FUJITSU SEMICONDUCTOR
CONTROLLER MANUAL

CM81-00306-2E

FR FAMILY F²MC FAMILY

32/16/8-BIT MICROCONTROLLER

SOFTUNE Workbench USER'S MANUAL

FUJITSU

FR FAMILY F^{2}MC FAMILY

32/16/8-BIT MICROCONTROLLER

SOFTUNE Workbench

USER'S MANUAL

FUJITSU LIMITED

1. The contents of this document are subject to change without notice. Customers are advised to consult with FUJITSU sales representatives before ordering.
2. The information and circuit diagrams in this document are presented as examples of semiconductor device applications, and are not intended to be incorporated in devices for actual use. Also, FUJITSU is unable to assume responsibility for infringement of any patent rights or other rights of third parties arising from the use of this information or circuit diagrams.
3. The contents of this document may not be reproduced or copied without the permission of FUJITSU LIMITED.
4. FUJITSU semiconductor devices are intended for use in standard applications (computers, office automation and other office equipments, industrial, communications, and measurement equipments, personal or household devices, etc.).
   CAUTION:
   Customers considering the use of our products in special applications where failure or abnormal operation may directly affect human lives or cause physical injury or property damage, or where extremely high levels of reliability are demanded (such as aerospace systems, atomic energy controls, sea floor repeaters, vehicle operating controls, medical devices for life support, etc.) are requested to consult with FUJITSU sales representatives before such use. The company will not be responsible for damages arising from such use without prior approval.
5. Any semiconductor devices have inherently a certain rate 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.
6. If any products described in this document represent goods or technologies subject to certain restrictions on export under the Foreign Exchange and Foreign Trade Control Law of Japan, the prior authorization by Japanese government should be required for export of those products from Japan.

PREFACE

What is SOFTUNE WORKBENCH?

SOFTUNE WORKBENCH is a software program supporting a global development environment for Fujitsu microcontrollers (FR Family, FFMC-16 Family and FF MC-8L Family).

Construction

This manual contains the following chapters.

Chapter 1 MCU Common Functions
This chapter describes the functions common to all MCUs.

Chapter 2 FR Family
This chapter describes the functions dependent on the FR family dependent functions.

Chapter 3 FFMC-16 Family
This chapter describes the functions dependent on the FFMC-16 family dependent functions.

Chapter 4 FFMC-8L Family
This chapter describes the functions dependent the FFMC-8L family dependent functions.

Table of Contents

Chapter 1 MCU Common Functions

This chapter describes the functions common to the FR, FFMC-16, and FFMC-8L families.

1.1 Project Management Function

This section describes the project management function in SOFTUNE WORKBENCH.

Project

SOFTUNE WORKBENCH processes and manages jobs in projects. A project has all data such as the files and procedures required for generating a target file. All the data managed by the project is stored in the project file.

Project Management Function

The project manages all information for developing a micro-controller system, particularly, the most important function of the project is managing the information required to generate a target file.

The project manages the following information:

- Target file name and directory
- Information on constituent source files, include files, other object files, library files
- Information on options for language tools used for compiling/assembling and linking source files when generating target file
- Debugger setup information required to debug target file

1.2 Make/Build Function

This section describes the SOFTUNE WORKBENCH Make/Build function.

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. For further details, see ***Section 1.3 Include Dependencies Analysis Function***.

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.

1.3 Include Dependencies Analysis Function

This section describes the function of the Include Dependencies Analysis.

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]*** command. 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.

1.4 Functions for Setting Tool Options

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

Setting Tool Options

Options for language tools such as a compiler, assembler and linker, must be defined to use these tools to generate the required target file. In SOFTUNE WORKBENCH, the options for each tool are registered and managed with a project.
There are two types of option settings: setting options valid for all source files, and setting options valid for a specific source file (Individual Option Setup).

- **Setting options valid for all source files**
  Options that are set using the ***[Project] - [Setup Tool Option]*** command are valid for all source files registered with a project.

- **Individual Option Setup**
  The valid compile/assemble options can be set only for a specific source file by right-clicking a registered source file name in the Project window to view the short-cut menu, and then selecting the ***[Setup Tool Option] - [Individual Option Setup]*** command.

Tool Options

For further details on options available for each tool, please refer to the manual for each tool.

Reference Section

Setup Tool Option
Individual Option Setup
Development Environment

1.5 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 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 registed. To set Error Jump, execute the ***[Setup] - [Error]*** command.

Syntax

An error message format can be described in **Syntax**. SOFTUNE WORKBENCH uses macro descriptions as shown in the ***Table 1-5-1*** 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, 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 to be skipped, describe it as **"\]"**. Blank characters in succession can be specified with a single blank character.

Table 1-5-1 Special Characters for Analyzing Error Messages

Characters	Semantics
%f	Interpret as source file name and inform editor.
%1	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

1.6 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/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.
- **Tag setup function**
  A tag can be set on any line, and instantaneously jumps to the line. Once a tag 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 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, Line Feed code, and Tab code**
  When a file includes a 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 characters displayed in the Edit window can be selected.

Reference section

Edit Window (The Standard Editor)

1.7 Setting External Editor

This section describes the function to set an external editor in 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. There is no particular limit on which editor can be set, but some precautions (below) may be necessary. Use the ***[Setup] - [Editor]*** command 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.

- **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 1-7-1***. If any other character are described by these parameters, such character are passed as is to the editor.

