH         4       11111111
-00586      :FF0106   0008   ECF        EXECUTE    ---   @     4       11111111
-00585      :FF0106   0106   ---        EXECUTE    ---         5       11111111
-00584      :1010E8   10E8   ---        -------    ---         5       11111111
-00583      :1010E8   0102   EWA        EXECUTE    --    @     5       11111111
-00582      :1010E8   0102   ---        EXECUTE    ---         5       11111111
```

```
-00581    :000186   0186   ---    -------    2by        5  11111111
-00580    :000186   10F2   IRA    EXECUTE    ---   @    5  11111111
-00579    :1010E6   10E6   ---    -------    ---        5  11111111
-00578    :1010E6   10F2   EWA    EXECUTE    ---   @    5  11111111
-00577    :1010E6   10F2   ---    EXECUTE    ---        5  11111111
-00576    :000186   0186   ---    -------    ---        5  11111111
```

How to read trace data

```
 frame no.  address  data   a-status  d-status   Qst   dfg   level   ext-probe
    (1)       (2)     (3)     (4)        (5)      (6)   (7)    (8)       (9)
```

(1):frame number (Decimal, number)

(2):executed instruction address, and data access address (Hexadecimal number)

(3):data (Hexadecimal number)

(4):access information (a-status)

IWA	:	write access to internal memory
EWA	:	write access to external memory
IRA	:	read access to internal memory
ERA	:	read access to external memory
ICF	:	code fetch to internal memory
ECF	:	code fetch to external memory
---	:	valid "d-status" information

(5):device information (d-status)

STANDBY	:	hardware standby
THOLD	:	tool hold
UHOLD	:	user hold
WAIT	:	ready pin
SLEEP	:	sleep
STOP	:	stop
EXECUTE	:	execute instruction
RESET	:	reset
-------	:	invalid d-status information

(6):instruction queue status

FLH:flush queue

-by:number of remainder code of queue is -byte(-:1 to 8)

(7):valid flag

@:valid frame for this data

(8):sequencer level

(9):external probe data

■ Display in Source Line Units (Specify /SOURCE.)

Only the source line can be displayed. This mode is enabled only in the single trace mode while in the debugging mode.

[Example]

```
 >>SHOW TRACE/SOURCE -194
step no.    source
-00194:     gtg1.c$251 {
-00190:     gtg1.c$255          sub5(nf, nd);
-00168:     gtg1.c$259 {
-00164:     gtg1.c$264          p = (char *)  &df;
-00161:     gtg1.c$264          p = (char *)  &df;
-00157:     gtg1.c$265          *(p++) = 0x00;
-00145:     gtg1.c$266          *(p++) = 0x00;
-00133:     gtg1.c$267          *(p++) = 0x80;
-00121:     gtg1.c$268          *p     = 0x7f;
-00116:     gtg1.c$270          p = (char *)  &dd;
-00111:     gtg1.c$271          *(p++) = 0xff;
-00099:     gtg1.c$272          *(p++) = 0xff;
```

2.2.7.8 Reading Trace Data On-the-fly

Trace data can be read while executing a program. However, this is not possible during sampling. Disable the trace function or terminate tracing before attempting to read trace data.

■ Reading Trace Data On-the-fly in Single Trace

To disable the trace function, use the DISABLE TRACE command. Check whether or not the trace function is currently enabled by executing the SHOW TRACE command with /STATUS specified, or by using the built-in variable, %TRCSTAT.

Tracing terminates when the delay count ends after the sequencer has terminated. If Not Break is specified here, tracing terminates without a break operation. It is possible to check whether or not tracing has terminated by executing the SHOW TRACE command with /STATUS specified, or by using the built-in variable, %TRCSAMP.

To read trace data, use the SHOW TRACE command; to search trace data, use the SEARCH TRACE command. Use the SET DELAY command to set the delay count and break operation after the delay count.

[Example]

```
 >GO
 >>SHOW TRACE/STATUS
en/dis       = enable
buffer full  = nobreak
sampling     = on          <- Trace sampling continues.
 >>SHOW TRACE/STATUS
en/dis       = enable
buffer ful   = nobreak
sampling     = end         <- Trace sampling ends.
frame no.    = -00805 to  00000
step no.     = -00262 to  00000
 >>SHOW TRACE -52
step no.     address      mnemonic                                 level
\sub5:
-00052       : FF0125     LINK                     #02             1
-00051       : 000186     internal read access.    10E6            1
-00050       : 1010D6     external write access.   10E6            1
-00049       : 000186     internal write access.   10D6            1
.            .            .
```

If the CLEAR TRACE command is executed with the trace ending state, trace data sampling can be re-executed by re-executing the sequencer from the beginning.

■ Reading Trace Data On-the-fly in the Multi Trace

Use the DISABLE MULTITRACE command to disable the trace function before reading trace data. Check whether or not the trace function is currently enabled by executing the SHOW MULTITRACE command with /STATUS specified, or by using the built-in variable %TRCSTAT.

To read trace data, use the SHOW MULTITRACE command; to search trace data, use the SEARCH MULTITRACE command.

[Example]

```
 >GO
>>SHOW MULTITRACE/STATUS
en/dis          = enable
buffer full     = nobreak
sampling        = on
 >>DISABLE MULTITRACE
 >>SHOW MULTITRACE/STATUS
en/dis          = disable
buffer full     = nobreak
sampling        = end
block no.       = 1 to 20
frame no.       = 00001 to 00639
 >>SHOW MULTITRACE  1
frame no.       address         data    a-status   d-status      Qst     dfg    level   ext-probe
block no. = 1
00001           -7 : 10109C     109C    ---        -------       ---            1       11111111
00002           -6 : 10109C     0000    EWA        EXECUTE        2by    @      1       11111111
00003           -5 : 10109C     0000    ---        EXECUTE       ---            1       11111111
00004           -4 : FF0120     0120    ---        -------       ---            1       11111111
                        .                          .                            .
                        .                          .                            .
```

2.2.7.9 Saving Trace Data

The debugger has function of saving trace data to a file.

■ Saving Trace Data

Save the trace data to the specified file.

For details on operations, refer to Sections 3.14 Trace Window, and 4.4.8 Trace in the SOFTUNE Workbench Operation Manual; and Section 4.23 SHOW TRACE in the SOFTUNE Workbench Command Reference Manual.

2.2.8 Measuring Performance

It is possible to measure the time and pass count between two events. Repetitive measurement can be performed while executing a program in real-time, and when done, the data can be totaled and displayed.
Using this function enables the performance of a program to be measured. To measure performance, set the event mode to the performance mode using the SET MODE command.

■ Performance Measurement Function

The performance measurement function allows the time between two event occurrences to be measured and the number of event occurrences to be counted. Up to 32767 event occurrences can be measured.

- **Measuring Time**

  Measures time interval between two events.

  Events can be set at 8 points (1 to 8). However, in the performance measurement mode, the intervals, starting event number and ending event number are combined as follows. Four intervals have the following fixed event number combination:

Interval	Starting Event Number	Ending Event Number
1	1	2
2	3	4
3	5	6
4	7	8

- **Measuring Count**

  The specified events become performance measurement points automatically, and occurrences of that particular event are counted.

2.2.8.1 Performance Measurement Procedures

Performance can be measured by the following procedure:

- **Set event mode.**
- **Set minimum measurement unit for timer.**
- **Specify performance-buffer-full break.**
- **Set events.**
- **Execute program.**
- **Display measurement result.**
- **Clear measurement result.**

■ Setting Event Mode

Set the event mode to the performance mode using the SET MODE command. This enables the performance measurement function.

[Example]

```
>SET MODE/PERFORMANCE
>
```

■ Setting Minimum Measurement Unit for Timer

Using the SET TIMESCALE command, choose either 1 µs or 100 ns as the minimum measurement unit for the timer used to measure performance. The default is 1 µs.

When the minimum measurement unit is changed, the performance measurement values are cleared.

[Example]

```
>SET TIMERSCALE/1U           <- Set 1 µs as minimum unit.
>
```

■ Setting Performance-Buffer-Full Break

When the buffer for storing performance measurement data becomes full, a executing program can be broken. This function is called the performance-buffer-full break. The performance buffer becomes full when an event occurs 32767 times.

If the performance-buffer-full break is not specified, the performance measurement ends, but the program does not break.

[Example]

```
>SET PERFORMANCE/NOBREAK          <- Specifying Not Break
>
```

■ Setting Events

Set events using the SET EVENT command.

The starting/ending point of time measurement and points to measure pass count are specified by events.

Events at 8 points (1 to 8) can be set. However, in the performance measurement, the intervals, starting event number and ending event number are fixed in the following combination.

- **Measuring Time**

  Four intervals have the following fixed event number combination.

Interval	Starting Event Number	Ending Event Number
1	1	2
2	3	4
3	5	6
4	7	8

- **Measuring Count**

  The specified events become performance measurement points automatically.

■ Executing Program

Start measuring when executing a program by using the GO or CALL command. If a break occurs during interval time measurement, the data for this specific interval is discarded.

■ Displaying Performance Measurement Data

Display performance measurement data by using the SHOW PERFORMANCE command.

■ Clearing Performance Measurement Data

Clear performance measurement data by using the CLEAR PERFORMANCE command.

[Example]

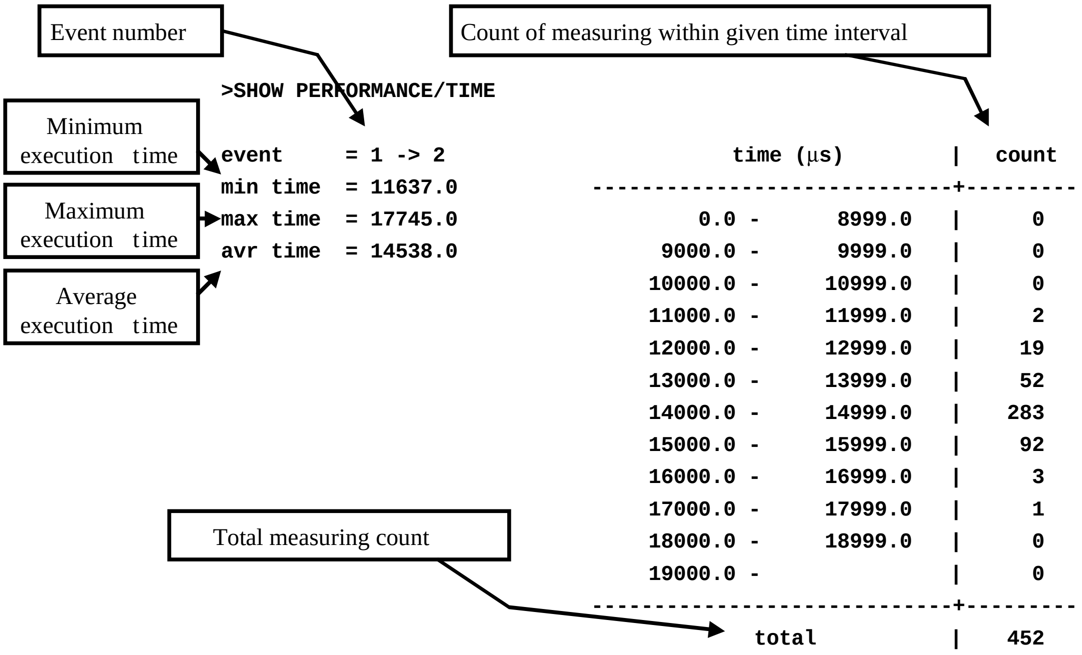
```
>CLEAR PERFORMANCE
>
```

2.2.8.2 Display Performance Measurement Data

Display the measured time and measuring count by using the SHOW PERFORMANCE command.

■ Displaying Measured Time

To display the time measured, specify the starting event number or the ending event number.



Event number

Count of measuring within given time interval

Minimum execution time

Maximum execution time

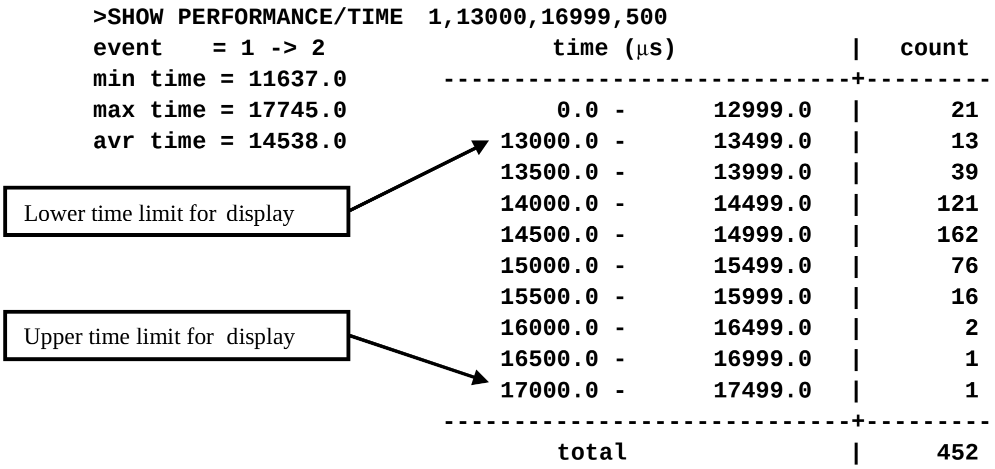
Average execution time

Total measuring count

```
>SHOW PERFORMANCE/TIME

event     = 1 -> 2                    time (µs)        |   count
min time  = 11637.0        ----------------------------+---------
max time  = 17745.0              0.0 -      8999.0     |     0
avr time  = 14538.0           9000.0 -      9999.0     |     0
                             10000.0 -     10999.0     |     0
                             11000.0 -     11999.0     |     2
                             12000.0 -     12999.0     |    19
                             13000.0 -     13999.0     |    52
                             14000.0 -     14999.0     |   283
                             15000.0 -     15999.0     |    92
                             16000.0 -     16999.0     |     3
                             17000.0 -     17999.0     |     1
                             18000.0 -     18999.0     |     0
                             19000.0 -                 |     0
                           ----------------------------+---------
                                    total              |   452
```

The lower time limit, upper time limit and display interval can be specified. The specified time value is in 1µs, when the minimum measurement unit timer is set to 1 us by the SET TIMESCALE command, and in 100 ns when the minimum is set to 100 ns.



Lower time limit for display

Upper time limit for display

```
>SHOW PERFORMANCE/TIME  1,13000,16999,500
event    = 1 -> 2                time (µs)           |  count
min time = 11637.0      ----------------------------+---------
max time = 17745.0            0.0 -      12999.0    |      21
avr time = 14538.0        13000.0 -      13499.0    |      13
                          13500.0 -      13999.0    |      39
                          14000.0 -      14499.0    |     121
                          14500.0 -      14999.0    |     162
                          15000.0 -      15499.0    |      76
                          15500.0 -      15999.0    |      16
                          16000.0 -      16499.0    |       2
                          16500.0 -      16999.0    |       1
                          17000.0 -      17499.0    |       1
                        ----------------------------+---------
                                total               |     452
```

2.2.9 Measuring Coverage

This emulator has the C0 coverage measurement function. Use this function to find what percentage of an entire program has been executed.

■ Coverage Measurement Function

When testing a program, the program is executed with various test data input and the results are checked for correctness. When the test is finished, every part of the entire program should have been executed. If any part has not been executed, there is a possibility that the test is insufficient.

This emulator coverage function is used to find what percentage of the whole program has been executed. In addition, details such as which addresses were not accessed can be checked.

This enables the measurement coverage range to be set and the access attributes to be measured.

To execute the C0 coverage, set a range within the code area and set the attribute to Code attribute. In addition, specifying the Read/Write attribute and setting a range in the data area, permits checking the access status of variables such as finding unused variables, etc.

Execution of coverage measurement is limited to the address space specified as the debug area.

Therefore, set the debug area in advance. However, the measurement attribute for coverage measurement can be specified regardless of attributes of the debug area.

■ Coverage Measurement Procedures

The procedure for coverage measurement is as follows:

- Set range for coverage measurement:	SET COVERAGE
- Measuring coverage:	GO, STEP, CALL
- Displaying measurement result:	SHOW COVERAGE

■ Coverage Measurement Operation

The following operation can be made in coverage measurement:

- Load/Save of coverage data:	LOAD/COVERAGE, SAVE/COVERAGE
- Abortion and resume of coverage measurement:	ENABLE COVERAGE, DISABLE COVERAGE
- Clearing coverage data:	CLEAR COVERAGE
- Canceling coverage measurement range:	CANCEL COVERAGE

Reference:

With MB2141 emulator, the code coverage is affected by a prefetch by the MCU. Note the prefetch when using the COVERAGE function.

2.2.9.1 Coverage Measurement Procedures

The procedure for coverage measurement is as follows:

- Set range for coverage measurement	**:**	**SET COVERAGE**
- Measure coverage	**:**	**GO, STEP, CALL**
- Display measurement result	**:**	**SHOW COVERAGE**

■ Setting Range for Coverage Measurement

Use the SET COVERAGE command to set the measurement range. The measurement range can be set only within the area defined as the debug area. Up to 32 ranges can be specified.

In addition, the access attribute for measurement can be specified. This attribute can be specified regardless of the attributes of the debug area.

By specifying /AUTOMATIC for the command qualifier, the code area for the loaded module is set automatically. However, the library code area is not set when the C compiler library is linked.

[Example]

```
>SET COVERAGE FF0000 .. FFFFFF
```

■ Measuring Coverage

When preparing for coverage measurement, execute the program.

Measurement starts when the program is executed by using the GO, STEP, or CALL command.

■ Displaying Coverage Measurement Result

To display the coverage measurement result, use the SHOW COVERAGE command. The following can be displayed:

- Display coverage rate of total measurement area
- Displaying coverage rate of load module
- Summary of 16 addresses as one block
- Details indicating access status of each address
- Displaying coverage measurement result per source line
- Displaying coverage measurement result per machine instruction

● Displaying coverage rate of total measurement area (specify /TOTAL for the command qualifier)

```
 >SHOW COVERAGE/TOTAL
total coverage : 82.3%
```

● Displaying coverage rate of load module (specify /MODULE for the command qualifier)