%f (File name) is determined as follows:

(1) If the focus is on the 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.
Also filenames 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 1-7-1 Parameters Used in Option Setups (For External Editors)

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

Reference Section

Editor Setup

Example of Optional Settings

Examples. Editor name : Argument

(A) WZ Editor V4.0 : %f --j%l

(B) MIFES V1.0 : %f /j%l

(C) UltraEdit32 : %f/%l/1

(D) TextPad32 : %f(%l)

(E) PowerEDITOR : %f -g%l

(F) Codewright32 : %f -g%l

(G) Hidemaru for Win3.1/95 : /j%l:1 %f

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".

1.8 Setting External Tool

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

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 Microsoft's ***Visual SourceSafe***, a source file version control tool. For further details on coordination with ***Visual SourceSafe***, see "Coordination with source file version control tools".

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 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] - [Tool]*** command.

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

Setting Options

When activating an external tool from SOFTUNE WORKBENCH, options must be added immediately after the 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 ***(Table 1-8-1)***.

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

%f (File name) is determined as follows:

(1) If the focus is on the 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 having a focus in the built-in editor becomes the file name.

Precautions

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

- Once a tool is activated, neither other tools nor the compiler/assembler can be 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**.

Table 1-8-1 Parameters in Option Setups (External Tools)

Parameter	Semantics
%f	Means file name
%F	Means main file name of file
%d	Means file path
%e	Means file extension
%a	Means target file name of Project
%A	Means main file name of target file name of Project
%D	Means path of target file of Project
%E	Means extension of Project target file
%x	Means project path.
%X	Means Project main file name
%%	Means % itself

[Example] Macro Expansion Example

If the target file name is c: **Sample\target.abs**, macro expanded as follows:

```
%a:  c:\Sample\target.abs
%A:  target.abs
%D:  c:\Sample\
%E:  .abs
```

Reference Section

Setting Tools

Start an External Tool

1.9 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 items for the Project.
To set the operating environment, use the **[Setup]-[Development]** command.

- **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.

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]***.

1.10 Debugger Types

This section describes the functions of SOFTUNE WORKBENCH debuggers.

Debug Function

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 In-Circuit Emulator (ICE) from a host through a communications line (RS-232C, LAN).
Before using this debugger, the ICE 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.

1.11 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 the ***Operation Manual 3.11 Memory Window***.

- **Display/Modify C/C++ variables**
  The names of variables in a C/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 the ***Operation Manual 3.13 Watch Window***.

1.12 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 the ***Operation Manual 4.4.4 Register***.

The name of the register and flag 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

1.13 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 the ***Operation Manual 4.4.3 Assembly***.

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

1.14 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.
There are three types of symbols as follows:

- **Global symbol**
  A global symbol can be referred to from anywhere within a program. In C/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 can be referred to only within the module where the symbol is defined. In C/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/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/C++, in assembler for each source file to be assembled in assembler.
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/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
```

1.14.1 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 preceded 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 the ***Operation Manual 4.6.7 Stack***. 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:

- Local symbols in function in Current Scope
- The class member which can access with the this pointer
- Static symbols in module in Current Scope
- Global symbols

If a global symbol has the same name as a local symbol in the Current Scope, specify **"\"** 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.

1.14.2 Referring to C/C++ Variables

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

Specifying C/C++ Variables

C/C++ variables can be specified using the same descriptions as in the source program. The address of C/C++ variables should be preceded by the ampersand symbol **"&"**. Some examples are shown in the ***Table 1-14-1***.

Table 1-14-1 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[l]**	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 Reference type	**int data;** **ini i;**	**&data** **ri**	Address of **data** Same as i
variables class	**int &ri = i;** **class X {** **static int i;** **}cls;**	**cls.i** **X::i**	Value of member i of class X Same as cls.i
Member pointer class	**int X::i;** **class X{** **short cs;** **}clo;** **short X::* ps = &X::cs;**	**clo.*ps** **clp->*ps**	Same as clo.cs Same as clp->cs

Notes on C/C++ Symbols

The C/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 program debugging easier.

Consequently, a symbol name described in C/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 FR Family

This chapter describes functions dependent on the FR family MCUs

2.1 Simulator

This section describes the functions of the simulator for the FR Family

Simulator Debugger

The simulator debugger (later referred as simulator) 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.

Simulation Range

The simulator simulates the MCU operations (instruction operations, memory space, I/O ports, interrupts, reset, low power-save mode, etc.) using software to execute operations. It does not support built-in resources and related registers not described in the manual.

- 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 FR Family. 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 must first secure memory space to simulate instructions because it simulates the memory space secured in the host machine memory.

- 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

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 Inport`** command in the Command window.

Up to 16 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 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 16 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.

2.1.4 Interrupt Simulation

This section describes interrupt simulation executed by SOFTUNE WORKBENCH.

Interrupt Simulation

This simulates the MCU operation for an interrupt request. The correspondence between the cause of each interrupt and the interrupt control register is made by referring to the install file read at simulator start-up.
The following types can be used to allow an interrupt to occur.

- When the instruction is executed as many cycles as the specified cycle count while executing a program (executing execution commands), generate an interrupt corresponding to the specified interrupt number to reset the interrupt generating condition.
- Whenever the instruction executing cycle count exceeds the specified cycle, an interrupt continues to be generated

The type of interrupt can be set using either the ***[Setup] - [Debug Environment] - [Interrupt]*** command, or the **`Set Interrupt`** command in the Command window. If an interrupt is masked by an interrupt-enabled flag when the interrupt generating condition is met, the interrupt is generated after resetting the mask. When an interrupt is generated while executing a program, an interrupt cause number is displayed on the Status Bar.

Furthermore, the simulator supports the MCU operation for interrupt requests for the following exception processing.

- Executing undefined instruction

2.1.5 Reset Simulation

This section describes the reset simulation executed by SOFTUNE WORKBENCH.

Reset Simulation

The simulator simulates the MCU operation when a reset signal is input to the MCU by using either the ***[Debug]*** - ***[Reset of MCU]*** command, or the **`Reset`** command in the Command window. This initializes registers.

2.1.6 Power-Save Consumption Mode Simulation

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

Power-Save Consumption Mode Simulation

The MCU enters the power 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 ***[Run] - [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.

2.2 Emulator

This section describes the functions of the emulator for the FR family.

Emulator Debugger

The emulator debugger (lates referred as emulator) is software to evaluate a program by controlling an ICE from a host via a communications line (RS-232C, LAN).
Before using this emulator, the ICE must be initialized.
For further details, refer to the ***Operation Manual Appendix B Download Monitor Program***, and ***Appendix C Setting LAN Interface***.

2.2.1 Setting Operating Environment

This section describes how to set the emulator operating environment for the FR family.

Setting Operating Environment

Before using the emulator, the operating environment for the MCU operation mode, DRAM refresh control and cache flush control must be set. Each setting has a start default, so the operating environment does not require setting when using the defaults. In addition, specified values can be used as defaults.

- MCU operation mode
- DRAM refresh control
- Cache flash control

2.2.1.1 MCU Operation Mode

The following four modes are in the MCU Operation Mode. The Internal Trace Mode and External Trace Mode are enabled only with products using the DSU3 chips.

- Full Trace Mode
- Full Real Time Mode
- Internal Trace Mode
- External Trace Mode

Setting MCU Operation Mode

Set the MCU operation mode. There are two modes: full trace, and full real-time. To set the operation mode, use either the ***[Setup] - [Debug Environment] - [Debug Environment]*** command, or the **`Set RunMode`** command in the Command window.

Full Trace Mode

In the full trace mode, all instruction executions can be traced without omission. However, if branching occurs more than three times within 11 cycles, operations may not be real-time due to the wait entered to MCU as acquiring the trace data is preceded.

Full Real-time Mode

In the full real-time mode, a program runs in real-time. However, if branching occurs more than three times within 11 cycles, some trace data may be omitted.
DSU3 chips may cause an error at cycle count measurement. When measuring the cycle count, use the internal or external trace mode.

Internal Trace Mode

Trace data is stored in the specialized trace memory built-in to the chip. The program is executed at real time, but this is possible only with DSU3 chips which include that function.

External Trace Mode