```
>SHOW COVERAGE/MODULE
sample.abs  . . . . . . . . . . . . . . (84.03%)
 +-  startup.asm . . . . . . . . . . . (90.43%)
 +-  sample.c  . . . . . . . . . . . . . (95.17%)
 +-  samp.c  . . . . . . . . . . . . . .  (100.00%)
```

Displays the load modules and the coverage rate of each module.

● Summary (Specify /GENERAL for command qualifier)

```
>SHOW COVERAGE/GENERAL
  (HEX)0X0        +1X0         +2X0
       +--------------+--------------+------        ------
address 0123456789ABCDEF0123456789ABCDEF0123456    ... ABCDEF   C0(%)
FF0000  **3*F*.......                                           32.0
```

Display the access status of every 16 addresses
.   : No access
1 to F : Display the number accessed in 16 addresses by the hexadecimal number.
*   : Access all of the 16 addresses.

● Details (Specify /DETAIL for command qualifier)

Display one line of a coverage rate

```
>SHOW COVERAGE/DETAIL FF0000

address +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +A +B +C +D +E +F C0(%)
FF0000   -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  - 100.0
FF0010   -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  - 100.0
FF0020   .  .  .  .  -  -  -  .  .  .  .  .  .  .  .  .  18.6
FF0030   -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  - 100.0
FF0040   -  .  -  -  -  -  -  -  -  -  -  -  -  -  -  -  93.7
FF0050   -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  - 100.0
FF0060   .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .   0.0
FF0070   .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .   0.0
FF0080   .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .   0.0
```

Display the access status of every 1 address
.   : No access
-     : Access

- ● Displays per source line (specify /SOURCE for the command qualifier)

```
>SHOW COVERAGE/SOURCE main
*     70: {
      71:         int    i;
      72:         struct table *value[16];
      73:
*     74:         for (i=0; i<16; i++)
*     75:                 value[i] = &target[i];
      76:
*     77:         sort_val(value, 16L);
.     78: }
```

Displays access status of each source line.

. : No Access

* : Accessed

Blank : Line which the code had not been generated or is outside the scope of the coverage measurement

- ● Displays per machine instruction (specify /INSTRUCTION for the command qualifier)

```
>SHOW COVERAGE/INSTRUCTION F9028F
sample.c$70 {
 * F9028F               ¥main:
 * F9028F 0822               LINK      #22
 * F90291 4F01               PUSHW     RW0
sample.c$74       for (i=0; i<16; i++)
 . F90293 D0                 MOVN      A,#0
 . F90294 CBFE               MOVW      @RW3-02,A
 . F90296 BBFE               MOVW      A,@RW3-02
 . F90298 3B1000             CMPW      A,#0010
 . F9029B FB18               BGE       F902B5
sample.c$75              value[i] = &target[i];
 . F9029D BBFE               MOVW      A,@RW3-02
 . F9029F 0C                 LSLW      A
 . F902A0 98                 MOVW      RW0,A
 . F902A1 71F3DE             MOVEA     A,@RW3-22
 . F902A4 7700               ADDW      RW0,A
 . F902A6 4214               MOV       A,#14
 . F902A8 7833FE             MULUW     A,@RW3-02
 . F902AB 38A001             ADDW      A,#01A0
```

Displays access status of each source line.

. : No Access

* : Accessed

Blank : Instruction outside the scope of the coverage measurement

---

Note:

With MB2141 emulator, the code coverage measurement is affected by a prefetch. Note when analyzing.

2.2.10 Measuring Execution Time Using Emulation Timer

The timer for measuring time is called the emulation timer. This timer can measure the time from the start of MCU operation until suspension.

■ Measuring Executing Time Using Emulation Timer

Choose either 1 µs or 100 ns as the minimum measurement unit for the emulation timer and set the measurement unit using the SET TIMESCALE command.

When 1 µs is selected as the minimum unit, the maximum is about 70 minutes; when 100 ns is selected as the minimum unit, the maximum is about 7 minutes.

The default is 1 µs.

By using this timer, the time from the start of MCU operation until the suspension can be measured.

The measurement result is displayed as two time values: the execution time of the preceding program, and the total execution time of programs executed so far plus the execution time of the preceding program.

If the timer overflows during measurement, a warning message is displayed. Measurement is performed every time a program is executed.

The emulation timer cannot be disabled but the timer value can be cleared instead.

Use the following commands to control the emulation timer.

SHOW TIMER: Displays measured time

CLEAR TIMER: Clear measured timer

[Example]

```
 >GO  main,$25
Break at FF008D by breakpoint
 >SHOW TIMER
                               min   s    ms    µs    ns
from init            =         00:  42: 108: 264: 000
from last executed   =         00:  03: 623: 874: 000
 >CLEAR TIMER
 >SHOW TIMER
                               min   s    ms    µs    ns
from init            =          00: 00: 000: 000: 000
from last executed   =          00: 00: 000: 000: 000
 >
```

Note:

Note that the measured execution time is added about ten extra cycles per execution.

2.2.11 Sampling by External Probe

An external probe can be used to sample (input) data. There are two sampling types: sampling the trace buffer as trace data, and sampling using the SHOW SAMPLING command.

■ Sampling by External Probe

There are two sampling types to sample data using an external probe: sampling the trace buffer as trace data, and sampling using the SHOW SAMPLING command.

When data is sampled as trace data, such data can be displayed by using the SHOW TRACE command or SHOW MULTITRACE command, just as with other trace data. Sampling using the SHOW SAMPLING command, samples data and displays its state.

In addition, by specifying external probe data as events, such events can be used for aborting a program, and as multitrace and performance trigger points.

Events can be set by using the SET EVENT command.

■ External Probe Sampling Timing

Choose one of the following for the sampling timing while executing a program.

- At rising edge of internal clock (clock supplied by emulator)
- At rising edge of external clock (clock input from target)
- At falling edge of external clock (clock input from target)

Use the SET SAMPLING command to set up; to display the setup status use the SHOW SAMPLING command.

When sampling data using the SHOW SAMPLING command, sampling is performed when the command is executed and has nothing to do with the above settings.

[Example]

```
>>SET SAMPLING/INTERNAL
>>SHOW SAMPLING
sampling timing : internal
channel     7 6 5 4 3 2 1 0
            1 1 1 1 0 1 1 1
```

■ Displaying and Setting External Probe Data

When a command that can use external probe data is executed, external probe data is displayed in 8-digit binary or 2-digit hexadecimal format. The displayed bit order is in the order of the IC clip cable color code order (Table 2.2-12). The MSB is at bit 7 (Violet), and the LSB is at bit 0 (Black). The bit represented by 1 means HIGH, while the bit represented by 0 means LOW. When data is input as command parameters, these values are also used for input.

Table 2.2-12 Bit Order of External Probe Data

IC Clip Cable Color	Violet	Blue	Green	Yellow	Orange	Red	Brown	Black
Bit Order	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
	External probe data							

■ Commands for External Probe Data

Table 2.2-13 shows the commands that can be used to set or display external probe data.

Table 2.2-13 Commands that can be used External Probe Data

Usable Command	Function
SET SAMPLING SHOW SAMPLING	Sets sampling timing for external probe Samples external probe data
SET EVENT SHOW EVENT	Enables to specify external probe data as condition for event Displays event setup status
SHOW TRACE	Displays external probe trace-sampled (single trace)
SHOW MULTITRACE	Displays external probe trace-sampled (multi-trace)

2.3 Emulator Debugger (MB2147-01)

This section explains the functions of the emulator debuggers for the MB2147-01.

■ Emulator

When choosing the emulator debugger from the setup wizard, select one of the following emulators. Select the MB2147-01.

MB2141

MB2147-01

MB2147-05

The emulator debugger for the MB2147-01 is software that controls an emulator from a host computer via a communications line (RS-232C, LAN or USB) to evaluate programs.

The following series can be debugged:

F^{2}MC-16L

F^{2}MC-16LX

Before using the emulator, the emulator must be initialized. For details, refer to Appendix B, Monitoring Program Download, and Appendix C, LAN Interface Setup, in the SOFTUNE WORKBENCH Operation Manual.

For further details, refer to the Operation Manual Appendix B Download Monitor Program, and Appendix C Setting up LAN Interface.

2.3.1 Setting Operating Environment

This section explains the operating environment setup.

■ Setting Operating Environment

For the emulator debugger for the MB2147-01, it is necessary to set the following operating environment. Predefined default settings for all these setup items are enabled at startup. Therefore, setup is not required when using the default settings. Adjusted settings can be used as new default settings from the next time.

- Monitoring program automatic loading
- MCU operation mode
- Debug area
- Memory mapping
- Debug function
- Event mode

2.3.1.1 Monitoring Program Automatic Loading

The MB2147-01 emulator can automatically update the monitoring program at emulator startup.

■ Setting Monitoring Program Automatic Loading

When the MB2147-01 emulator is specified, data in the emulator can be checked at the beginning of debugging to load an appropriate monitoring program and configuration binary data automatically into the emulator.

The monitoring program and configuration binary data to be compared for update are in Lib\907 under the directory where Workbench is installed.

Enable/disable the monitoring program automatic loading function by choosing [Environment] - [Debug Environment Setup] - [Setup Wizard] menu.

2.3.1.2 MCU Operation Mode

There are two MCU operation modes as follows:

- **Debugging Mode**
- **Native Mode**

■ Setting MCU Operation Mode

Set the MCU operation mode.

There are two operation modes: the debugging mode, and the native mode.

Choose either one using the SET RUNMODE command.

At emulator start-up, the MCU is in the debugging mode.

The data access to internal bus may not be detected by emulator in native mode. Therefore, when the MCU operation mode is changed, all the following are initialized:

- Data break points
- Data monitoring break
- Event condition settings
- Sequencer settings
- Trace measurement settings and trace buffer

■ Debugging Mode

All the operations of evaluation chips can be analyzed, but their operating speed is slower than that of mass-produced chips.

■ Native Mode

Evaluation chips have the same timing as mass-produced chips to control the operating speed. Note that the restrictions the shown in Table 2.3-1 are imposed on the debug functions.

Table 2.3-1 Restrictions on debug functions

Applicable series	Restrictions on debug functions
Common to all series	- When a data read access occurs on the MCU internal bus, the internal bus access information is not sampled and stored in the trace buffer. - Even when a data break or event (data access condition) is set for data on the MCU internal bus, it may not become a break factor or sequencer-triggering factor. - The coverage function may fail to detect an access to data on the MCU internal bus.

2.3.1.3 Debug Area

Set the intensive debugging area out of the whole memory space. The area functions are enhanced.

■ Setting Debug Area

There are two debug areas: DEBUG3, and DEBUG4. A continuous 1 MB area (16 banks) is set for each area.

Set the debug area using the SET DEBUG command.

Setting the debug area enhances the coverage measurement function.

- **Enhancement of Coverage Measurement Function**

Setting the debug area enables the coverage measurement function. In coverage measurement, the measurement range can be specified only within the area specified as the debug area. In 0x00 to 0x0F and 0x0F0 to 0x0FF, a break point can be set without specifying the debug area. (DEBUGGER1, DEBUGGER2)

2.3.1.4 Memory Area Types

A unit in which memory is allocated is called an area. There are five different area types.

■ Memory Area Types

A unit to allocate memory is allocated is called an area. There are five different area types as follows:

- **User Memory Area**

  Memory space in the user system is called the user memory area and this memory is called the user memory. Up to four user memory areas can be set with no limit on the size of each area. Define a region on a 256-byte boundary.

  Access attributes can be set for each area; for example, CODE, READ, etc., can be set for ROM area, and READ, WRITE, etc. can be set for RAM area. If the MCU attempts access in violation of these attributes, the MCU operation is suspended and an error is displayed (guarded access break).

  Memory manipulation commands can be executed in relation to emulation memory areas while MCU execution is in progress.

  To set the user memory area, use the SET MAP command.

- **Emulation Memory Area**

  Memory space substituted for emulator memory is called the emulation memory area, and this memory is called emulation memory.

  It is possible to set up to four areas of 1 MB maximum (including an internal ROM area described later) as emulation memory area. Define a region on a 256-byte boundary. An area larger than 1 MB can be specified at one time but is divided internally into two or more 1 MB areas for management purposes.

  Memory manipulation commands can be executed in relation to emulation memory areas while MCU execution is in progress.

  Emulation memory areas can be set using the SET MAP command.

  Further, the access attributes can be set as with user memory areas.

Note:

Even if the MCU internal resources are set as emulation memory area, access is made to the internal resources. Re-executing this setup may damage data. The F^{2}MC-16/16H only allows this setup in the debugging mode.

- **Internal ROM Area**

  The area where the emulator internal memory is substituted for internal ROM is called the internal ROM area, and this memory is called the internal ROM memory.

  The internal ROM area with a size up to 1 MB can be specified two areas.

  An area larger than 1 MB can be specified at one time but is divided internally into two or more 1 MB areas for management purposes.

  Memory manipulation commands can be executed in relation to emulation memory areas while MCU execution is in progress.

The internal ROM area is capable to set by the "Setup Map" dialog opening by "Debugger Memory Map" from "Setup".

Note:

The internal memory area, it is set a suitable area automatically by the selected MCU.

- **Internal ROM Image Area**

  Some types of MCUs have data in a specific area of internal ROM appearing to 00 bank. This specific area is called the internal ROM image area.

  The internal ROM image area is capable to set by the "Setup Map" dialog opening by "Debugger Memory Map" from "Setup". This area attribute is automatically set to READ/CODE. The same data as in the internal ROM area appears in the internal ROM image area.

  Note that the debug information is only enabled for either one (one specified when linked). To debug only the internal ROM image area, change the creation type of the load module file.

Note:

The internal memory area, it is set a suitable area automatically by the selected MCU.

- **Undefined Area**

  A memory area that does not belong to any of the areas described above is part of the user memory area. This area is specifically called the undefined area.

  The undefined area can be set to either NOGUARD area, which can be accessed freely, or GUARD area, which cannot be accessed. Select either setup for the whole undefined area. If the area attribute is set to GUARD, a guarded access error occurs if access to this area is attempted.

2.3.1.5 Memory Mapping

Memory space can be allocated to the user memory and the emulation memory, etc., and the attributes of these areas can be specified.
However, the MCU internal resources are not dependent on this mapping setup and access is always made to the internal resources.

■ Access Attributes for Memory Areas

The access attributes shown in Table 2.3-2 can be specified for memory areas.

A guarded access break occurs if access is attempted in violation of these attributes while executing a program.

When access to the user memory area and the emulation memory area is made using program commands, such access is allowed regardless of the CODE, READ, WRITE attributes. However, access to memory with the GUARD attribute in the undefined area, causes an error.

Table 2.3-2 Types of Access Attributes

Area	Attribute	Description
User Memory Emulation Memory	CODE	Instruction Execution Enabled
	READ	Data Read Enabled
	WRITE	Data Write Enabled
Undefined	GUARD	Access Disabled
	NOGUARD	No check of access attribute

When access is made to an area without the WRITE attribute by executing a program, a guarded access break occurs after the data has been rewritten if the access target is the user memory. However, if the access target is the emulation memory, the break occurs before rewriting. In other words, write-protection (memory data cannot be overwritten by writing) can be set for the emulation memory area by not specifying the WRITE attribute for the area.

This write-protection is only enabled for access made by executing a program, and is not applicable to access by commands.

■ Creating and Viewing Memory Map

Use the following commands for memory mapping.

SET MAP:	Set memory map.
SHOW MAP:	Display memory map.
CANCEL MAP:	Change memory map setting to undefined.

[Example]

```
 >SHOW MAP
address                    attribute          type
000000 .. FFFFFF           noguard
The rest of setting area numbers
user = 8      emulation = 5
 >SET MAP/USER H'0..H'1FF
 >SET MAP/READ/CODE/EMULATION H'FF0000..H'FFFFFF
 >SET MAP/USER H'8000..H'8FFF
 >SET MAP/MIRROR/COPY H'8000..H'8FFF
 >SET MAP/GUARD
 >SHOW MAP
address                    attribute          type
000000 .. 0001FF           read write         user
000200 .. 007FFF           guard
008000 .. 008FFF           read write         user
009000 .. FEFFFF           guard
FF0000 .. FFFFFF           read write code    emulation
mirror address area
008000 .. 008FFF           copy
The rest of setting area numbers
user = 6      emulation = 3
 >
```

■ Internal ROM Area Setting

The [Map Setting] dialog box is displayed using [Environment] - [Debugger Memory Map]. You can set the internal ROM area using the [Internal ROM Area] tab after the [Map Adding] dialog box is displayed by clicking on the [Setting] button. You can set two areas. Both require empty Emulation area to be set. You can set the region size by (Empty space of the emulation area) x (one area size).

Specify the internal ROM area from the ending address H'FFFFFF (fixed) for area 1. Also, it is possible to delete the internal ROM area.

2.3.1.6 Debug Function

The debug function has the following two types. Only the function of the selected mode can be used. The selectable debug mode depends on the emulator or its connection form.

- **RAM Checker mode**
- **Trace Enhancement mode**

■ Setting of Debug Function

Set the debug function. The debug function has the RAM Checker and the Trace Enhancement mode. The selectable mode depends on the emulator or its connection form. These modes can be set by using [Setup] - [Debug Environment] - [Debug Function] or the SET MODE command on the command window.

At the emulator activated, this is set to the RAM Checker mode.

When the debug function is changed, all the followings are initialized:

- Performance measurement data
- Trace buffer

■ RAM Checker mode

Enables the RAM Checker function. The history of accessing the monitoring addresses can be recorded into the log file.

■ Trace Enhancement mode

Enable the trace enhancement.

The following functions become available.

1.Trace acquisition in the multitrace mode

2.Trace acquisition control by trace trigger (resumption/pausing/termination)

3.Trace control by data monitoring condition

4.Trace control by sequencer

2.3.1.7 Event Mode

There are three event modes as listed below.

- **Normal mode**
- **Multi trace mode**
- **Performance mode**

■ Event Mode

Event mode is used to determine which function the event triggers are used for. To set the mode, use [Event] tab on [Setup] - [Debug Environment] - [Debug Function] or the SET MODE command on the command window. The default is normal mode.

There are three event modes as listed below.

- Normal mode
  Event triggers are used for the single trace.
- Multi trace mode
  Event triggers are used for the multitrace (trace function which samples data before and after the event trigger occurred).
- Performance mode
  Event triggers are used for the performance measurement. It enables to measure time duration between two event trigger occurrence and count of event trigger occurrence.

Note:

The multi trace mode can be specified only when the debug function on MB2147-01 is set to Trace Enhancement mode. For more details, see "2.3.1.6 Debug Function".

2.3.2 Notes on Commands for Executing Program

When using commands to execute a program, there are several points to note.

■ Notes on GO Command

For the GO command, two break points that are valid only while executing commands can be set. However, care is required in setting these break points.

- **Invalid Breakpoints**
  - No break occurs when a break point is set at the instruction immediately after the following instructions.

F^2MC-16L/16LX	PCB NCC SPB MOV ILM,#imm8 OR CCR,#imm8	DTB ADB CNR AND CCR,#imm8 POPW PS

  - No break occurs when break point set at address other than starting address of instruction.
  - No break occurs when both following conditions met at one time.
    - Instruction for which break point set starts from odd-address,
    - Preceding instruction longer than 2 bytes length, and break point already set at last 1-byte address of preceding instruction (This "already-set" break point is an invalid break point that won't break, because it has been set at an address other than the starting address of an instruction).
- **Abnormal Break Point**

  Setting a break point at the instruction immediately after string instructions listed below, may cause a break in the middle of the string instruction without executing the instruction to the end.

F^2MC-16L/16LX	MOVS SECQ WBTS MOVSWI SECQWI MOVSD SECQD FILS FILSW	MOVSW SECQW MOVSI SECQI WBTC MOVSWD SECQWD FILSI FILSWI

■ Notes on STEP Command

- **Exceptional Step Execution**

  When executing the instructions listed in the notes on the GO command as invalid break points and abnormal break points, such instructions and the next instruction are executed as a single instruction. Furthermore, if such instructions are continuous, then all these continuous instructions and the next instruction are executed as a single instruction.

- **Step Execution that won't Break**

  Note that no break occurs after step operation when both the following conditions are met at one time.

  - When step instruction longer than 2 bytes and last code ends at even address
  - When break point already set at last address (This "already-set" break point is an invalid break point that won't break, because it has been set at an address other than the starting address of an instruction.)

■ Controlling Watchdog Timer

It is possible to select "No reset generated by watchdog timer counter overflow" while executing a program using the GO, STEP, CALL commands.

Use the ENABLE WATCHDOG, DISABLE WATCHDOG commands to control the watchdog timer.

- ENABLE WATCHDOG:	Reset generated by watchdog timer counter overflow
- DISABLE WATCHDOG:	No reset generated by watchdog timer counter overflow

The start-up default in this program is "Reset generated by watchdog timer counter overflow".

[Example]