Trace data is stored in the specialized trace memory mounted on the adapter board. The program is executed at real time, but this is possible only with DSU3 chips which include that function.

2.2.1.2 DRAM Refresh Control

This section describes setting the DRAM refresh in the emulator for the FR family.

DRAM Refresh Control

The operating frequency of some DSU chips is automatically divided at a break (in emulation mode). When this happens, the register (RFCR) must be reset if the built-in DRAM refresh function is used on the user target.

The **RFCR** register values for On Execution (in user mode) and On Break (in emulation mode) can be set by using the ***[Setup]*** - ***[Debug Environment]*** - ***[Debug Environment]*** - ***[RFCR]*** command. When the mode is switched, the values set here are used to set to the **RFCR** register.

<Note>

When using chips with an operating frequency that is not divided automatically at a break (in emulation mode), or when the built-in DRAM refresh function at the user target is not in use, this function causes a slowdown in debugger operation due to writing to the **RFCR** register.

2.2.1.3 Cache Flash Control

This section describes setting the Cache Flash in the emulator for the FR family.

Cache Flash Control

When using a chip with cache memory, rewriting the memory and software break point setup using commands is not reflected in the cache. Therefore, cache flashing must be performed when such commands are executed. The debugger has a function to flush the cache automatically, monitor memory rewriting, and set software break points, etc.

This function is controlled using the ***[Setup] - [Debug Environment] - [Debug Environment] - [Emulation]*** command.

<Note>

When the automatic cache flashing option is enabled, it may negatively affect the program speed.

2.2.1.4 Auto-wait Control

This section explains the settings of the auto-wait function in the DSU3 chip for emulator of the FR family.

Auto-wait Control

Wait has been added to the DSU3 chip for when the program is stopped. This is an auto-wait function for allowing access from the emulator even on chips that are operating at high speeds. You can set the auto-wait count using this debugger.

This function is set using the ***[Environment] - [Debug Environment Setting] - [Debug Environment] - [Auto-wait]*** commands.

2.2.2 Notes on Executing Program

There are several points to note about program execution commands in the emulator for the FR family.

Real-Time Functionality in Running Program

When the MCU is in the full trace mode, there are some cases when a program cannot execute in real-time.
The MCU operation mode can be set up by using either the **[*Setup] - [Debug Environment] - [Emulation]*** command, or the **Set Runmode** command in the Command window.

Notes on Delayed Branch Instruction when executed using *[Run] - [Step In]* or *[Run] - [Step Over]* command

If a delay branch instruction is executed by the ***[Run] - [Step In]*** command or ***[Run] - [Step Over]*** command, the program runs past the instruction at the delay slot (instruction immediately after delay branch instruction) and breaks immediately after executing the delay branch instruction.

Restrictions when Suspended by Software Break (EML, MON)

When there is a software break at the current PC location, if either the ***[Run] - [Go]*** command or the **Go** command is executed, the emulator performs one execution step internally, and then executes the program in batch processing. In addition, when a software break is set for the instruction to clear the T-flag, and when either the ***[Run] - [Go]*** command or the Go command is executed from that address, all software breaks are disregarded. When this happens, any interrupt is masked too.

Value of TBR Register

Note a program null-function may occur if you specify such value for the **TBR** register as the vector table overlaps to the I/O area.

Notes on Instruction to Clear T-Flag when Executed using *[Run] - [Step In]* or *[Run] - [Step Over]* command (EML, RDB)

If an instruction to clear the T-flag is executed using either the ***[Run] - [Step In]*** command, or ***[Run] - [Step Over]*** command, the program will be executed in batch processing. When this happens, all software breaks are ignored.

2.2.3 Command Execution while Executing Program

This section describes command execution while executing a program in the emulator for the FR family.

Command Execution while Executing Program

When executing a program using the ***[Run] - [Go]*** command, the Status Bar displays "**`Execute`**" to show that the program is running.
Certain commands can be executed in this circumstances (they vary with the type of debugger). The execution procedures are the same, but some commands cannot be executed and some can be executed but are subject to restrictions.
To forcibly terminate the program, use the ***[Run] - [Abort]*** command.

Note that in the emulator debugger, memory Read/Write is re-executed while executing a program after permitting the MCU to break once for access.

2.3 Monitor Debugger

This section describes the functions of the monitor debugger for the FR 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.

2.3.1 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	Required	For communication with host computer 4800/9600/19200/38400 baud
2	Monitor ROM	Required	About 6 KB required (For further details, see Link Map.)
3	Work RAM	Required	About 2 KB required (For further details, see Link Map.)
4	NMI Switch	Optional	Used for suspending program forcibly. If not implemented, forced termination only can be performed by reset, etc.
5	Timer	Optional	Used by SET TIMER/SHOW TIMER. Requires 32-bit timer in 1 us.

2.4 Suspension of Program Execution (SIM, EML, MON)

When program execution is suspended, the address where the break occurred and the break source are displayed.

Suspension of Program Execution

When program execution is suspended, the address where the break occurred and the break factor are displayed.

In the emulator debugger, the following factor can suspend program execution.

- Software Breaks
- Hardware Breaks
- Code Event Breaks
- Data Event Breaks
- Trace Buffer Full Break
- Alignment Error Break
- External Trigger Break
- Task Dispatch Break
- System Call Break
- Forced Break

<Note>

- Set the DRAM refresh control when using chips in which the operating frequency is divided automatically in the emulation mode (when execution suspended). Use the ***[Setup] - [Debug Environment] - [Debug Environment] - [RFCR]*** command for setting.
- Wait is automatically added in the emulation mode (when pausing execution) on the DSU3 chip to allow setting of the auto-wait count. Set this using the [Environment] - [Debug Environment Setting] - [Debug Environment] - [Auto-wait] commands.

In the simulator debugger, the following factors can suspend program execution.

- Break Points
- Data Break Points
- Guarded Access Breaks
- Task Dispatch Break
- System Call Break
- Forced Break

In the monitor debugger, the following factors can suspend program execution.

- Software Breaks
- Task Dispatch Break
- System Call Break
- Forced Break

<Note>

If a user program rewrites the **TBR** register, the interrupt vectors used from instruction suspension, such as a break point, step execution, forced break, etc., must be enabled. To enable the interrupt vectors, use the ***Copy Vector*** command.

2.4.1 Software Breaks (EML, MON)

A software break is a function to make a break by executing an instruction embedded in memory. The break occurs before executing the instruction at the specified address.

Software Breaks

A maximum of 8192 software break points **(EML)** can be set.
A maximum of 16 software break points **(MON)** can be set.

Software breaks can be controlled by one of the following:

- ***[Run] - [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 in the Status Bar.

`Break at` Address **`breakpoint`**

Notes on Software Breaks

There are two points to note when using software break point.

- Software breaks cannot be set in read only areas, such as ROM. If an attempt is made to do so, a verify error occurs at program startup (continuous execution in batch processing, step execution, etc.).
- Always set a software break at the instruction start address. Setting a software break point in the middle of an instruction, may cause a software error.

2.4.2 Hardware Breaks (EML)

A hardware break is a break point achieved by monitoring the chip bus using hardware. A hardware break suspends the instruction operation at the specified address immediately before executing the instruction.

Hardware Breaks

A maximum of five hardware break points can be set.

Instruction breaks can be controlled by either of the following:

- ***[Run] - [Breakpoints]*** command
- **`Set Break/Hard`** command

When a break occurs due to a hardware break, the following message is displayed on the Status Bar.

`Break at` Address **`by hardware breakpoint`**

Notes on Hardware Breaks

There are several points to note when using hardware breaks point.

- Do not set a hardware break for an instruction in a delay slot. If such a setting is made, no branching is executed when re-running after the break.
- Always include the instruction starting address in the hardware break point within the specified range. Otherwise, the break may not occur.
- When starting executing from the address where a hardware break has been set, the break occurs without executing the instruction if the preceding execution has been suspended by any cause other than an instruction break. When this happens, re-execute to execute the instruction.

2.4.3 Code Event Breaks (EML)

A code event break is a break point that makes use of break points built into an evaluation chip. Address masking, pass count and sequential can be specified.

Code Event Breaks

A maximum of two code event break points can be set.
The address, address masking, and pass count can be set for each code event. Furthermore, two-point-OR (breaks if either one is hit) and sequential (breaks if 1-->2 sequential hits occur) can be set.

To set a code event break, use either of the following:

- **[Analyze] - [Event] - [Code]** command
- **Set CodeEvent** command

When a code event break (**OR**) occurs, the following message is displayed on the Status Bar.

Break at Address **by code event break** (No. code event number)
(The number of hit event is displayed for an event number.)

When a code event break (Sequential) occurs, the following message is displayed on the Status Bar.

Break at Address **by code event break** (sequential)

<Note>

With the FR family DSU3 chips, you can use code events as the causes of breaks or as the causes of trace measurement starts. This mode is called the Trace Sampling Mode and there are two types.

- Full Mode: This uses the code event as a cause of a break.
- Trigger Mode: This uses the code event as a cause of a trace measurement start.

For this reason, set to the Full Mode so that you can use the following commands as the cause of the break.

- **[Analyze]-[Trace]** commands, **[Setup]** in the Pop-up menu.
- **Set Trace/Full** command.

2.4.4 Data Event Breaks (EML)

A data event break is a break point that makes use of break points built into an evaluation chip. Address masking, data size, access type and sequential can be specified.

Data Event Breaks

A maximum of two data event break points can be set.
The address, address masking, data size (byte, half word, word), and access type (Read/Write) can be set for each data event. Furthermore, two-point-OR (breaks if either one is hit) and sequential (breaks if 1->2 sequential hits occur) can be set.

To set data a event break, use either one of the following.

- **[Analyze] - [Event] - [Data]** command
- **Set DataEvent** command

When a data event break (OR) occurs, the following message is displayed on the Status Bar.

Break at Address **by data event break** (No. data event number)
(The number of hit event is displayed for an event number.)

When a data event break (Sequential) occurs, the following message is displayed on the Status Bar.

Break at Address **by data event break** (sequential)

<Note>

With the FR family DSU3 chips, you can use data events as the causes of breaks or as the causes of trace measurement starts. This mode is called the Trace Sampling Mode and there are two types.

- Full Mode: This uses the data event as a cause of a break.
- Trigger Mode: This uses the data event as a cause of a trace measurement start.

For this reason, set to the Full Mode so that you can use the following commands as the cause of the break.

- **[Analyze]-[Trace]** commands, **[Setup]** in the Pop-up menu.
- **Set Trace/Full** command.

2.4.5 Trace Buffer Full Break (SIM, EML)

A break occurs when the trace buffer becomes full.

Trace Buffer Full Break

A trace buffer full break can be set by using either ***[Setup] - [Trace]*** in the short-cut menu of ***[Analyze] - [Trace]*** command, or the **`Set Trace/Break`** command in the Command window.

When a break occurs due to a trace buffer full break, the following message is displayed on the Status Bar.

`Break at` Address **`by trace buffer full`**

2.4.6 Alignment Error Break (EML)

An alignment error break is a function to suspend program execution, when an instruction accesses an odd-address or word/half word accesses beyond a boundary.

Alignment Error Break

Enable/disable can be set for an alignment error break by using either the ***[Setup] - [Debug Environment] - [Emulation]*** command or the **`Enable Alignment Break`** command, or the **`Disable AlignmentBreak`** command. Enable/disable can be set for an alignment error break for each instruction and data access.

When a break occurs due to an alignment error break from an instruction access, the following message is displayed on the Status Bar.

`Break at` Address **`by alignment error break`** (code)

When a break occurs due to an alignment error break from a data access, the following message is displayed on the Status Bar.

`Break at` Address **`by alignment error break`** (data)

2.4.7 External Trigger Break (EML)

An external trigger break is a function to suspend program execution when an external signal is input to the emulator TRIG pin.

External Trigger Break

Enable/disable can be set for an external trigger break. Use either the ***[Setup] - [Debug Environment] - [Emulation]*** command, or the **`Set Trigger`** command in the Command window.

When a break occurs due to an external trigger break, the following message is displayed on the Status Bar.

`Break at` Address **`by external trigger break`**

2.4.8 Break Points (SIM)

A break point is a program memory location where the simulator suspends the program each time the point is reached while the program is executing (while executing execution commands). Normally a break point is set in the program space (the space where the **CODE** attribute is specified).

Break Points

Break points can be controlled using the ***[Run] - [Breakpoints] - [Code]*** command. When the program reaches a break point (immediately before executing the instruction at the memory location), the simulator executes the following processes:

1. Suspends program execution (before executing instruction).
2. Checks count of arrival time. If the count of arrival time at the specified break point has not yet been reached, the simulator resumes the program execution. If the count of arrival time has been reached, the simulator proceeds to step 3.