```
>DISABLE WATCHDOG
>GO
```

2.3.3 On-the-fly Executable Commands

Certain commands can be executed even while executing a program. This is called "on-the-fly" execution.

■ On-the-fly Executable Commands

Certain commands can be executed on-the-fly. If an attempt is made to execute a command that cannot be executed on-the-fly, an error "MCU is busy" occurs. Table 2.3-3 lists major on-the-fly executable functions. For further details, refer to the SOFTUNE WORKBENCH Command Reference Manual.

Meanwhile, on-the-fly execution is enabled only when executing the MCU from the menu or the tool button. On-the-fly commands cannot be executed when executing the GO command, etc., from the Command window.

Using the real-time monitoring function, it is also possible to display a specified memory region in the real-time monitoring window and read (update) data even during an MCU execution.

Table 2.3-3 Major Functions Executable in On-the-fly Mode (1 / 2)

Function	Restrictions	Major Commands
MCU reset	-	RESET
Displaying MCU execution status	-	SHOW STATUS
Displaying trace data	1. Enabled only when trace execution ended *1 2. Enabled only when the debug function is in "Trace Enhancement" mode.*2 (only MULTITRACE)	SHOW TRACE, SHOW MULTITRACE
Clear trace data	1. Enabled only when trace execution ended *1 2. Enabled only when the debug function is in "Trace Enhancement" mode.*2 (only MULTITRACE)	CLEAR TRACE, CLEAR MULTITRACE
Search trace data	1. Enabled only when trace execution ended *1 2. Enabled only when the debug function is in "Trace Enhancement" mode.*2 (only MULTITRACE)	SEARCH TRACE, SEARCH MULTITRACE
Set trace acquisition data	Enabled only when trace execution ended *1	ENABLE TRACE, DISABLE TRACE
Set trace trigger	1. Enabled only when trace execution ended *1 2. Enabled only when the debug function is in "Trace Enhancement" mode.*2	SET TRACETRIGGER, SHOW TRACETRIGGER, CANCEL TRACETRIGGER

Table 2.3-3 Major Functions Executable in On-the-fly Mode (2 / 2)

Function	Restrictions	Major Commands
Set filtering area	1. Enabled only when trace execution ended *1 2. Enabled only when the debug function is in "Trace Enhancement" mode.*2	SET DATATRACEAREA, SHOW DATATRACEAREA, CANCEL DATATRACEAREA
Set trace delay	1. Enabled only when trace execution ended *1 2. Enabled only when the debug function is in "Trace Enhancement" mode.*2	SET DELAY, SHOW DELAY
Enable/Disable trace	-	ENABLE TRACE, DISABLE TRACE
Displaying execution cycle measurement value (Timer)	-	SHOW TIMER
Memory operation (Read/Write)	Emulation memory only operable Read only enabled in real-time monitoring area	ENTER, EXAMINE , DUMP, SEARCH MEMORY, SHOW MEMORY, SET MEMORY
Line assembly, Disassembly	Emulation memory only enabled Real-time monitoring area, Disassembly only enabled	ASSEMBLE DISASSEMBLE
Load, Save program	Emulation memory only enabled Real-time monitoring area, Save only enabled	LOAD SAVE
Break Point Settings	This is possible only when it is enabled by setting a break point while executing in the execution tabs [Environment] - [Debugging Environment Setting] - [Debugging Environment] menus.	SET BREAK, ENABLE BREAK, DISABLE BREAK, CANCEL BREAK, SET DATABREAK, ENABLE DATABREAK, DISABLE DATABREAK, CANCEL DATABREAK

*1: For detail, see "2.3.6 Real-time Trace".

*2: For detail, see "2.3.1.6 Debug Function".

*3: For detail, see "2.2.1.4 Memory Mapping".

*4: For detail, see "2.3.4 Break".

2.3.4 Break

The Emulator can use the following functions:

- **Code break**
- **Data monitoring break**
- **Data break**

■ Code Break Setup

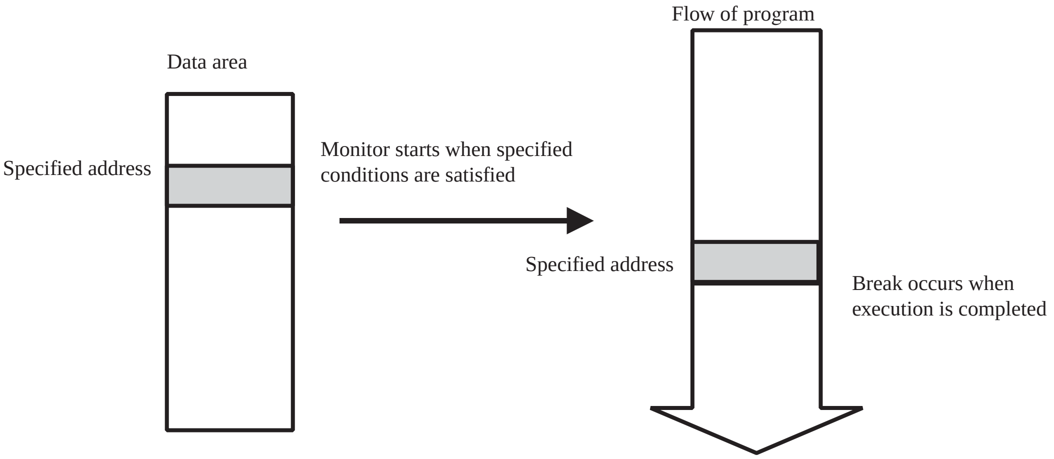
Code breaks can be set with no limits.

It is possible to set a break point while executing a user program if enabled by setting a break point while executing in the execution tabs [Environment] - [Setup Debugging Environment] - [Debug Environment] menus. However, when setting a break point, execution temporarily stops for a maximum of 1 ms.

■ Data Monitoring Breaks Setup

This is a particular function that aborts the program execution when the program reaches a specified address during matching with specified data. There are two patterns of software and hardware.

The figure below shows the conditions of data monitoring break.


Flow of program
Data area
Specified address
Monitor starts when specified conditions are satisfied
Specified address
Break occurs when execution is completed

The maximum constant of data monitoring breaks is calculated as follows:

Current data monitoring break maximum constant

= 8 - (current trace trigger count setting + current event count setting)

Use the following command for data monitoring break setup:

```
SET BREAK /DATAWATCH
```

■ Data Break Setup

Up to two data breaks can be set. The number of measurements and generated event triggers can be measured.

It is possible to set a break point while executing a user program if enabled by setting a break point while executing in the execution tabs [Environment] - [Setup Debugging Environment] - [Debug Environment] menus. However, when setting a break point, execution temporarily stops for a maximum of 1 ms.

2.3.5 Control by Sequencer

This emulator has a sequencer to control events. By using this sequencer, sampling of breaks or traces can be controlled while monitoring program flow (sequence). A break caused by this function is called a sequential break.
Use the SET EVENT command to set events.

■ Control by Sequencer

As shown in Table 2.3-4, controls can be made at 3 different levels.

One event can be set for one level.

The sequencer always moves from Level 1 through Level 2 to Level 3. One event can be specified as a sequencer restart condition.

When the debug function on MB2147-01 is set to Trace Enhancement mode, it is possible to control a trace by a sequencer.

1. Complete the trace acquisition.
2. Transit to the next block (Only in multitrace mode)

Table 2.3-4 Sequencer Specifications

Function	Specifications
Level count	3 levels+ restart condition
Conditions settable for each level	1 event conditions (1 to 16777215 times pass count can be specified for each condition.)
Restart conditions	1 event conditions (1 to 16777215 times pass count can be specified.)
Operation when conditions established	Branching to another level or terminating sequencer

■ Setting Events

The emulator can monitor the MCU bus operation, and generate a trigger for a sequencer at a specified condition. This function is called an event.

In the event, code (/CODE) and data access (/READ/WRITE) can be specified.

Up to eight events can be set. However, since hardware is shared with trace triggers, the actual numbers is calculated as follows.

Current maximum constant of events
= 8 - (current number of trace trigger settings + current number of data monitoring break settings)

Table 2.3-5 shows the conditions that can be set for events.

Table 2.3-5 Conditions for Event and Trace Trigger

Condition	Description
Address	Memory location (address bit masking disabled)
Data	16-bit data (data bit masking enabled)
Access size	Byte, word
Status	Select from code, data read or data write

Note:

In instruction execution (/CODE), an event trigger is generated only when an instruction is executed. This cannot be specified concurrently with other status (/READ or /WRITE).

Use the following commands to set an event.

SET EVENT :	Sets an event
SHOW EVENT :	Displays the status of event setting
CANCEL EVENT :	Deletes an event

[Example]

```
>SET EVENT/CODE  func1
>SET EVENT/WRITE data[2],!d=h'10
>SET EVENT/READ/WRITE 102
```

2.3.5.1 Setting Sequencer

The sequencer operates in the following order:

1) The sequencer starts after the program execution.

2) Depending on the setting at each level (1 & 2), branching to the next level is performed when the condition is met.

3) The sequencer is restarted when the restart condition is met.

4) The sequencer is terminated and a break occurs when the level 3 condition is met.

■ Setting Sequencer

The sequencer operates in the following order: The event can be set at each level and as a restart condition.

1. The sequencer starts after the program execution.
2. Depending on the setting at each level (1 & 2), branching to the next level is performed when the condition is met.
3. The sequencer is restarted when the restart condition is met.
4. The sequencer is terminated and a break occurs when the level 3 condition is met.

Use the following commands to set the sequencer.

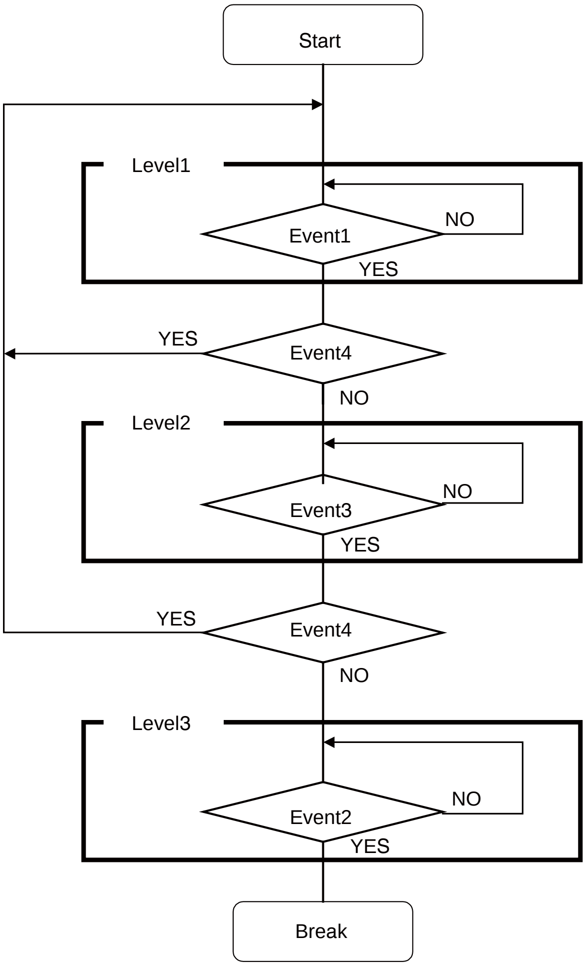
SET SEQUENCE: Setting an event for the sequencer

[Example]