3. Displays memory location where execution suspended on Status Bar.
   Break points set using the ***[Run] - [Breakpoints] - [Code]*** command, remain valid until canceled or temporarily reset.

Up to 65535 break points can be set. Break points cause the program to suspend instruction operation just before executing the instruction at the specified address. In addition to using the ***[Run] - [Breakpoints] - [Code]*** command, break points can be set as follows:

- Setting break points in Source window
- Setting break points in Disassemble window
- **`Set Break`** command

The following message is displayed on the Status Bar when a break occurs due to a break point.

`Break at` Address **`by breakpoint`**

2.4.9 Data Break Points (SIM)

A data break point is the memory location where the simulator suspends a program execution when data access (Read/Write) is performed while executing a program (while executing execution commands).

Data Break Points

The simulator monitors whether the specified data access is made to a data break point while a program is executing; if such data access is made, it suspends the program. Data break points can be controlled using either the ***[Run] - [Breakpoints] - [Data]*** command, or the **`Set DataBreak`** command in the Command window. When specifying a data break point with a symbol, the starting address of that symbol becomes the data break point. Up to 65535 data break points can be set.

When a break occurs due to a data break point, the following message is displayed.

`Break at` Address **`by DataBreak`** at Access Address

Writing to Data Break Point

When data is written to a data break point, the simulator executes the following processes:

1. Suspends program execution after completing instruction execution
2. Checks access count. If the access count has not yet reached the count for the specified data break point, the simulator resumes the program execution. If the count has been reached, the simulator proceeds to step 3.
3. If program execution is suspended by reaching access count, on Status Bar, displays memory location of data break point and of instruction writing to it.
4. Displays memory location executed next

Reading from Data Break Point

When reading from a data break point, the simulator executes the following processes:

1. Suspends program execution after completing execution of the instruction.
2. Checks access count. If the access count has not yet reached the count for the specified data break point, the simulator resumes the program execution. If the count has been reached, the simulator proceeds to step 3.
3. If program execution is suspend by reaching count, on Status Bar, displays memory location of data break point and of instruction reading from it
4. Displays memory location executed next

Notes on Using Data Breaks

There are two points to note when using data break points as follows:

- If an automatic variable within a C/C++ function is specified, a data break is set at the address where the automatic variable is held. Therefore, the data break remains valid even after the specified automatic variable becomes invalid (after exiting function), causing a break due to unexpected access.
- To allow access to a variable in C/C++ to cause a break, specify the variable address by putting an ampersand symbol **"&"** immediately before the variable symbol.

2.4.10 Guarded Access Breaks (SIM)

A guarded access break suspends a executing program when accessing in violation of the access attribute set by using the ***[Setup] - [Memory Map]*** command, and accessing 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 execution suspended.

`Break at` Address **`by guarded access {code/read/write}`** at Access Address

2.4.11 Task Dispatch Break (SIM, EML, MON)

A task dispatch break is a break that happens when dispatch from the specified dispatch source task to the dispatch destination task. In other words, the break occurs when the dispatch destination task becomes the executing state. If the dispatch destination task is currently in the executing state, then the break occurs when the task reenters the executing state 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 E Embedding the REALOS Debug Module***.

To control the task dispatch break, use either of the following commands.

- ***[Run] - [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**`>`**

2.4.12 System Call Break (SIM, EML, MON)

A system call break occurs at ending execution of a system call specified by the specified task.

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 E Embedding the REALOS Debug Module***.

To control the system call break, use either of the following commands.

- ***[Run] - [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**`>`

2.4.13 Forced Break (SIM, EML)

A program execution can be forcibly suspended by using the ***[Run] - [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 debugger while the MCU is in the power-save mode or hold state. When a forced break is requested by the ***[Run] - [Abort]*** command while executing a program, the command is disregarded if the MCU is in the power-save mode or hold state. If a break must occur, then reset the factor at user system side, or reset the factor by using the ***[Run] - [Reset of MCU]*** command, after inputting the ***[Run] - [Abort]*** command.

When the MCU enters the power-save mode or hold state while executing, the status is displayed on the Status Bar.

2.5 Analyzing Program Execution (SIM, EML, MON)

The execution history can be traced and the instruction execution cycle count can be calculated to analyze a program.

Analyzing Program Execution

The following types are available for analyzing program execution.

- Trace
- Task Trace
- Measuring Execution Time

2.5.1 Trace (SIM, EML)

While executing a program, address and status information can be sampled and recorded in the trace buffer. This function is called trace.

Trace

The program execution history can be deep-analyzed using the data recorded by the trace function. This function is only available in the simulator debugger and the emulator debugger.

The trace buffer has a ring structure, so when the trace buffer becomes full, it automatically returns to the buffer start address to overwrite existing data.

- Trace data
- Tracing Function
- Setting trace
- Displaying trace data
- Displaying format of trace data
- Searching trace data
- Clearing trace data
- Notes on use of tracing function

2.5.2 Trace Data (SIM, EML)

Data sampled and recorded by tracing is called trace data. The recording format of trace data varies with each debugger.

Trace Data

You can sample the following sizes using the emulation debugger.

- Full Trace Mode, Real Time Trace Mode: 65536 Frames
- Internal Trace Mode: 128 Frames
- External Trace Mode: 65536 Frames

The following data is sampled.

- **Address** (32 Bits)
- **Data** (32 Bits)
- **Status Information**
  - **Access Data Size**
    Word/Half-word/Byte
  - **Data Types**
    Data Access/Instruction Execution

The simulator debugger can sample trace data by 1,000 frames. The address of the executed instruction is sampled as trace data.

2.5.3 Tracing Function (SIM, EML)

The status of the program execution is trace measured during the period from starting execution of the plan to the end of the execution of the program. With the DSU3 chip emulator you can trace measure up to the end of the execution of the program as the cause of starting trace measurement of code events (numbers 1 and 2) and data events (number 1).

Tracing Function

If the trace function is enabled, data is always sampled while executing the command and that is stored in the trace buffer.

In DSU3 chip emulator there is another function for trace measurement of the data access of a specified field and for starting trace measurement while executing the following programs.

- When you switch from the trace sampling mode to the trigger mode, trace measurements are started by the bit of either code events (numbers 1 and 2) or data events (number 1).
- When the MCU operation mode is set to internal trace mode or external trace mode, data is sampled only for data accessed in the specified data trace measurement area.

The program execution is terminated by the break of a break point, etc and tracing stops.

When the trace buffer is full, you can break the program. This break is called a trace buffer full break.

Frame Number

The sampled trace data is numbered in frame units. This number is called the frame number. To display data at a specific location in the trace buffer, specify the location with a frame number. The trace data sampled last is numbered 0, and the trace data sampled before the trigger generating location are numbered in negative numbers.

2.5.4 Setting Trace (SIM, EML)

You must set the following three items to perform a trace. After that, trace data will be sampled with the execution of the program. You can set this from the command window. When using the DSU3 chip, you can specify the trace measurement area of the data access.

Setting Trace

1. Enable the trace function

   This is done by ***[Setup] - [Trace]*** in the trace window shortcut menu. This program will startup and will be enabled.

2. Set the MCU operation mode **(Only Emulator Debugger)**

   Execute the ***[Environment]*** and ***[Debug Environment]*** commands.

   Full real time mode operates while executing, but there is a great possibility of losing trace data. Full trace mode does not operate while executing, but there is a very low possibility of losing trace data. If there are many divisional instructions, we recommend that you use the full trace mode.

   With the DSU3 chip, you can specify internal trace mode or external trace mode. Using these two modes, you can measure while operating during execution without losing trace data.

3. Set the trace buffer full break

   When the trace buffer is full, you can make a break. This is done using the setting dialog boxes of the trace window shortcut menu ***[Setup] - [Trace]***.

   When starting up this program, it is setup for no breaks.

   Also, on emulator debuggers using FR DSU3 chips, you can specify the data access area for performing the trace measurements.

2.5.5 Displaying Trace Data (SIM, EML)

Data recorded in the trace buffer can be Displayed.

Displaying Trace Data

The trace window displays how much trace data is stored in the trace buffer. Also, you can use the Show Trace command from the command window.

If the DSU3 chip is being used by the FR emulator debugger, this displays the data access information and divisional information as trace data. When you want to display instructions that were executed between divisional instructions, you need to open the trace details dialog box. Also, you can use the **`Show Detail Trace`** command from the command window.

2.5.6 Display Format of Trace Data (SIM, EML)

There are display formats for displaying trace buffer data:

Display Format of Trace Data

- **Display Only Instruction Operation (Specify Instruction)**
- **Display Bus Cycles (Specify Cycle)**
- **Display by Unit of Source Lines (Specify Source)**

If the DSU3 chip is being used by the FR emulator debugger, this displays only instruction executions. Source line unit displays are performed using the trace details dialog box.

Display Only Instruction Operation

In this mode, the instruction operation is displayed in disassembly units.

Display Bus Cycles

In this mode, detailed information on all sampled instruction fetch cycles and data access cycles is displayed. This mode is available only in the emulator debugger.

Display by Unit of Source Lines

This mode only displays source lines.

2.5.7 Searching Trace Data (SIM, EML)

Trace data can be searched to find where data to be displayed is stored.

Searching Trace Data

Specify the search by address, data or access information. In addition, the masking function can be used with address and data.

Click the Search button in the Trace window to use this function. Only search by address is available in the simulator debugger.

2.5.8 Clearing Trace Data (SIM, EML)

To clear trace data, use the following command.

Clearing Trace Data

Execution the ***[Clear]*** command from the short-cut menu in the Trace window to clear trace data.

2.5.9 Notes on Use of Tracing Function (SIM, EML)

There are several points to note when displaying or searching trace data.

Notes on Trace Function

When the emulator debugger is in use, tracing is enabled by the following:

- Output address information at fetching branch instruction

For these reasons, note the following points when displaying and searching trace data

- Since address information is not output immediately after executing a program until the branch instruction being executed, trace data may not be established on the program executing side.
- When displaying disassembly, data is read from memory and processed. Therefore, the displayed data may not be correct if the instruction is rewritten after code fetching.
- When specifying a starting frame number for searching data, an instruction longer than 2 bytes (LDI: 32, LDI: 20 instructions) may not be displayed correctly when the instruction starting address is not specified.
- In the full real-time mode, partial omission of trace data may occur under the following conditions (Output trace omission information instead) because of the real-time operation.

  When branching occurs more than three times within 11 cycles.

  When data tracing occurs more than three times in succession.
- The address is not displayed until the first branching information is found, because the trace data immediately before starting execution has been overwritten.
- If a break occurs under conditions such as the following combination of break points has been set up in sequence at continuous addresses (code addresses of factors in case of data event), the trace data immediately before the break is not displayed correctly.

  When break points set in sequence from software break to either one of I-group breaks at continuous addresses.

  When break points set in sequence from either one of I-group breaks to either one of I-group breaks at continuous addresses.

  The I-group breaks here means the following breaks:

  - Hardware break
  - Code event break
  - Data event break

- This occurs because the address next to the actual break factor address is detected as a break cause simply by such next address being pre-fetched.

  When displaying valid pass cycles or instruction, the omitted trace data frame is displayed as follows:

Frame where address at code fetching could not be sampled

***** Address Lost Error *****

- At step execution by a single instruction, trace data may not be sampled correctly for each single instruction execution. If this happens, ***** Address Lost Error ***** is displayed.

2.5.10 Task Trace (SIM, EML, MON)

Information on system call names, dispatched task IDs, and system call names that have timed out, can be traced.

Task Trace

Information, which is stored in trace buffer, on system call names issued by the task handler, dispatched task IDs and system call names that have been timed out, can be referred after executing a program. The task execution history can be analyzed with data recorded by tracing.