```
>SET SEQUENCE 1, 3, 2, r=4
```

Set event 1, 3, 2 to level 1, 2, 3 respectively, and event 4 for the restart condition.

Figure 2.3-1 Operation of Sequencer



2.3.6 Real-time Trace

While execution a program, the address, data and status information, and the data sampled by an external probe can be sampled in machine cycle units and stored in the trace buffer. This function is called real-time trace.
In-depth analysis of a program execution history can be performed using the data recorded by real-time trace.

■ Trace Buffer

The data recorded by sampling in machine cycle units, is called a frame.

The trace buffer can store 64K frames (65536). Since the trace buffer has a ring structure, when it becomes full, it automatically returns to the start to overwrite existing data.

■ Trace Data

Data sampled by the trace function is called trace data.

The following data is sampled:

- Address
- Data
- Status Information
  - Access status: Read/Write/Internal access, etc.
  - Device status: Instruction execution, Reset, Hold, etc.
  - Queue status: Count of remaining bytes of instruction queue, etc.
  - Data valid cycle information: Data valid/invalid

    (Since the data signal is shared with other signals, it does not always output data. Therefore, the trace samples information indicating whether or not the data is valid.)
- Execution time based on the previous trace frame (in 25-ns units)

■ Data Not Traced

The following data does not leave access data in the trace buffer.

- **Portion of access data while in native mode.**

  When operating in the native mode, the F^2MC-16L/16LX family of chips sometime performs simultaneous multiple bus operations internally. However, in this emulator, monitoring of the internal ROM bus takes precedence. Therefore, other bus data being accessed simultaneously may not be sampled (in the debugging mode, all operations are sampled).

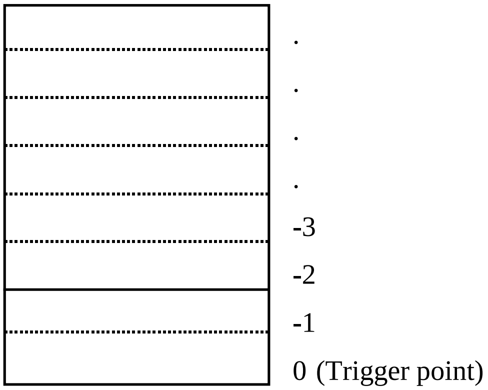
■ Frame number

A number is assigned to each frame of sampled trace data. This number is called a frame number.

The frame number is used to specify the display start position of the trace buffer. The value 0 is assigned to trace data at the triggering position for sequencer termination. Negative values are assigned to trace data that have been sampled before arrival at the triggering position (See Figure 2.3-1).

If there is no triggering position for sequencer termination, the value 0 is assigned to the last-sampled trace data.

Figure 2.3-2 Frame Numbering at Tracing



■ Trace Filter

To make effective use of the limited trace buffer capacity, in addition to the code fetch function, a trace filter function is incorporated to provide a means of acquiring information about data accesses to a specific region.

The data trace filter function allows the following values to be specified for three regions:

- Address
- Address mask
- Access attribute (read/write)

Another function can be used so that sampling of redundant frames occupying two or more trace frames, such as SLEEP and READY, can be reduced to sampling of one frame.

■ Trace Trigger Setup

When preselected conditions are met during MCU bus operation monitoring, a trigger for starting a trace can be generated. This function is called a trace trigger.

For the use of the trace trigger function, specify the code (/CODE) and data access (/READ/WRITE).

Up to 8 trace triggers can be preset each for code attribute and data access attribute. However, actually, the maximum number of trace triggers is determined as indicated below because the common hardware is used with events.

Current trace trigger maximum constant
= 8 - (current data monitoring break count setting + current event count setting)

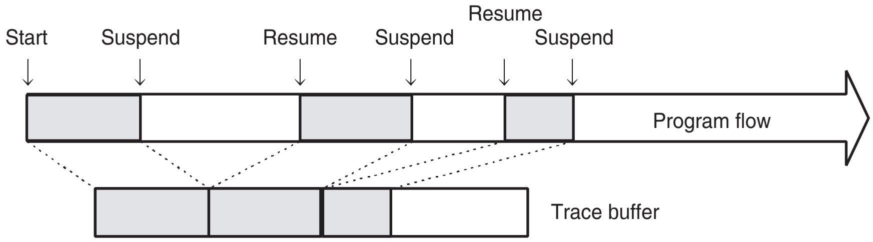
For the trace trigger setup conditions that can be defined, see Table 2.3-4.

For trace trigger setup, use the following commands:

- SET TRACETRIGGER :	Trace trigger setup
- CANCEL TRACETRIGGER :	Trace trigger deletion
- SHOW TRACE/STATUS :	Trace setup status display

Figure 2.3-2 shows a trace sampling operation.

Figure 2.3-3 Trace Sampling Control (Trace Trigger)



■ Setting Data Monitoring Trace Trigger

When the debug function on MB2147-01 is set to Trace Enhancement mode, it is possible to set a trace trigger by a data monitoring condition.

For the data monitoring condition, see the data monitoring break in "2.3.4 Break".

Current maximum constant of data monitoring trace triggers
= 8 - (number of data monitoring break settings + number of trace trigger settings +
current number of event settings)

Use the following commands to set the data monitoring trace trigger.

SET TRACETRIGGER/DATAWATCH :	Sets a data monitoring trace trigger
CANCEL TRACETRIGGER/DATAWATCH :	Deletes a data monitoring trace trigger
SHOW TRACETRIGGER/DATAWATCH :	Displays a data monitoring trace trigger

■ Trace Control during Executing User Program

In MB2147-01, the trace control is enabled while the user program is executed. However, it is necessary to end the trace execution.

The parameter that can be controlled is as follows;

- Set trace trigger
- Set filtering area
- Display trace data
- Clear trace data
- Search trace data
- Set trace delay*
- Display measurement result of time*
- Forced termination/resumption of trace execution*

*: Only when the debugging is in trace enhancement mode.

Note:

The trace execution means the trace data acquisition is "Tracing" or "Pause".

The following method exists to terminate the trace execution.

1. Forced termination of trace execution
   - Trace window - Shortcut menu [Forced termination]
   - Trace toolbar [Forced termination] button
2. Trace trigger (Termination)
   - ET TRACE TRIGGER command
   - Trace trigger setting dialog

2.3.6.1 Setting Single Trace

To perform a single trace, follow steps 1 through 4 below. When a program is executed after completion of the following steps, trace data is sampled.

1) Set an event mode to single trace mode.

2) Enable the trace function.

3) Perform the event and sequencer setup.

4) Perform trace buffer full break setup.

■ Setting Trace

To perform a single trace, complete the following setup steps. When a program is executed after completion of the steps, trace data is sampled.

1) Set an event mode to single trace mode.

Use SET MODE command for this setting.

2) Enable the trace functions.

Enable the trace function using the ENABLE TRACE command.

To disable the trace function, use the DISABLE TRACE command.

Note that the trace function is enabled by default when the program is launched.

3) Perform the event and sequencer setup.

Use of a trace trigger makes it possible to control trace sampling and make effective use of the limited trace buffer capacity. If there is no such necessity, setup need not be performed.

With a trace trigger, it is possible to specify the start and stop of trace sampling to be performed at a trigger hit.

To use a trace trigger, input the SET TRACE/TRIGGER command and then perform trace trigger setup using the SET TRACETRIGGER command.

4) Perform trace buffer full break setup.

A break can be invoked when the trace buffer becomes full.

To perform setup, use the SET TRACE command. This break feature is disabled when the program starts. To view the setting, use SHOW TRACE/STATUS.

Table 2.3-6 lists trace-related commands in the single trace.

Table 2.3-6 Trace-related Commands Available in Single Trace

Available command	Function
SET TRACETRIGGER CANCEL TRACETRIGGER	Sets trace trigger Deletes trace trigger
SET TRACE SHOW TRACE SEARCH TRACE ENABLE TRACE DISABLE TRACE CLEAR TRACE	Sets trace buffer full break Displays trace data Searches for trace data Enables trace function Disables trace function Cleares trace function

2.3.6.2 Multi Trace

Only when an event trigger occurred, the multitrace samples data before and after the event trigger.

■ Multi Trace

To use the multi trace function, the SET MODE command is set to the following mode.

Debug function: "Trace Enhancement" mode

Event mode: "Multi trace" mode

The multitrace samples data where an event trigger (trace end trigger) occurs before and after the event trigger.

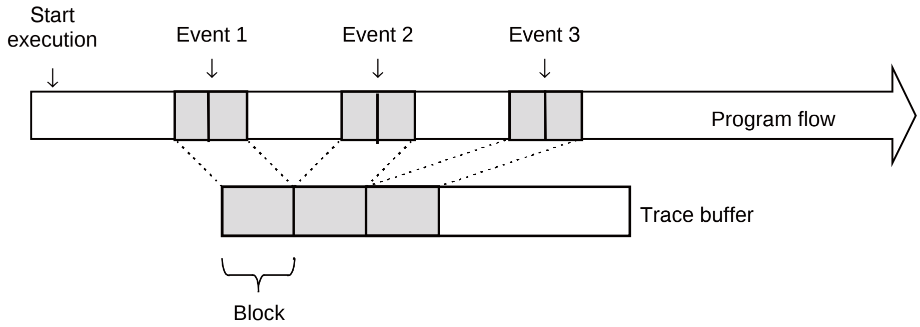
It can be used for tracing required only when a certain variable access occurs, instead of continuous tracing.

The trace data sampled at one event trigger is called a block. The trace buffer for multitrace in MB2147-01 can hold 64K frames. When dividing into blocks, select the size of one block from 128/256/512/1024 frame. 64 to 512 blocks can be sampled according to the block size.

There are the following two event triggers of the multi trace.

- Trace end trigger: Change to the next block in the point that becomes a hit.
- Multi trace end trigger: Terminate the trace acquisition in the point that becomes a hit.

Figure 2.3-4 Multi Trace Sampling



■ Multi Trace Frame Number

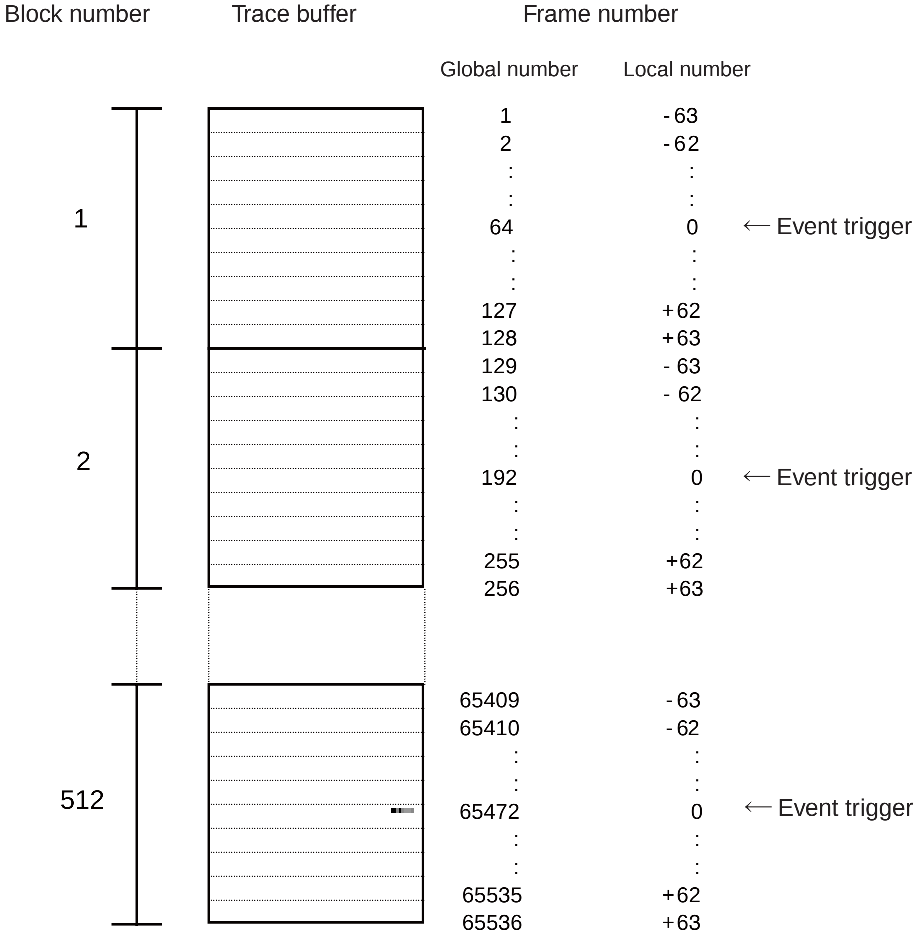
Data of 128 to 1024 frames can be sampled according to the block size at each time an event occurs (trace end trigger). This data unit is called a block, and each sampled block is numbered starting from 0. This is called the block number.

A block is a collection of sampled data before and after the event trigger occurs. At the event trigger is 0, trace data sampled before reaching the event trigger point is numbered negatively, and trace data sampled after the event trigger point is numbered positively. These frame numbers are called local numbers (See Figure 2.3-5).

In addition to this local number, there is another set of frame numbers starting 1 with the oldest data in the trace buffer. This is called the global number. Since the trace buffer can hold 64K frames, frames are numbered 1 to 65536 (See Figure 2.3-5).

To specify which frame data is displayed, use the global number or block and local numbers.

Figure 2.3-5 Frame Number in Multi Trace



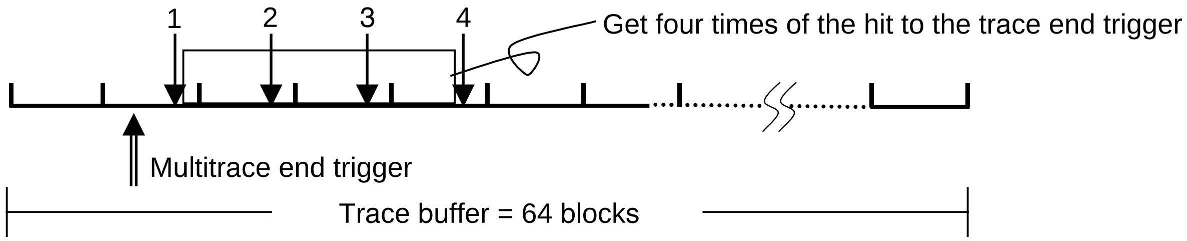
■ Trace Delay

The trace data which is acquired after one event occurrence is called a trace delay. There are two types of trace delay depending on the event hit.

When the trace end trigger (event) hit occurs, the delay can be set within the scope of the block size (128 to 1024 frames). A block is sampled data in combination with the trace data before the event hit and the trace delay.

When the multitrace end trigger (event) hit occurs, the delay is acquired as many as the number of occurrence of the subsequent trace end trigger hit.

Example: If you want to get the trace delay for three blocks, the event hit needs to occur four times.



Note:

The multitrace function in MB2147-01 is exclusive with the RAM Checker function. For more details, see "2.3.1.6 Debug Function".

2.3.6.3 Setting Methods of Multi Trace

Before executing the multitrace, the following settings must be made. After these settings, trace data is sampled when a program is executed.

1. **Set the debug function to "Trace Enhancement" mode.**
2. **Set event mode to multitrace mode.**
3. **Enable trace function.**
4. **Set event and sequencer.**
5. **Set trace-buffer-full break.**

■ Setting Methods of Multi Trace

Before executing the multitrace, the following settings must be made. After these settings, trace data is sampled when a program is executed.

1) Set the debug function to Trace Enhancement mode.
Use SET MODE command for this setting.

2) Set event mode to multitrace mode.
Use the SET MODE command for this setting.

3) Enable trace function.
Use the ENABLE MULTITRACE command for this setting. To disable the function, use the DISABLE MULTITRACE command.

4) Set an event (trace trigger).
Set an event for sampling the multitrace. Use the SET TRACETRIGGER command for this setting.

5) Set trace-buffer-full break.
To break when the trace buffer becomes full, set the trace-buffer-full break. Use the SET MULTITRACE command for this setting.

6) Set a block size.
Use SET MULTITRACE command to set this.

7) Set a trace delay.
Use SET DELAY command to set this.

Table 2.3-7 shows the list of trace-related commands that can be used in multitrace mode.

Table 2.3-7 Trace-related Commands That Can Be Used in Multi Trace Mode

Mode	Usable Command	Function
Multi trace mode	SET TRACETRIGGER SHOW TRACETRIGGER CANCEL TRACETRIGGER ENABLE TRACETRIGGER DISABLE TRACETRIGGER	Sets events Displays event setup status Deletes event Enables event Disables event
	SET MULTITRACE SHOW MULTITRACE SEARCH MULTITRACE ENABLE MULTITRACE DISABLE MULTITRACE CLEAR MULTITRACE	Sets trace-buffer-full break Displays trace data Searches trace data Enables trace function Disables trace function Clears trace data
	SET DELAY SHOW DELAY	Sets trace delay Displays trace delay

2.3.6.4 Displaying Trace Data Storage Status

It is possible to displays how much trace data is stored in the trace buffer. This status data can be read by specifying /STATUS to the SHOW TRACE command.

■ Displaying Trace Data Storage Status

It is possible to displays how much trace data is stored in the trace buffer. This status data can be read by specifying /STATUS to the SHOW TRACE.

[Example]

```
>SHOW TRACE/STATUS
en/dis      = enable                Trace function enabled
buffer full = nobreak               Buffer full break function disabled
sampling    = end                   Trace sampling terminates
code        = enable                Code execution enabled
verbose     = disable               Verbose trace disabled
frame no. = -00120 to  00000        Frame -120 to 0 store data
>
```

2.3.6.5 Specify Displaying Trace Data Storage Status

The data display start position in the trace buffer can be specified by inputting a step number or frame number using the SHOW TRACE command. The data display range can also be specified.

■ Specifying Displaying Trace Data Start

Specify the data display start position in the trace buffer by inputting a step number or frame number using the SHOW TRACE command. The data display range can also be specified.

[Example]

- In Single Trace Mode

>SHOW TRACE/CYCLE -6	Start displaying from frame -6
>SHOW TRACE/CYCLE -6..0	Display from frame -6 to frame 0
>SHOW TRACE -6	Start displaying from frame -6
>SHOW TRACE -6..0	Displays from frame -6 to frame 0

2.3.6.6 Display Format of Trace Data

The trace data display format can be selected by running the SHOW TRACE command with a command modifier specified. If setup is completed with the SET SOURCE command so as to select a source line addition mode, a source line is attached to the displayed trace data.

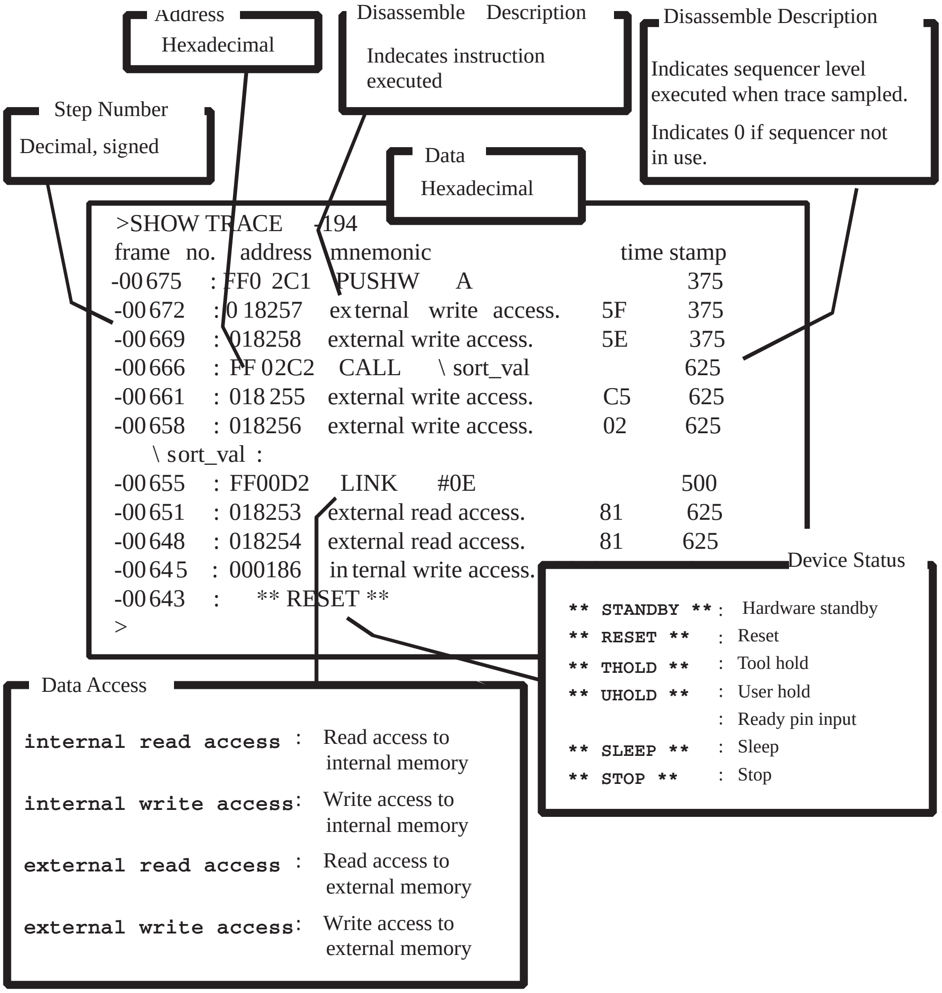
There are three formats to display trace data:

- **Display in instruction execution order (Specify /INSTRUCTION.)**
- **Display all machine cycles (Specify /CYCLE.)**
- **Display in source line units (Specify /SOURCE.)**

■ Display in Instruction Execution Order (Specify /INSTRUCTION.)

Trace sampling is performed at each machine cycle, but the sampling results are difficult to display because they are influenced by pre-fetch, etc. This is why the emulator has a function to allow it to analyze trace data as much as possible. The resultant data is displayed after processes such as eliminating pre-fetch effects, analyzing execution instructions, and sorting in instruction execution order are performed automatically. However, this function can be specified only in the single trace while in the debugging mode.

In this mode, data can be displayed in the following format.



Address
Hexadecimal

Disassemble Description
Indecates instruction executed

Disassemble Description
Indicates sequencer level executed when trace sampled.
Indicates 0 if sequencer not in use.

Step Number
Decimal, signed

Data
Hexadecimal

```
>SHOW TRACE    -194
frame  no.  address  mnemonic                          time stamp
-00675   : FF0  2C1   PUSHW     A                            375
-00672   : 0 18257   external  write  access.     5F        375
-00669   : 018258   external write access.        5E        375
-00666   : FF 02C2   CALL     \ sort_val                    625
-00661   : 018 255   external write access.       C5        625
-00658   : 018256   external write access.        02        625
     \ sort_val  :
-00655   : FF00D2    LINK     #0E                          500
-00651   : 018253   external read access.         81        625
-00648   : 018254   external read access.         81        625
-00645   : 000186   internal write access.
-00643   :      ** RESET **
>
```

Device Status

`** STANDBY **`	:	Hardware standby
`** RESET **`	:	Reset
`** THOLD **`	:	Tool hold
`** UHOLD **`	:	User hold
	:	Ready pin input
`** SLEEP **`	:	Sleep
`** STOP **`	:	Stop

Data Access

`internal read access`	:	Read access to internal memory
`internal write access`	:	Write access to internal memory
`external read access`	:	Read access to external memory
`external write access`	:	Write access to external memory

■ Displaying All Machine Cycles

Detailed information at all sampled machine cycles can be displayed.

In this mode, no source is displayed irrespective of the setup defined by the SET SOURCE command.

[Example]

```
>SHOW TRACE/CYCLE -672
frame no. address     data   a-status   d-status   Qst dfg   event   time   stamp
-00672    : 018257    ----   EWA        -------    ---  &                   125
-00671    : 018257    5F     ---        EXECUTE    ---  @                   125
-00670    : 018257    5F     ---        EXECUTE    ---  @                   125
-00669    : 018257    ----   EWA        -------    ---  &                   125
-00668    : 018257    82     ---        EXECUTE    ---  @                   125
-00667    : FF02C6    5F     ---        EXECUTE    ---  @                   125
-00666    : ------    ----   ---        EXECUTE    4by       C              125
-00665    : FF02C6    ----   ICF         -------   ---  &    D              125
-00664    : FF02C6    5F06   ---        EXECUTE    FLH@                     125
-00663    : FF00D2    ----   ICF        -------    ---  &                   125
```

```
-00662   : FF00D2    0E08   ---        EXECUTE  ---  @                     125
-00661   : 018255    ----   EWA        -------  ---  &                     125
```

How to read trace data

```
frame no.  address  data  a-status  d-status  Qst  dfg  event  time stamp
   (1)        (2)     (3)     (4)       (5)      (6)  (7)   (8)       (9)
```

(1):frame number (Decimal number)

(2):executed instruction address, and data access address (Hexadecimal number)

(3):data (Hexadecimal number)

(4):access information (a-status)

IWA	:	write access to internal memory
EWA	:	write access to external memory
IRA	:	read access to internal memory
ERA	:	read access to external memory
ICF	:	code fetch to internal memory
ECF	:	code fetch to external memory
---		:valid "d-status" information

(5):device status (d-status)

STANDBY	:	hardware standby
THOLD	:	tool hold
UHOLD	:	user hold
WAIT	:	ready pin by waiting
SLEEP	:	sleep
STOP	:	stop
EXECUTE	:	execute instruction
RESET	:	reset
-------	:	invalid d-status information

(6):instruction queue status

FLH	:	flush queue
?by	:	number of remainder code of queue is ? byte (?: 1 to 8)

(7):information valid flag

&	:	address is valid
@	:	data is valid

(8):event information

C	:	code event
D	:	data event

(9):time stamp display (ns unit)

displays difference of executed time between this frame and next frame (decimal)

Note:

Information about event hits is excluded from the displayed information. For code execution, in particular, the effect of a prefetch is eliminated in consideration of the count of data in the instruction queue. Therefore, the information about hits is displayed for frames after a prefetch frame at an address for which an event is set.

■ Display in Source Line Units (Specify /SOURCE.)

Only the source line can be displayed. This mode is enabled only in the debugging mode.

[Example]

```
>SHOW TRACE/SOURCE -1010..-86
step no.    source
-01007      : sample.c$68            value[i] =  &target[I];
-00905      : sample.c$68            value[i] =  &target[I];
-00803      : sample.c$68            value[i] =  &target[I];
-00698      : sample.c$70      sort_val(value, 16L);
-00655      : sample.c$9   {
-00594      : sample.c$13      for (k = max / 2; k  >= 1; k--){
-00185      : sample.c$14                  i = k;
-00149      : sample.c$15            p = tblp[i - 1];
-00088      : sample.c$16            while ((j = 2 * i)  <= max){
```

Note:

The following operation may be subjected to trace sampling immediately after the MCU operation is stopped (tool hold). Remember that the operation is unique to evaluation chips and not performed by mass-produced products.

Access to address 0x000100 and addresses between 0x0FFFFDC and 0x0FFFFFF

2.3.6.7 Reading Trace Data On-the-fly

Trace data can be read while executing a program. However, this is not possible during sampling. Disable the trace function or terminate tracing before attempting to read trace data.

■ Reading Trace Data On-the-fly

To disable the trace function, use the DISABLE TRACE command. Check whether or not the trace function is currently enabled by executing the SHOW TRACE command with /STATUS specified, or by using the built-in variable, %TRCSTAT.

Tracing terminates when the delay count ends after the sequencer has terminated. If Not Break is specified here, tracing terminates without a break operation. It is possible to check whether or not tracing has terminated by executing the SHOW TRACE command with /STATUS specified, or by using the built-in variable, %TRCSAMP.

To read trace data, use the SHOW TRACE command; to search trace data, use the SEARCH TRACE command. Use the SET DELAY command to set the delay count and break operation after the delay count.

[Example]

```
>GO
>>SHOW TRACE/STATUS
en/dis        = enable
buffer ful    = nobreak
sampling      = on                <- Trace sampling continues.
code          : enable
verbose       : disable
>>SHOW TRACE/STATUS
en/dis        = enable
buffer full   = nobreak
sampling      = end               <- Trace sampling ends.
code          : enable
verbose       : disable
frame no.     = -00805 to  00000
>>SHOW TRACE -52
step no.      address      mnemonic                    time stamp
      sort_val:
-00655        : FF00D2     LINK          #0E           625
-00651        : 018253     external  read access.      81     500
-00648        : 013254     external  read access.      81     625
-00645        : 000186     internal  write access.     0000   625
                     .                  .            .
```

If the CLEAR TRACE command is executed with the trace ending state, trace data sampling can be re-executed by re-executing the sequencer from the beginning.

2.3.6.8 Saving Trace Data

The debugger has function of saving trace data to a file.

■ Saving Trace Data

Save the trace data to the specified file.

For details on operations, refer to Sections 3.14 Trace Window, and 4.4.8 Trace in the SOFTUNE Workbench Operation Manual; and Section 4.23 SHOW TRACE in the SOFTUNE Workbench Command Reference Manual.

2.3.7 Measuring Performance

It is possible to measure the time and pass count between two events. Repetitive measurement can be performed while executing a program in real-time, and when done, the data can be totaled and displayed.
Using this function enables the performance of a program to be measured. To measure performance, set the event mode to the performance mode using the SET MODE command.

■ Performance Measurement Function

The performance measurement allows the time between two event occurrences to be measured and the number of event occurrences to be counted. Up to 32765 event occurrences can be measured.

- **Measuring Time**

Measures time interval between two events.

Events can be set at 8 points (1 to 8). However, in the performance measurement mode, the intervals, starting event number and ending event number are combined as follows. Four intervals have the following fixed event number combination:

Interval	Starting Event Number	Ending Event Number
1	1	2
2	3	4
3	5	6
4	7	8

- **Measuring Count**

The specified events become performance measurement points automatically, and occurrences of that event are counted.

2.3.7.1 Performance Measurement Procedures

Performance can be measured by the following procedure:

- **Set event mode.**
- **Set minimum measurement unit for timer.**
- **Specify performance-buffer-full break.**
- **Set events.**
- **Execute program.**
- **Display measurement result.**
- **Clear measurement result.**

■ Setting Event Mode

Set the event mode to the performance mode using the SET MODE command. This enables the performance measurement function.

[Example]

```
>SET MODE/PERFORMANCE
>
```

■ Setting Minimum Measurement Unit for Timer

It is 1ns as the minimum measurement unit for the timer used to measure performance. And a resolution of performance measurement data is 25ns.

■ Setting Performance-Buffer-Full Break

When the buffer for storing performance measurement data becomes full, a executing program can be broken. This function is called the performance-buffer-full break. The performance buffer becomes full when an event occurs 65535 times.

If the performance-buffer-full break is not specified, the performance measurement ends, but the program does not break.

[Example]

```
>SET PERFORMANCE/NOBREAK          <- Specifying Not Break
>
```

■ Setting Events

Set events using the SET EVENT command.

The starting/ending point of time measurement and points to measure pass count are specified by events.

Events at 8 points (1 to 8) can be set. However, in the performance measurement, the intervals, starting event number and ending event number are fixed in the following combination.

- **Measuring Time**

  Four intervals have the following fixed event number combination.

Interval	Starting Event Number	Ending Event Number
1	1	2
2	3	4
3	5	6
4	7	8

- **Measuring Count**

  The specified events become performance measurement points automatically.

■ Executing Program

Start measuring when executing a program by using the GO or CALL command. If a break occurs during interval time measurement, the data for this specific interval is discarded.

■ Displaying Performance Measurement Data

Display performance measurement data by using the SHOW PERFORMANCE command.

■ Clearing Performance Measurement Data

Clear performance measurement data by using the CLEAR PERFORMANCE command.

[Example]

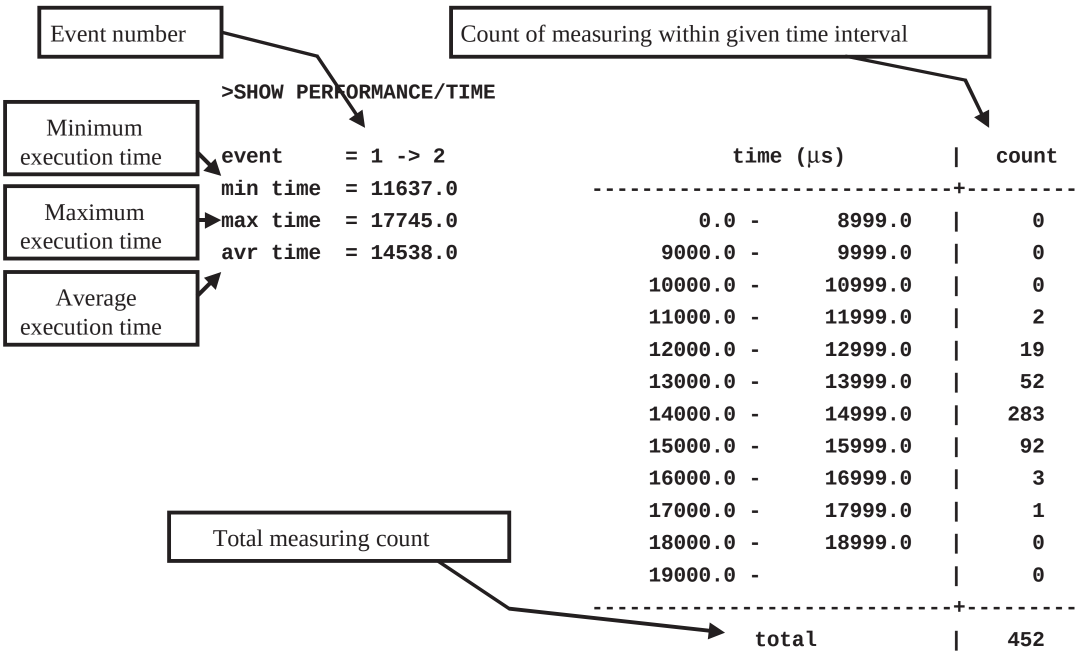
```
>CLEAR PERFORMANCE
>
```

2.3.7.2 Display Performance Measurement Data

Display the measured time and measuring count by using the SHOW PERFORMANCE command.

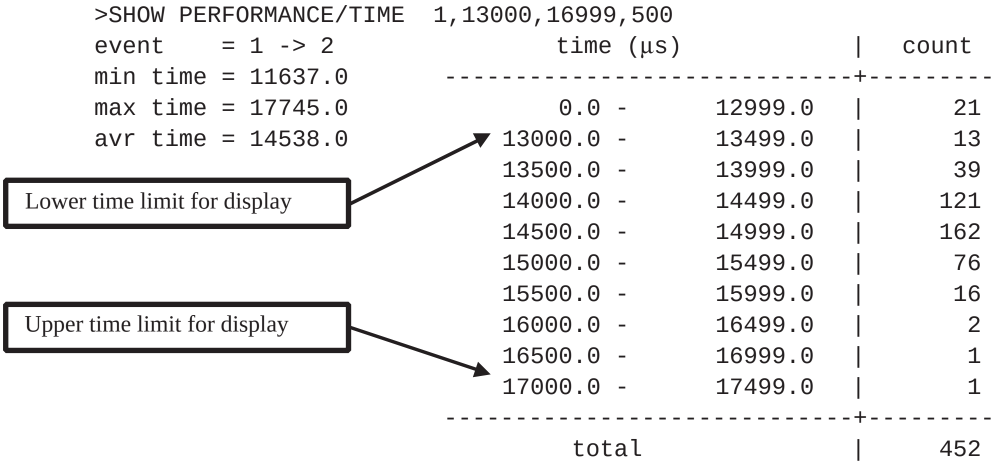
■ Displaying Measured Time

To display the time measured, specify the starting event number or the ending event number.



```
>SHOW PERFORMANCE/TIME

event     = 1 -> 2                time (µs)        |   count
min time  = 11637.0      ----------------------------+---------
max time  = 17745.0            0.0 -      8999.0   |      0
avr time  = 14538.0         9000.0 -      9999.0   |      0
                           10000.0 -     10999.0   |      0
                           11000.0 -     11999.0   |      2
                           12000.0 -     12999.0   |     19
                           13000.0 -     13999.0   |     52
                           14000.0 -     14999.0   |    283
                           15000.0 -     15999.0   |     92
                           16000.0 -     16999.0   |      3
                           17000.0 -     17999.0   |      1
                           18000.0 -     18999.0   |      0
                           19000.0 -               |      0
                         ----------------------------+---------
                                 total             |    452
```



```
>SHOW PERFORMANCE/TIME  1,13000,16999,500
event    = 1 -> 2              time (µs)            |   count
min time = 11637.0      -----------------------------+---------
max time = 17745.0            0.0 -      12999.0   |      21
avr time = 14538.0        13000.0 -      13499.0   |      13
                          13500.0 -      13999.0   |      39
                          14000.0 -      14499.0   |     121
                          14500.0 -      14999.0   |     162
                          15000.0 -      15499.0   |      76
                          15500.0 -      15999.0   |      16
                          16000.0 -      16499.0   |       2
                          16500.0 -      16999.0   |       1
                          17000.0 -      17499.0   |       1
                        -----------------------------+---------
                               total                 |     452
```

2.3.8 Measuring Coverage

This emulator has the C0 coverage measurement function. Use this function to find what percentage of an entire program has been executed.

■ Coverage Measurement Function

When testing a program, the program is executed with various test data input and the results are checked for correctness. When the test is finished, every part of the entire program should have been executed. If any part has not been executed, there is a possibility that the test is insufficient.

This emulator coverage function is used to find what percentage of the whole program has been executed. In addition, details such as which addresses were not accessed can be checked.

This enables the measurement coverage range to be set.

To execute the C0 coverage, set a range within the code area. In addition, setting a range in the data area, permits checking the access status of variables such as finding unused variables, etc.

Execution of coverage measurement is limited to the address space specified as the debug area.

Therefore, set the debug area in advance.

This is operable by enabling the coverage function on the chip tabs: [Environment] - [Setup Debugging Environment] - [Debug Environment] command.

■ Coverage Measurement Procedures

The procedure for coverage measurement is as follows:

1.	Set range for coverage measurement:	SET COVERAGE
2.	Measuring coverage:	GO, STEP, CALL
3.	Displaying measurement result:	SHOW COVERAGE

■ Coverage Measurement Operation

The following operation can be made in coverage measurement:

-	Load/Save of coverage data:	LOAD/COVERAGE, SAVE/COVERAGE
-	Abortion and resume of coverage measurement:	ENABLE COVERAGE, DISABLE COVERAGE
-	Clearing coverage data:	CLEAR COVERAGE
-	Canceling coverage measurement range:	CANCEL COVERAGE

Note:

When the coverage measurement function is used, the monitoring function in RAM area of the 0 bank cannot be used. For more details, see "2.3.9 Real-time Monitoring".

2.3.8.1 Coverage Measurement Procedures

The procedure for coverage measurement is as follows:

- Set range for coverage measurement	**:**	**SET COVERAGE**
- Measure coverage	**:**	**GO, STEP, CALL**
- Display measurement result	**:**	**SHOW COVERAGE**

■ Setting Range for Coverage Measurement

Use the SET COVERAGE command to set the measurement range. The measurement range can be set only within the area defined as the debug area. Up to 32 ranges can be specified.

By specifying /AUTOMATIC for the command qualifier, the code area for the loaded module is set automatically. However, the library code area is not set when the C compiler library is linked.

[Example]

```
>SET COVERAGE FF0000 .. FFFFFF
```

■ Measuring Coverage

When preparing for coverage measurement, execute the program.

Measurement starts when the program is executed by using the GO, STEP, or CALL command.

■ Displaying Coverage Measurement Result

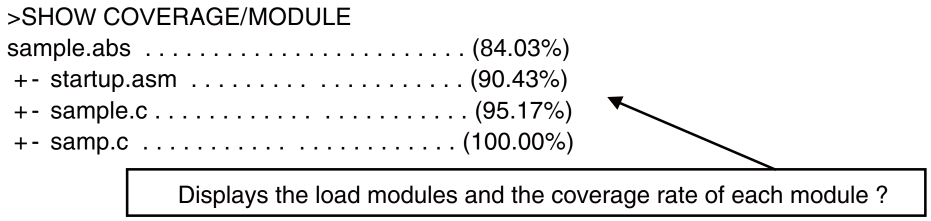
To display the measurement result, use the SHOW COVERAGE command. The following can be displayed:

- Display coverage rate of total measurement area
- Displaying coverage rate of load module
- Summary of 16 addresses as one block
- Details indicating access status of each address
- Displaying coverage measurement result per source line
- Displaying coverage measurement result per machine instruction

● Display Coverage Rate of Total Measurement Area (Specify /TOTAL for command qualifier)

```
        >SHOW COVERAGE/TOTAL
    total coverage : 82.3%
```

● Displaying coverage rate of load module (Specify /MODULE for the command qualifier)



```
>SHOW COVERAGE/MODULE
sample.abs . . . . . . . . . . . . . . . . . . . . . . . . (84.03%)
 +-  startup.asm . . . . . . . .  . . . . . . . . . . . (90.43%)
 +-  sample.c . . . . . . . . . . . .  . . . . . . . . . . . (95.17%)
 +-  samp.c  . . . . . . . . . . .  . . . . . . . . . . . . (100.00%)
```

Displays the load modules and the coverage rate of each module ?

- Summary (Specify /GENERAL for command qualifier)

```
>SHOW COVERAGE/GENERAL
  (HEX)0X0        +1X0         +2X0
       +---------------+---------------+------         ------
address 0123456789ABCDEF0123456789ABCDEF0123456    ... ABCDEF   C0(%)
FF0000  **3*F*.......                                           32.0
```

Display the access status of every 16 addresses

.	: No access
1 to F	: Display the number accessed in 16 addresses by the hexadecimal number.
*	: Access all of the 16 addresses.

- Details (Specify /DETAIL for command qualifier.)

```
>SHOW COVERAGE/DETAIL FF0000

address +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +A +B +C +D +E +F C0(%)
FF0000   -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  - 100.0
FF0010   -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  - 100.0
FF0020   .  .  .  .  -  -  -  .  .  .  .  .  .  .  .  .  18.6
FF0030   -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  - 100.0
FF0040   -  .  -  -  -  -  -  -  -  -  -  -  -  -  -  -  93.7
FF0050   -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  - 100.0
FF0060   .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .   0.0
FF0070   .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .   0.0
FF0080   .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .   0.0
```

Display one line of a coverage rate

Display the access status of every 1 address

.	: No access
-	: Access

● Displays per source line (Specify /SOURCE for the command qualifier)

```
>SHOW COVERAGE/SOURCE main
 *    70: {
      71:      int    i;
      72:      struct table *value[16];
      73:
 *    74:      for (i=0; i<16; i++)
 *    75:           value[i] = &target[i];
      76:
 *    77:      sort_val(value, 16L);
 .    78: }
```

Displays access status of each source line.

. :   No Access
* :   Accessed
Blank: Line which the code had not been generated or is outside
       the scope of the coverage measurement

● Displays per machine instruction (Specify /INSTRUCTION for the command qualifier)

```
>SHOW COVERAGE/INSTRUCTION F9028F
sample.c$70 {
 * F9028F                     ¥main:
 * F9028F 0822                     LINK     #22
 * F90291 4F01                     PUSHW   RW0
sample.c$74          for (i=0; i<16; i++)
 . F90293 D0                       MOVN     A,#0
 . F90294 CBFE                     MOVW     @RW3-02,A
 . F90296 BBFE                     MOVW     A,@RW3-02
 . F90298 3B1000                   CMPW     A,#0010
 . F9029B FB18                     BGE      F902B5
sample.c$75                      value[i] = &target[i];
 . F9029D BBFE                     MOVW     A,@RW3-02
 . F9029F 0C                       LSLW     A
 . F902A0 98                       MOVW     RW0,A
 . F902A1 71F3DE                   MOVEA    A,@RW3-22
 . F902A4 7700                     ADDW     RW0,A
 . F902A6 4214                     MOV      A,#14
 . F902A8 7833FE                   MULUW    A,@RW3-02
 . F902AB 38A001                   ADDW     A,#01A0
```

Displays access status of each source line.

. :   No Access
*:   Accessed
Blank: Instruction outside the scope of the coverage measurement

2.3.9 Real-time Monitoring

The real-time monitoring function is used to display the memory contents during program execution.

■ Real-time Monitoring

The emulator can use the real-time monitoring function when the evaluation chip has the external trace bus interface.

A real-time monitoring window is provided to display two 256-byte regions for real-time monitoring purposes. The real-time monitoring window has a function for reading data from the actual memory and displaying it before program execution (copy function), and a function for displaying updated data in red.

■ RAM Area Referencing of the 0 Bank

To use the real-time monitoring function in the RAM area of the 0 bank, the coverage function must be disabled by the following methods.

- Command

  DISABLE COVERAGE

  See "4.15 DISABLE COVERAGE" in SOFTUNE Workbench Command Reference Manual.

- Dialog

  [Chip] tab on the Setup debug environment dialog.

  See "4.7.2.3 Debug Environment" in SOFTUNE Workbench Operation Manual.

2.3.10 Measuring Execution Time Using Emulation Timer

The timer for measuring time is called the emulation timer. This timer can measure the time from the start of MCU operation until suspension.

■ Measuring Executing Time Using Emulation Timer

With a resolution of 25 ns and 56 significant bits, the timer can measure the time of up to 72,057,594,037,927, multiply 935 by 25 ns.

The display shows the results of two time measurements: the time duration of the last program execution and the total time duration of all the completed program executions. If the timer overflows, a notification is displayed. Measurement is performed at each program execution. The emulation timer cannot be disabled. However, the measured execution times can be cleared.

■ Execution Cycle Measurement by Cycle Counter

The cycle counter has 56 significant bits and permits measurement of up to 72,057,594,037,927,935 cycles.

The display shows the results of two cycles count measurements: the cycle count of the last program execution, and the total cycle count of all completed program executions. If the timer overflows, a notification is displayed. Measurement is performed at each program execution. The emulation timer cannot be disabled. However, measured cycle count can be cleared.

Use the following commands for control:

- SHOW TIMER :   Indicates execution time and measured cycle count
- CLEAR TIMER :   Clears measurement results

[Example]

```
>GO  main,$25
Break at FF008D by breakpoint
>SHOW TIMER
                                m   s   ms  µs  ns
from init            =            42108264000
from last executed   =             3623874000
>CLEAR TIMER
>SHOW TIMER
                                m   s   ms  µs  ns
from init            =
from last executed   =
>
                                m   s   ms  µs  ns
from init            =          00: 00:000:000:000
from last executed   =          00: 00:000:000:000
>
```

Note:

Note that the measured execution time is added about ten extra cycles per execution.

2.3.11 Power-on Debugging

This section explains power-on debugging by the emulators for the MB2147-01.

■ Power-on Debugging

Power-ON debugging refers to the operation to debug the operating sequence that begins when the power to the target is switched on.

For products with a dedicated power-on debugging terminal, the MB2147-01 emulator can debug the sequence performed immediately after power-on. The following functions are available:

Code break

Data monitoring break

Data break

Sequencer and event

Trace trigger

Trace measurement

Coverage measurement

The power-on debugging procedure is described below:

- Set the DIP switch on the adapter board mounted in the upper part of the emulator.
- Turn on the target board and emulator main unit.
- Launch Workbench to start debugging.

  For debugging, set hardware breaks, etc.
- To start a power-on debugging, run [Execute] - [Power-ON Debug] command.

  Input the lower limit value of the monitoring voltage from the [User Power Monitor Voltage] dialog box to display PON in the input status bar.
- Run the program.
- Turn the target board off while running and then back on.
- Conduct debugging.
- To terminate the power-on debugging, run [Execute] - [Power-ON Debug] command.

2.3.12 RAM Checker

This section describes the functions of the RAM Checker.

■ Overview

The RAM checker obtains history logs of accessing the monitoring addresses on SOFTUNE Workbench and graphically displays log files using the accessory tool, RAM Checker Viewer.

SOFTUNE Workbench has the following functions

- Sets monitoring addresses at 16 points
- Logs data access history of monitoring addresses at intervals of 1 ms
- Monitors monitoring addresses at intervals of 100 ms

■ RAM Check Window

The debugging window "RAM Checker" has been added to SOFTUNE Workbench to log/monitor monitoring addresses.

For operations of Ram check Window, refer to Section 3.21 of SOFTUNE Workbench Operation Manual.

■ Use Conditions

The RAM Checker operates under the following conditions.

- Emulator: MB2147-01
- Communication device: USB

The RAM Checker cannot be used for the MB2141/MB2147-05 emulator, or the RS/LAN communication device. In those environments, the main menu [View] - [RAM Checker] is not disabled.

Note:

The RAM Checker is enabled only when the debug function on MB2147-01 is set to "RAM Checker" mode. For more details, see "2.3.1.6 Debug Function".

■ Specifications List

Monitoring Point Count	16 points
Size	Bytes/word (16 bits)
Event Functions	Max. 8 Points
Sampling Time	1 ms (Fixed)
Update Intervals	100 ms (Fixed)
Log File Formats	SOFTUNE format or CSV format

- SOFTUNE format
  - To display in the RAM Checker viewer (recommended)
  - Default extension is "SRL".
- CSV format
  - To display in other applications than the RAM Checker viewer
  - Default extension is "CSV".

Note:

The CSV format requires size of data approximately 4 times that of the SOFTUNE format.

■ To Use the RAM Checker

User sets the monitoring points, Log File, logging status by GUI or Command to use the RAM Checker.

- GUI
  - Set the debug function to "RAM Checker" mode by using [Debug] - [Debug Function].
  - By short cut menu [Setup...] on the Ram check Window, user sets the monitoring points.
  - By short cut menu [File...] on the Ram check Window, user sets the Log File.
  - By checking the short cut menu: [Logging start] on the Ram check Window, a logging status of the Ram Checker becomes to enable.
- COMMAND
  - Set the debug function to "RAM Checker" mode by using SET MODE/CONFIG command.
  - By command: SET RAMCHECK, user sets the monitoring points.
  - By command: SET RAMCHECK, user sets the Log File.
  - By command: ENABLE RAMCHECK, a logging status of the Ram Checker becomes to enable.

After these commands are set, user program execute and Log File is created by stopped user program. If it is restarted, a Log File is overwritten.

Note:

If a setting of Overwrite control is enabled on Setup file dialog, a Log File is saved with different name every other execution.

For details about settings of the RAM Checker viewer, refer to Section 3.21, SOFTUNE Workbench Operation Manual and 4.35 - 4.39, SOFTUNE Workbench Command Reference Manual.

Note:

Execution state for MCU such as stop mode or sleep mode cannot be displayed at status bar during logging.

■ About Log File

Following restrictions are made for the size of log file to be created depending on file system where log file is stored.

FAT:Up to 2GB

FAT32: Up to 4GB

NTFS: No limit.

Others: No limit

If the file system is FAT, FAT32, file name will be changed and continue logging when the size of file is exceeded limitation.

Note:

If a file is already existed, log file will be overwritten

Example of an operation

If the size of file is exceeded it's limitation, log file will be created as

filename.srl --> filename#1.srl

If the size gets exceeded the limitation again, log will be shown and changes as follows.

filename#1.srl --> filename#2.srl

•

•

filename#N-1.srl --> filename#N.srl

Notes:

1. Log files should only be saved to built-in HDD only. If network, external HDD or external disk (CD, DVD, MO etc) are used as destination for saving files, files will not be saved.
2. More than 500MB memory is required for disk to save log file of RAM checker. If the capacity of disk become less than 500MB, logging will be halted.

■ RAM Checker Viewer

The RAM Checker Viewer is a tool for graphically displaying changes in data values with the passage of time. There are the following three types of data display formats:

- Bit display (Logic Analyzer image)
- Data value display (bent line graph)
- Bit/data value display (simultaneous display bit and data values)

It displays halting CPU, trigger points and the Data Lost as other information.

To halt the operation of CPU, stop mode for low power consumption and power off condition at power-on debug function will be saved to log.

Trigger point uses event-hit in SOFTUNE Workbench. It is necessary to set event in SOFTUNE Workbench to use trigger point. When the event-hit is appeared, its information is recorded in a log.

The Data Lost is appeared in the following two causes.

- The Data Lost caused by hardware
  The emulator obtains data access history of RAM at intervals of 1 ms, but if two or more data access the same address within 1 ms, the emulator obtains only the data of the last access.
  Data loss caused by hardware indicates that several data accessed the same address.
- The Data Lost caused by software
  SOFTUNE Workbench obtains data from the emulator at intervals of 100 ms. However, other application may disable the SOFTUNE Workbench for obtaining data at intervals of 100 ms.
  In such cases, the RAM Checker Viewer does not display a portion of the data, but displays the invalid

time band graphically.

Note:

If logging is halted by break or stopping an execution, software lost could be appeared for 1ms ~ 15ms. at the end of log. This happens because log after stopping an execution will be obtained until logging is stopped, thus this is not an actual data lost.

For details of RAM Checker viewer, refer to FswbRViewE.pdf and Help.

2.4 Emulator Debugger (MB2147-05)

This section explains the functions of the emulator debuggers for the MB2147-05.

■ Emulator

When choosing the emulator debugger from the setup wizard, select one of the following emulators. Select the MB2147-05.

MB2141

MB2147-01

MB2147-05

The emulator debugger for the MB2147-05 is software that controls an emulator from a host computer via a communications line (RS-232C or USB) to evaluate programs.

The following series can be debugged:

F^{2}MC16L

F^{2}MC16LX

Before using the emulator, it must be initialized. For details, refer to Appendix B, Downloading Monitor Program in the SOFTUNE WORKBENCH Operation Manual.

2.4.1 Setting Operating Environment

This section explains the operating environment setup.

■ Setting Operating Environment

For the emulator debugger for the MB2147-05, it is necessary to set the following operating environment. Predefined default settings for all these setup items are enabled at startup. Therefore, setup is not required when using the default settings. Adjusted settings can be used as new default settings from the next time.

- Monitoring program automatic loading
- MCU operation mode
- Debug area
- Memory mapping

2.4.1.1 Monitoring Program Automatic Loading

Emulators for MB2147-05 can automatically update the monitoring program at emulator startup.

■ Monitoring Program Automatic Loading

When the emulators for MB2147-05 is specified, data in the emulator can be checked at the beginning of debugging to load an appropriate monitoring program and configuration binary data automatically into the emulator.

The monitoring program and configuration binary data to be compared for update are in Lib\907 under the directory where Workbench is installed.

Enable/disable the monitoring program automatic loading function by choosing [Environment] - [Debugging Environment Setup] - [Setup Wizard] menu.

2.4.1.2 MCU Operation Mode

There are two MCU operation modes as follows:

- **Debugging Mode**
- **Native Mode**

■ Setting MCU Operation Mode

Set the MCU operation mode.

There are two operation modes: the debugging mode, and the native mode. Choose either one using the SET RUNMODE command.

At emulator start-up, the MCU is in the debugging mode.

The data access to internal bus may not be detected by emulator in native mode. Therefore, when the MCU operation mode is changed, all the following are initialized:

- Data break points
- Trace measurement settings and trace buffer

■ Debugging Mode

All the operations of evaluation chips can be analyzed, but their operating speed is slower than that of mass-produced chips.

■ Native Mode

Evaluation chips have the same timing as mass-produced chips to control the operating speed. Note that the restrictions the shown in Table 2.4-1 are imposed on the debug functions.

Table 2.4-1 Restrictions on debug functions

Applicable series	Restrictions on debug functions
Common to all series	- When a data read access occurs on the MCU internal bus, the internal bus access information is not sampled and stored in the trace buffer. - Even when a data break or event (data access condition) is set for data on the MCU internal bus, it may not become a break factor or sequencer-triggering factor. - The coverage function may fail to detect an access to data on the MCU internal bus.

2.4.1.3 Debug Area

Set the intensive debugging area out of the whole memory space. The area functions are enhanced.

■ Setting Debug Area

There are two debug areas: DEBUG3, and DEBUG4. A continuous 1 MB area (16 banks) is set for each area.

Set the debug area using the SET DEBUG command.

Setting the debug area enhances the break point function.

- **Enhancement of Break Points**

  Up to six break points (not including temporary break points set using GO command) can be set when the debug area has not yet been set.

  When setting the debug area as the CODE attribute, up to 65535 break points can be set if they are within the area. At this time, up to six break points can be set for an area other than the debug area, but the total count of break points must not exceed 65535. In 0x00 to 0x0F and 0x0F0 to 0x0FF, a break point can be set without specifying the debug area. (DEBUGGER1, DEBUGGER2)

2.4.1.4 Memory Area Types

A unit in which memory is allocated is called an area. There are five different area types.

■ Memory Area Types

A unit to allocate memory is allocated is called an area. There are five different area types as follows:

- **User Memory Area**

  Memory space in the user system is called the user memory area and this memory is called the user memory. Up to four user memory areas can be set with no limit on the size of each area. Define a region on a 256-byte boundary.

  Access attributes can be set for each area; for example, CODE, READ, etc., can be set for ROM area, and READ, WRITE, etc. can be set for RAM area. If the MCU attempts access in violation of these attributes, the MCU operation is suspended and an error is displayed (guarded access break).

  To set the user memory area, use the SET MAP command.

- **Emulation Memory Area**

  Memory space substituted for emulator memory is called the emulation memory area, and this memory is called emulation memory.

  It is possible to set up to four areas of 256-KB maximum (including an internal ROM area described later) as emulation memory area. Define a region on a 256-byte boundary. An area larger than 256-KB can be specified at one time but is divided internally into two or more 256-KB areas for management purposes.

  Memory manipulation commands can be executed in relation to emulation memory areas while MCU execution is in progress.

  Emulation memory areas can be set using the SET MAP command.

  Further, the access attributes can be set as with user memory areas.

Note:

Even if the MCU internal resources are set as emulation memory area, access is made to the internal resources.

- **Internal ROM Area**

  The area where the emulator internal memory is substituted for internal ROM is called the internal ROM area, and this memory is called the internal ROM memory.

  Only one internal ROM area with a size up to 256-KMB can be specified.

  The internal ROM area with a size up to 1 MB can be specified 2 areas.

  Memory manipulation commands can be executed in relation to emulation memory areas while MCU execution is in progress.

  To set the internal ROM area, use "Setup CPU Information" from "Setup Project Basics". The area attribute is set automatically to READ/CODE.

Note:

The internal memory area, it is set a suitable area automatically by the selected MCU.

- **Internal ROM Image Area**

  Some types of MCUs have data in a specific area of internal ROM appearing to 00 bank. This specific area is called the internal ROM image area.

  By using "Setup CPU Information" from "Setup Project Basics", specify whether or not to set the internal ROM image area. This area attribute is automatically set to READ/CODE. The same data as in the internal ROM area appears in the internal ROM image area.

  Note that the debug information is only enabled for either one (one specified when linked). To debug only the internal ROM image area, change the creation type of the load module file.

Note:

The internal memory area, it is set a suitable area automatically by the selected MCU.

- **Undefined Area**

  A memory area that does not belong to any of the areas described above is part of the user memory area. This area is specifically called the undefined area.

  The undefined area can be set to either NOGUARD area, which can be accessed freely, or GUARD area, which cannot be accessed. Select either setup for the whole undefined area. If the area attribute is set to GUARD, a guarded access error occurs if access to this area is attempted.

2.4.1.5 Memory Mapping

Memory space can be allocated to the user memory and the emulation memory, etc., and the attributes of these areas can be specified.
However, the MCU internal resources are not dependent on this mapping setup and access is always made to the internal resources.

■ Access Attributes for Memory Areas

The access attributes shown in Table 2.4-2 can be specified for memory areas.

A guarded memory access break occurs if access is attempted in violation of these attributes while executing a program.

When access to the user memory area and the emulation memory area is made using program commands, such access is allowed regardless of the CODE, READ, WRITE attributes. However, access to memory with the GUARD attribute in the undefined area, causes an error.

Table 2.4-2 Types of Access Attributes

Area	Attribute	Description
User Memory Emulation Memory	CODE	Instruction Execution Enabled
	READ	Data Read Enabled
	WRITE	Data Write Enabled
Undefined	GUARD	Access Disabled
	NOGUARD	No check of access attribute

When access is made to an area without the WRITE attribute by executing a program, a guarded access break occurs after the data has been rewritten if the access target is the user memory. However, if the access target is the emulation memory, the break occurs before rewriting. In other words, write-protection (memory data cannot be overwritten by writing) can be set for the emulation memory area by not specifying the WRITE attribute for the area.

This write-protection is only enabled for access made by executing a program, and is not applicable to access by commands.

■ Creating and Viewing Memory Map

Use the following commands for memory mapping.

SET MAP:	Set memory map.
SHOW MAP:	Display memory map.
CANCEL MAP:	Change memory map setting to undefined.

[Example]

```
>SHOW MAP
address                 attribute           type
000000 .. FFFFFF        noguard
The rest of setting area numbers
user = 8     emulation = 5
>SET MAP/USER H'0..H'1FF
>SET MAP/READ/CODE/EMULATION H'FF0000..H'FFFFFF
>SET MAP/USER H'8000..H'8FFF
>SET MAP/MIRROR/COPY H'8000..H'8FFF
>SET MAP/GUARD
>SHOW MAP
address                 attribute           type
000000 .. 0001FF        read write          user
000200 .. 007FFF        guard
008000 .. 008FFF        read write          user
009000 .. FEFFFF        guard
FF0000 .. FFFFFF        read write code     emulation
mirror address area
008000 .. 008FFF        copy
The rest of setting area numbers
user = 6     emulation = 3
>
```

■ Internal ROM Area Setting

The [Map Setting] dialog box is displayed using [Environment] - [Debugger Memory Map] command. You can set the internal ROM area using the [Internal ROM Area] after the [Map Adding] dialog box is displayed by clicking on the [Setting] button. You can set two areas. Both require empty Emulation area to be set. You can set the region size by (Empty space of the emulation area) x (one area size).

Specify the internal ROM area from the ending address H'FFFFFF (fixed) for area 1. Also, it is possible to delete the internal ROM area.

2.4.2 Notes on Commands for Executing Program

When using commands to execute a program, there are several points to note.

■ Notes on GO Command

For the GO command, two break points that are valid only while executing commands can be set. However, care is required in setting these break points.

- **Invalid Breakpoints**
  - No break occurs when a break point is set at the instruction immediately after the following instructions.

F^{2}MC-16L/16LX	PCB	DTB
	NCC	ADB
	SPB	CNR
	MOV ILM,#imm8	AND
	OR CCR,#imm8	CCR,#imm8
		POPW PS

  - No break occurs when break point set at address other than starting address of instruction.
  - No break occurs when both following conditions met at one time.
    - Instruction for which break point set starts from odd-address,
    - Preceding instruction longer than 2 bytes length, and break point already set at last 1-byte address of preceding instruction (This "already-set" break point is an invalid break point that won't break, because it has been set at an address other than the starting address of an instruction).
- **Abnormal Break Point**

  Setting a break point at the instruction immediately after string instructions listed below, may cause a break in the middle of the string instruction without executing the instruction to the end.

F^{2}MC-16L/16LX	MOVS	MOVSW
	SECQ	SECQW
	WBTS	MOVSI
	MOVSWI	SECQI
	SECQWI	WBTC
	MOVSD	MOVSWD
	SECQD	SECQWD
	FILS	FILSI
	FILSW	FILSWI

■ Notes on STEP Command

- **Exceptional Step Execution**

  When executing the instructions listed in the notes on the GO command as invalid break points and abnormal break points, such instructions and the next instruction are executed as a single instruction. Furthermore, if such instructions are continuous, then all these continuous instructions and the next instruction are executed as a single instruction.

- **Step Execution that won't Break**

  Note that no break occurs after step operation when both the following conditions are met at one time.

  -When step instruction longer than 2 bytes length and last code ends at even address

  -When break point already set at last address (This "already-set" break point is an invalid break point that won't break, because it has been set at an address other than the starting address of an instruction.)

■ Controlling Watchdog Timer

It is possible to select "No reset generated by watchdog timer counter overflow" while executing a program using the GO, STEP, CALL commands.

Use the ENABLE WATCHDOG, DISABLE WATCHDOG commands to control the watchdog timer.

-ENABLE WATCHDOG:Reset generated by watchdog timer counter overflow

-DISABLE WATCHDOG:No reset generated by watchdog timer counter overflow

The start-up default in this program is "Reset generated by watchdog timer counter overflow".

[Example]

```
>DISABLE WATCHDOG
>GO
```

2.4.3 On-the-fly Executable Commands

Certain commands can be executed even while executing a program. This is called "on-the-fly" execution.

■ On-the-fly Executable Commands

Certain commands can be executed on-the-fly. If an attempt is made to execute a command that cannot be executed on-the-fly, an error "MCU is busy" occurs. Table 2.4-3 lists major on-the-fly executable functions. For further details, refer to the Command Reference Manual.

Meanwhile, on-the-fly execution is enabled only when executing the MCU from the menu or the tool button. On-the-fly commands cannot be executed when executing the GO command, etc., from the Command window.

Table 2.4-3 Major Functions Executable in On-the-fly Mode

Function	Restrictions	Major Commands
MCU reset	-	RESET
Displaying MCU execution status	-	SHOW STATUS
Displaying execution cycle measurement value (cycle)	-	SHOW TIMER
Memory operation (Read/Write)	Emulation memory only operable	ENTER, EXAMINE, COMPARE FILL, MOVE, DUMP, SEARCH MEMORY, SHOW MEMORY, SET MEMORY
Line assembly, Disassembly	Emulation memory only enabled	ASSEMBLE DISASSEMBLE
Load, Save program	Emulation memory only enabled	LOAD SAVE

2.4.4 Real-time Trace

While execution a program, the address, data and status information, and the data sampled by an external probe can be sampled in machine cycle units and stored in the trace buffer. This function is called real-time trace.
In-depth analysis of a program execution history can be performed using the data recorded by real-time trace.

■ Trace Buffer

The data recorded by sampling in machine cycle units, is called a frame.

The trace buffer can store 64K frames (65536). Since the trace buffer has a ring structure, when it becomes full, it automatically returns to the start to overwrite existing data.

■ Trace Data

Data sampled by the trace function is called trace data.

The following data is sampled:

- Address
- Data
- Status Information
  - Access status: Read/Write/Internal access, etc.
  - Device status: Instruction execution, Reset, Hold, etc.
  - Queue status: Count of remaining bytes of instruction queue, etc.
  - Data valid cycle information: Data valid/invalid
    (Since the data signal is shared with other signals, it does not always output data. Therefore, the trace samples information indicating whether or not the data is valid.)

■ Data Not Traced

The following data does not leave access data in the trace buffer.

- **Portion of access data while in native mode.**

  When operating in the native mode, the F^{2}MC-16L/16LX family of chips sometime performs simultaneous multiple bus operations internally. However, in this emulator, monitoring of the internal ROM bus takes precedence. Therefore, other bus data being accessed simultaneously may not be sampled (in the debugging mode, all operations are sampled).

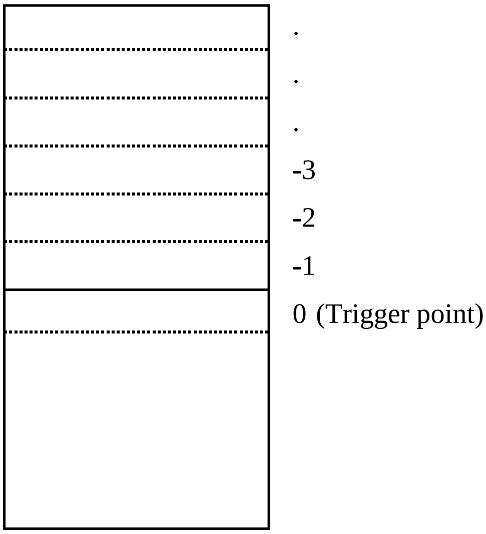
■ Frame Number

A number is assigned to each frame of sampled trace data. This number is called a frame number.

The frame number is used to specify the display start position of the trace buffer. The value 0 is assigned to trace data at the triggering position for sequencer termination. Negative values are assigned to trace data that have been sampled before arrival at the triggering position (See Figure 2.4-1).

If there is no triggering position for sequencer termination, the value 0 is assigned to the last-sampled trace data.

Figure 2.4-1 Frame Number at Tracing



■ Trace Filter

To make effective use of the limited trace buffer capacity, in addition to the code fetch function, a trace filter function is incorporated to provide a means of acquiring information about data accesses to a specific region.

The data trace filter function allows the following values to be specified for two regions:

- Address
- Address mask
- Access attribute (read/write)

Another function can be used so that sampling of redundant frames occupying two or more trace frames, such as SLEEP and READY, can be reduced to sampling of one frame.

2.4.4.1 Setting Trace

To perform a trace, follow steps (1), (2) below. When a program is executed after completion of the following steps, trace data is sampled.

(1) Enable the trace function.

(2) Perform trace buffer full break setup.

■ Setting Trace

To perform a trace, complete the following setup steps. When a program is executed after completion of the steps, trace data is sampled.

1) Enable the trace function.

Enable the trace function using the ENABLE TRACE command.

To disable the trace function, use the DISABLE TRACE command.

The trace function is enabled by default when the program is launched.

2) Perform trace buffer full break setup.

A break can be invoked when the trace buffer becomes full.

To perform setup, use the SET TRACE command. This break feature is disabled when the program starts. To view the setting, use SHOW TRACE/STATUS.

Table 2.4-4 shows the commands related to a trace.

Table 2.4-4 Trace-related Commands

Available command	Function
SET TRACE	Sets trace buffer full break
SHOW TRACE	Displays trace data
SEARCH TRACE	Searches for trace data
ENABLE TRACE	Enables trace function
DISABLE TRACE	Disables trace function
CLEAR TRACE	Clears trace function

2.4.4.2 Displaying Trace Data Storage Status

It is possible to displays how much trace data is stored in the trace buffer. This status data can be read by specifying /STATUS to the SHOW TRACE command.

■ Displaying Trace Data Storage Status

It is possible to displays how much trace data is stored in the trace buffer. This status data can be read by specifying /STATUS to the SHOW TRACE.

[Example]

```
>SHOW TRACE/STATUS
en/dis      = enable                 Trace function enabled
buffer full = nobreak                Buffer full break function disabled
sampling    = end                    Trace sampling terminates
flame no. = -00120 to  00050         Frame -120 to 50 store data
step no.  = -00091 to  00022         Step -91 to 22 store data
>
```

2.4.4.3 Specifying Displaying Trace Data Start

The data display start position in the trace buffer can be specified by inputting a step number or frame number using the SHOW TRACE command. The data display range can also be specified.

■ Specifying Displaying Trace Data Start

Specify the data display start position in the trace buffer by inputting a step number or frame number using the SHOW TRACE command. The data display range can also be specified.

[Example]

- In Single Trace Mode

```
>SHOW TRACE/CYCLE -6          Start displaying from frame -6
>SHOW TRACE/CYCLE -6..10      Display from frame -6 to frame 10
>SHOW TRACE -6                Start displaying from step -6
>SHOW TRACE -6..10            Displays from step -6 to step 10
```

2.4.4.4 Display Format of Trace Data

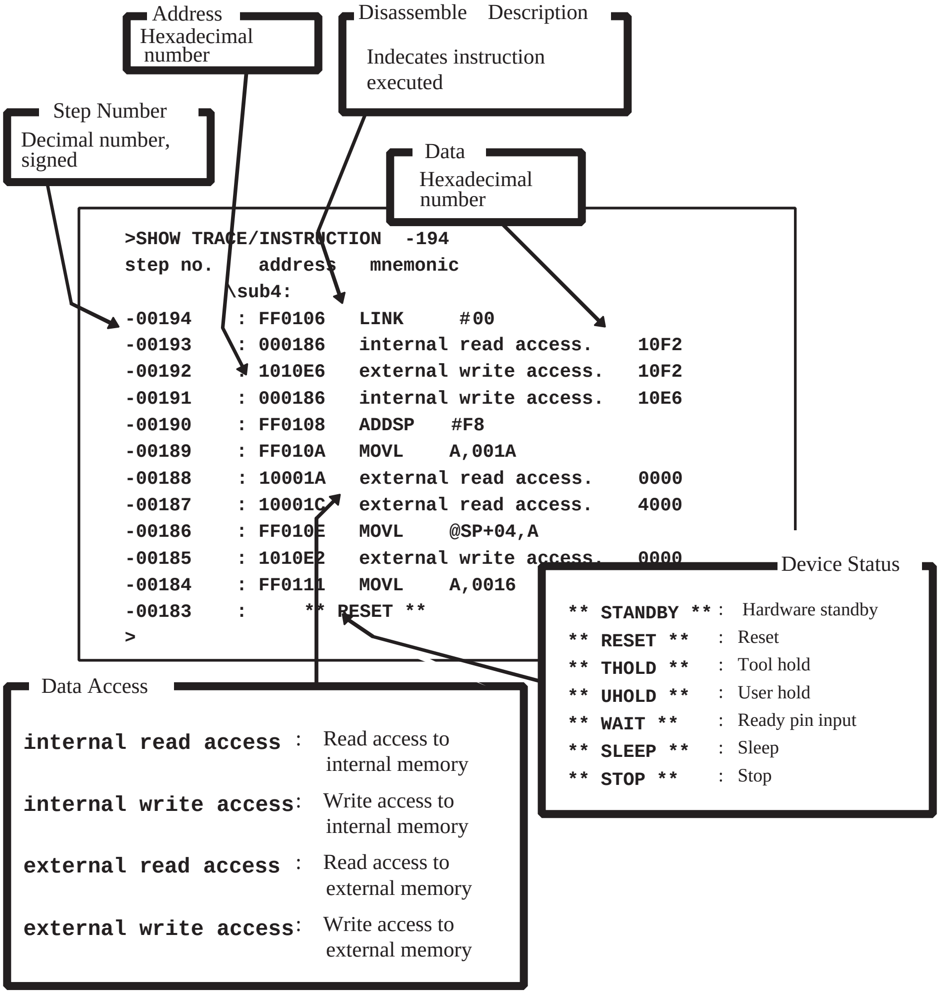
The trace data display format can be selected by running the SHOW TRACE command with a command modifier specified. If setup is completed with the SET SOURCE command so as to select a source line addition mode, a source line is attached to the displayed trace data.
There are three formats to display trace data:

- **Display in instruction execution order (Specify /INSTRUCTION.)**
- **Display all machine cycles (Specify /CYCLE.)**
- **Display in source line units (Specify /SOURCE.)**

■ Display in Instruction Execution Order (Specify /INSTRUCTION.)

Trace sampling is performed at each machine cycle, but the sampling results are difficult to display because they are influenced by pre-fetch, etc. This is why the emulator has a function to allow it to analyze trace data as much as possible. The resultant data is displayed after processes such as eliminating pre-fetch effects, analyzing execution instructions, and sorting in instruction execution order are performed automatically. However, this function can be specified only in the single trace while in the debugging mode.

In this mode, data can be displayed in the following format.



```
>SHOW TRACE/INSTRUCTION  -194
step no.    address   mnemonic
           \sub4:
-00194    : FF0106   LINK     #00
-00193    : 000186   internal read access.    10F2
-00192    : 1010E6   external write access.   10F2
-00191    : 000186   internal write access.   10E6
-00190    : FF0108   ADDSP    #F8
-00189    : FF010A   MOVL     A,001A
-00188    : 10001A   external read access.    0000
-00187    : 10001C   external read access.    4000
-00186    : FF010E   MOVL     @SP+04,A
-00185    : 1010E2   external write access.   0000
-00184    : FF0111   MOVL     A,0016
-00183    :      ** RESET **
>
```

Address
Hexadecimal number
Disassemble Description
Indecates instruction executed
Step Number
Decimal number, signed
Data
Hexadecimal number
Device Status
** STANDBY ** : Hardware standby
** RESET ** : Reset
** THOLD ** : Tool hold
** UHOLD ** : User hold
** WAIT ** : Ready pin input
** SLEEP ** : Sleep
** STOP ** : Stop
Data Access
internal read access : Read access to internal memory
internal write access: Write access to internal memory
external read access : Read access to external memory
external write access: Write access to external memory

■ Displaying All Machine Cycles

Detailed information at all sampled machine cycles can be displayed.

In this mode, no source is displayed irrespective of the setup defined by the SET SOURCE command.

[Example]

```
>SHOW TRACE/CYCLE -587
frame no.   address    data   a-status   d-statusQst        dfg
-00587      : FF0106   0106   ---        -------  FLH
-00586      : FF0106   0008   ECF        EXECUTE---         @
-00585      : FF0106   0106   ---        EXECUTE---
-00584      : 1010E8   10E8   ---        -------  ---
-00583      : 1010E8   0102   EWA        EXECUTE---         @
-00582      : 1010E8   0102   ---        EXECUTE---
-00581      : 000186   0186   ---        -------  2by
-00580      : 000186   10F2   IRA        EXECUTE---         @
-00579      : 1010E6   10E6   ---        -------  ---
```

```
-00578     : 1010E6   10F2   EWA       EXECUTE ---      @
-00577     : 1010E6   10F2   ---       EXECUTE---
-00576     : 000186   0186   ---       -------  ---
```

How to read trace data

```
frame no.  address  data  a-status  d-status  Qst  dfg
   (1)        (2)    (3)     (4)       (5)     (6)  (7)
```

(1):frame number (Decimal, signed)

(2):executed instruction address, and data access address (Hexadecimal number)

(3):data (Hexadecimal number)

(4):access information (a-status)

IWA:write access to internal memory

EWA:write access to external memory

IRA:read access to internal memory

ERA:read access to external memory

ICF:code fetch to internal memory

ECF:code fetch to external memory

---:valid "d-status" information

(5):device information (d-status)

STANDBY:hardware standby

THOLD :tool hold

UHOLD :user hold

WAIT   :ready pin

SLEEP :sleep

STOP   :stop

EXECUTE:execute instruction

RESET :reset

-------:invalid d-status information

(6):instruction queue status

FLH:flush queue

-by:number of remainder code of queue is -byte(-:1 to 8)

(7):valid flag

&:this frame address is valid

@:this frame data is valid

■ Display in Source Line Units (Specify /SOURCE.)

Only the source line can be displayed. This mode is enabled only in the debugging mode.

[Example]

```
>SHOW TRACE/SOURCE -194
step no.  source
-00194:      gtg1.c$251 {
-00190:      gtg1.c$255           sub5(nf, nd);
-00168:      gtg1.c$259 {
-00164:      gtg1.c$264           p = (char *)  &df;
-00161:      gtg1.c$264           p = (char *)  &df;
-00157:      gtg1.c$265           *(p++) = 0x00;
-00145:      gtg1.c$266           *(p++) = 0x00;
-00133:      gtg1.c$267           *(p++) = 0x80;
-00121:      gtg1.c$268           *p     = 0x7f;
-00116:      gtg1.c$270           p = (char *)  &dd;
-00111:      gtg1.c$271           *(p++) = 0xff;
-00099:      gtg1.c$272           *(p++) = 0xff;
```

Note:

The following operation may be subjected to trace sampling immediately after the MCU operation is stopped (tool hold). Remember that the operation is unique to evaluation chips and not performed by mass-produced products.

Access to address 0x000100 and addresses between 0x0FFFFDC and 0x0FFFFFF

2.4.4.5 Reading Trace Data On-the-fly

Trace data can be read while executing a program. However, this is not possible during sampling. Disable the trace function or terminate tracing before attempting to read trace data.

■ Reading Trace Data On-the-fly

To disable the trace function, use the DISABLE TRACE command. Check whether or not the trace function is currently enabled by executing the SHOW TRACE command with /STATUS specified, or by using the built-in variable, %TRCSTAT.

Tracing terminates when the sequencer has terminated. If Not Break is specified here, tracing terminates without a break operation. It is possible to check whether or not tracing has terminated by executing the SHOW TRACE command with /STATUS specified, or by using the built-in variable, %TRCSAMP.

To read trace data, use the SHOW TRACE command; to search trace data, use the SEARCH TRACE command.

[Example]

```
>GO
>>SHOW TRACE/STATUS
en/dis        = enable
buffer full   = nobreak
sampling      = on                     <- Trace sampling continues.
>>SHOW TRACE/STATUS
en/dis        = enable
buffer full   = nobreak
sampling      = end                    <- Trace sampling ends.
frame no.     = -00805 to  00000
step no.      = -00262 to  00000
>>SHOW TRACE -52
step no.      address mnemonic                            level
\sub5:
-00052        : FF0125    LINK        #02                      1
-00051        : 000186    internal    read access.      10E6   1
-00050        : 1010D6    external    write access.     10E6   1
-00049        : 000186    internal    write access.     10D6   1
                       .                       .          .
```

If the CLEAR TRACE command is executed with the trace ending state, trace data sampling can be re-executed by re-executing the sequencer from the beginning.

2.4.4.6 Saving Trace Data

The debugger has function of saving trace data to a file.

■ Saving Trace Data

Save the trace data to the specified file.

For details on operations, refer to Sections 3.14 Trace Window, and 4.4.8 Trace in the SOFTUNE Workbench Operation Manual; and Section 4.23 SHOW TRACE in the SOFTUNE Workbench Command Reference Manual.

2.4.5 Execution Cycle Measurement by Cycle Counter

The counter for measuring the execution cycles is called a cycle counter. It is possible to measure the number of cycles between the start and stop of MCU execution using this counter.

■ Execution Cycle Measurement by Cycle Counter

The cycle counter has 56 significant bits and permits measurement of up to 72,057,594,037,927,935 cycles.

The display shows the results of two cycles count measurements: the cycle count of the last program execution, and the total cycle count of all completed program executions. If the timer overflows, it is notified by a display. Measurement is performed at each program execution. The emulation timer cannot be disabled. However, the cycle values can be cleared.

Use the following commands for control:

- SOW TIMER : Indicates measured cycle count
- CLEAR TIMER : Clears measurement results

[Example]

```
>GO  main,$25
Break at FF008D by breakpoint
>SHOW TIMER
                            cycle
from init           =       4210826410
from last executed  =       362387415
>CLEAR TIMER
>SHOW TIMER
                            cycle
from init           =
from last executed  =
>
```

Note:

Note that the measured number of execution cycles is added about ten extra cycles per execution.

2.5 Monitor Debugger

This section describes the functions of the monitor debugger for the F^{2}MC-16 family.

■ Monitor Debugger

The monitor debugger performs debugging by putting the target monitor program for debugging into the target system and by communicating with the host.

Before using this debugger, the target monitor program must be ported to the target hardware.

2.5.1 Resources Used by Monitor Program

The monitor program of the monitor debugger uses the I/O resources listed below. The target hardware must have these resources available for the monitor program.

■ Required Resources

The following resources are required to build the monitor program into the target hardware.

1	UART	Necessary	For communication with host computer 4800/9600/19200/38400 bps
2	Monitor ROM	Necessary	Need about 10 KB (For details, refer to link map.)
3	Work RAM	Necessary	Need about 2 KB (For details, refer to link map.)
4	External-interrupt switch	Option	Uses for forced abortion of program. When the resource is not built, the program can suspend by only reset etc.
5	Timer	Option	Uses for SET TIMER/SHOW TIMER . Needs 32 bits in 1 µs units.

2.6 Abortion of Program Execution (SIM, EML, MON)

When program execution is aborted, the address where the break occurred and the break source are displayed.

■ Abortion of Program Execution

When program execution is aborted, the address where the break occurred and the break source are displayed.

In the emulator debugger, the following sources can abort program execution.

- Instruction Execution Breaks
- Data Access Breaks
- Sequential Break
- Guarded Access Breaks
- Trace-Buffer-Full Break
- Performance-Buffer-Full Break
- Task Dispatch Break
- System Call Break
- Forced Break

In the simulator debugger, the following sources can abort program execution.

- Instruction Execution Breaks
- Data Access Breaks
- Guarded Access Breaks
- Task Dispatch Break
- System Call Break
- Forced Break

In the monitor debugger, the following sources can abort program execution.

- Software Break
- Task Dispatch Break
- System Call Break
- Forced Break

2.6.1 Instruction Execution Breaks (SIM, EML)

An instruction execution break is a function to let an instruction break through monitoring bus, the chip built-in break points, etc.

■ Instruction Execution Breaks

An instruction execution break is a function to let an instruction break through monitoring bus, the chip built-in break points, etc.

Use the following commands to control instruction execution breaks.

SET BREAK:	Sets break points
SHOW BREAK:	Displays current break point setup status
CANCEL BREAK:	Cancels break point
ENABLE BREAK:	Enables break point
DISABLE BREAK:	Temporarily cancels break point

When a break occurs due to an instruction execution break, the following message is displayed.

Break at Address by breakpoint

The maximum count of break points are as follows:

[SIM]	Max 65535 points
[EML]	Within debugging area of Code attribute: 65535 points
	Areas other than above: 6 points

Note:

In the emulator, if the debug area is set again, the break points within the area are all cleared.

■ Notes on Instruction Execution Breaks

There are several points to note in using execution breaks. First, some points affecting execution breaks are explained.

- **Invalid Breakpoints**
  - No break occurs when a break point is set at the instruction immediately after the following instructions.

    F^{2}MC-16/16L/16LX/16H: - PCB - DTB - NCC - ADB - SPB - CNR
    - MOV ILM,#imm8 - AND CCR,#imm8
    - OR CCR,#imm8 - POPW PS

    F^{2}MC-16F: - PCB - DTB - NCC - ADB - SPB - CNR
  - No break occurs when break point set at address other than starting address of instruction.
  - No break occurs when both following conditions met at one time.
    - Instruction for which break point set starts from odd-address,
    - Preceding instruction longer than 2 bytes length, and break point already set at last 1-byte

address of preceding instruction (This "already-set" break point is an invalid break point that won't break, because it has been set at an address other than the starting address of an instruction).

- **Abnormal Break Point**

  Setting a break point at the instruction immediately after string instructions listed below, may cause a break in the middle of the string instruction without executing the instruction to the end.

  F^{2}MC-16L/16LX/16/16H: - MOVS - MOVSW - SECQ - SECQW - WBTS - MOVSI
  - MOVSWI - SECQI - SECQWI - WBTC - MOVSD
  - MOVSWD - SECQD - SECQWD - FILS - FILSI - FILSW
  - FILSWI

  F^{2}MC-16F: Above plus - MOVM- MOVMW

Here are some additional points about the effects on other commands.

- **Dangerous Break Points**

  Never set a break point at an address other than the instruction starting address. If a break point is the last 1 byte of an instruction longer than 2 bytes length, and if such an address is even, the following abnormal operation will result:

  - If instruction executed by STEP command, instruction execution not aborted.
  - If break point specified with GO command, set at instruction immediately after such instruction, the break point does not break.

Note:

In order to set breakpoints, it is required to set memory map to high-speed simulator debugger.
When the memory map defined area is changed to an undefined attribute, the breakpoints are cancelled.

2.6.2 Monitoring Data Breaks (MB2147-01 EML)

The monitoring data break is a function to halt the program execution, when the data in the address of the specified data area is accessed and the program execution reaches the specified code address.

■ Monitoring Data Breaks

The monitoring data break is a function to halt the program execution, when the specified data address is accessed by MCU and the program execution reaches the specified code address (However, detect it before the instruction is executed).

Monitoring data breaks can be controlled using the following commands:

SET BREAK/DATAWATCH:	Sets monitoring data break
SHOW BREAK/DATAWATCH:	Displays current monitoring data break setup status
CANCEL BREAK/DATAWATCH:	Cancels monitoring data break
ENABLE BREAK/DATAWATCH:	Enables monitoring data break
DISABLE BREAK/DATAWATCH:	Temporarily cancels monitoring data break

When a break occurs due to a data access break, the following message is displayed:

Break at Address by datawatchbreak

The maximum count of break points are as follows.

MB2147-01 emulator Max: 8 points

Current data monitoring break maximum constant
= 8 - (current event constant + current trace trigger constant)

Note:

Monitoring data breaks is specific function in the MB2147-01 emulator.

2.6.3 Data Access Breaks (SIM, EML)

A data access break is a function to abort a running program when data access (Read or Write) is made to the specified address while the program is executing.

■ Data Access Breaks

A data access break is a function to abort a executing program when the MCU accesses data at the specified address.

Data access breaks can be controlled using the following commands:

SET DATABREAK:	Sets break point
SHOW DATABREAK:	Displays current break point setup status
CANCEL DATABREAK:	Cancels break point
ENABLE DATABREAK:	Enables break point
DISABLE DATABREAK:	Temporarily cancels break point

When a break occurs due to a data access break, the following message is displayed:

Break at Address by databreak at Access address

The maximum count of break points are as follows.

[SIM]	Max 65535 points
[EML]	Within debug area of Data attribute: 65535 points
	Areas other than above: 6 points

Note:

1. In the emulator debugger, if the debug area is set up again, the break points in the area are all cleared.
2. In order to set breakpoints, it is required to set memory map to high-speed simulator debugger. When the memory map defined area is changed to an undefined attribute, the breakpoints are cancelled.

2.6.4 Software Break (MON)

A software break is a function to embed a break instruction within memory to enable a break to occur by executing the instruction. The break occurs before executing the instruction at the specified address.

■ Software Break

Up to 16 software break points can be set.

Software breaks can be controlled using the following settings and commands.

- [Debug]-[Breakpoints] command
- Setting break points in Source window
- Setting break points in Disassemble window
- SET BREAK/SOFT command

When a break occurs due to a software break, the following message is displayed on the status bar:

Break at Address breakpoint

■ Notes on Software Breaks

There are a couple of points to note when using software breaks.

- Software breaks cannot be set in an area that cannot be written, such as ROM. If attempted, a verify error occurs at starting the program (when continuous execution, step execution, etc., started).
- Always set a software break at the instruction starting address. If a software break is set in the middle of an instruction, it may cause a program null-function.

2.6.5 Sequential Break (EML)

A sequential break is a function to abort a executing program, when the sequential condition is met by event sequential control.

■ Sequential Break

Use a sequential break when the event mode is set to normal mode using the SET MODE command. Set a sequential break as follows:

- Set event mode (SET MODE).
- Set events (SET EVENT).
- Set sequencer (SET SEQUENCE).

When a break occurs due to a sequential break, the following message is displayed:

Break at Address by sequential break (level = Level No.)

2.6.6 Guarded Access Breaks (SIM, EML)

A guarded access break aborts a executing program when access is made in violation of the access attribute set by using the [Setup]-[Memory Map] command, and access is attempted to a guarded area (access-disabled area in undefined area).

■ Guarded Access Breaks

Guarded access breaks are as follows:

- Code Guarded

  An instruction has been executed for an area having no code attribute.

- Read Guarded

  A read has been attempted from the area having no read attribute.

- Write Guarded

  A write has been attempted to an area having no write attribute.

If a guarded access occurs while executing a program, the following message is displayed on the Status Bar and the program is aborted.

Break at Address by guarded access {code/read/write} at Access address

■ Notes on Using Emulator

Code Guarded is affected by pre-fetching.

The F^2MC-16L/16LX/16/16H family pre-fetch up to 4 bytes. So, when setting the program area mapping, set a little larger area (5 bytes max.) than the program area actually used.

Similarly, the F^2MC-16F family pre-fetch up to 8 bytes. So, when setting the program area mapping, set a little larger area (9 bytes max.) than the program area actually used.

2.6.7 Trace-Buffer-Full Break (SIM, EML)

A trace-buffer-full break occurs when the trace buffer becomes full.

■ Trace-Buffer-Full Break

To set a trace-buffer-full break, use the [View]-[Trace] command in the short-cut menu of the [Set]-[Trace] command, or use the SET TRACE/BREAK command.

When a break occurs due to a trace-buffer-full break, the following message is displayed:

```
Break at  Address  by trace buffer full
```

2.6.8 Performance-Buffer-Full Break (EML)

A performance-buffer-full break is a function to abort an executing program when the buffer for storing performance measurement data becomes full.

■ Performance-Buffer-Full Break

To set a performance-buffer-full break, use the SET PERFORMANCE command. If a performance-buffer-full break is not specified, no break occurs even when the performance buffer becomes full.

When a break occurs due to a performance-buffer-full break, the following message is displayed:

```
Break at  Address  by performance buffer full
```

2.6.9 Task Dispatch Break (SIM, EML, MON)

A task dispatch break is a break that occurs when a dispatch is made from the specified dispatch source task to the dispatch destination task. In other words, the break occurs when the dispatch destination task becomes the execution state. If the dispatch destination task is currently in the execution state, then the break occurs when the task enters the execution state again via another state.

■ Task Dispatch Break

Only one break point can be set.

To use this function, the REALOS Debug Module must be embedded. For further details, see Operation Manual Appendix F Embedding the REALOS Debug Module.

To control the task dispatch break, use either of the following commands.

- [Debug]-[Break Points]-[Task Dispatch] command
- SET XBREAK command

When a break occurs due to a task dispatch break, the following message is displayed on the Status Bar.

Break at Address by dispatch task from task ID= <Dispatch source task ID> to task ID= <Dispatch destination task ID >

2.6.10 System Call Break (SIM, EML, MON)

A system call break occurs at ending execution of a system call specified by the task specified.

■ System Call Break

Only one break point can be set.

To use this function, the REALOS Debug Module must be embedded. For further details, see Operation Manual Appendix F Embedding the REALOS Debug Module.

To control the system call break, use either of the following commands.

- [Debug]-[Break Points]-[System Call] command
- Set Sbreak command

When a break occurs due to a system call break, the following message is displayed on the Status Bar.

```
Break at  Address  by system call <System call> on task ID= <Task ID>
```

2.6.11 Forced Break (SIM, EML)

A executing program can be forcibly aborted by using the [Debug]-[Abort] command. In the monitor debugger, the same result can be achieved by letting the target generate NMI.

■ Forced Break

When a break occurs due to a forced break, the following message is displayed on the Status Bar.

Break at Address by command abort request

■ Forced Break in Power-Save Mode and Hold State

A forced break is not allowed in the emulator while the MCU is in the power-save consumption mode or hold state. When a forced break is requested by the [Debug]-[Abort] command while executing a program, the command is disregarded if the MCU is in the power-save consumption mode or hold state. If a break must occur, then reset the cause at user system side, or reset the cause by using the [Debug]-[Reset MCU] command, after inputting the [Debug]-[Abort] command.

When the MCU enters the power-save consumption mode or hold state while executing, the status is displayed on the Status Bar.

CM41-00313-3E

FUJITSU SEMICONDUCTOR • CONTROLLER MANUAL

F^{2}MC-16 FAMILY

SOFTUNE WORKBENCH

USER’S MANUAL

January 2007 the third edition

Published **FUJITSU LIMITED** Electronic Devices
Edited Business Promotion Dept.