To use this function, the REALOS Debug Module must be embedded. For further details, see ***Operation Manual Appendix E Embedding the REALOS Debug Module***.

- Task Trace Data
- Task Trace Function
- Setting Task Trace
- Clearing Task Trace Data

2.5.11 Task Trace Data (SIM, EML, MON)

Data sampled and recorded by task trace is called task trace data.

Task Trace Data

Data sampled by the task trace function is called task trace data. The following data is sampled.

- Task ID
- System call

Task trace data can be sample as much as being set up by user definition.
The task trace buffer has a ring structure, so when the task trace buffer becomes full, it returns automatically to the start of the buffer to overwrite existing data.

User Definition

The **r_d_trc.asm** sample program contains **R_D_bufsiz**. This program defines the size of the task trace buffer.
Trace data uses 6 bytes per sample, so the specified size must be a multiple of 6.

2.5.12 Task Trace Function (SIM, EML, MON)

Task trace is a function to trace a program from start to stop.

Task Trace Function

If the task trace function is enabled, data is sampled continuously and recorded in the trace buffer while executing execution commands.

Frame Number

The sampled trace data is numbered in frame units. This number is called the frame number. To display data at a specific location in the trace buffer, specify the location using the frame number. The trace data at the program stop is numbered 0, and the trace data sampled before the program stops are numbered in negative number.

2.5.13 Setting Task Trace (SIM, EML, MON)

Perform the following steps 1 to 2 tracing. After these settings, trace data sampling start when the program execution starts.

Setting Task Trace

1. **Enable trace function**

   Use the ***[Setup] - [Task Trace]*** command in the short-cut menu in the Trace window to enable the trace function. The start-up default is disable.

2. **Set TASK ID**

   Set TASK ID using the ***[Setup] - [Task Trace]*** command in the short-cut menu in the Trace window. When All Tasks is selected, Specifying Individual Task is disabled.

2.5.14 Clearing Task Trace Data (SIM, EML, MON)

Clear task trace data as follows:

Clearing Task Trace Data

Clear task trace data by executing the **Clear** command in the Short Cut Menu, when the Trace window displays task trace data.

2.5.15 Measuring Execution Time (EML)

The program instruction execution time can be displayed by using either the ***[Analyze] - [Time Measurement]*** command, or the **`Show Timer`** command in the Command window.

Measuring Execution Time

Measures program execution time.

The measurement result can be displayed as two time values: the execution time of the preceding program, and the total execution time of the programs (total execution time before preceding program plus execution time of preceding program). Measurement is performed each time a program is executed.

Clear the measured values using the **`Clear Timer`** command.

2.5.16 Measuring Execution Time (SIM)

The instruction cycle count and step count of a program can be displayed by using either the ***[Analyze] - [Time Measurement]*** command, or the **`Show Timer`** command in the Command window,.

Measuring Execution Time

Measures program execution cycle count and step count.

The measurement result can be displayed as two time values: the execution time of the preceding program, and the total execution time of the programs (total execution time before preceding program plus execution time of preceding program). Measurement is performed each time a program is executed.

To display the execution cycle count, use the ***[Analyze] - [Time Measurement]*** command or the **`Show Timer`** command in the Command widow. Clear the measurement values using the **`Clear Timer`** command.

The counters for the instruction execution cycle count and program step count are both H'1 to H'FFFFFFFF.

The count of the instruction execution cycle is calculated based on the basic cycle count of each instruction described in the ***Programming Manual***.

Since the chip internal pipeline processing and cache operation are not simulated, such counts may be erroneous and different from those of the actual chip.

A compensation value (a, b), described in the list of instructions in the ***Programming Manual***, is calculated as one for each.

2.5.17 Measuring Execution Time (MON)

The program instruction execution time can be displayed by using either the ***[Analyze] - [Time Measurement]*** command, or the **`Show Timer`** command in the Command window.

Measuring Execution Time

Measures program execution time.

The measurement result can be displayed as two time values: the execution time of the preceding program, and the total execution time of the programs (total execution time before preceding program plus execution time of preceding program). Measurement is performed each time a program is executed.

To display the execution cycle count, use either the ***[Analyze] - [Time Measurement]*** command, or the **`Show Timer`** command in the Command window. Clear the measured values using the **`Clear Timer`** command.

Measurement is in 1 us units. The maximum measurement time is about 70 minutes. The measurement result may have a ±10 us error.

Chapter 3 FFMC-16 Family

This chapter describes the FFMC-16 family functions depending on MCUs.

3.1 Simulator

This section describes the functions of the simulator for the FFMC-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.

Simulation Range

The simulator simulates the MCU operations (instruction operations, memory space, I/O ports, interrupts, reset, power-save mode, etc.) using software to execute operations. It does not support built-in resources and related registers not described in the manual.

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

3.1.1 Instruction Simulation

This section describes the instruction simulation executed by SOFTUNE WORKBENCH.

Instruction Simulation

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

3.1.2 Memory Simulation

This section describes the memory simulation executed by SOFTUNE WORKBENCH.

Memory Simulation

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

- 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 3.-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 3-1-1 Types of Access Attributes

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

3.1.3 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 Inport`** command in the Command window.
Up to 16 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 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 16 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.

3.1.4 Interrupt Simulation

This section describes 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)

3.1.5 Reset Simulation

This section describes the reset simulation executed by SOFTUNE WORKBENCH.

Reset Simulation

The simulator simulates the operation when a reset signal is input to the MCU using the [Debug]-[Run]-[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..

3.1.6 Power-Save Consumption Mode Simulation

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

Power-Save Consumption Mode Simulation

The MCU enters the power 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 ***[Run] - [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.

3.2 Emulator

This section describes the functions of the emulator for the FFMC-16 family.

Emulator

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

Before using this emulator, the ICE must be initialized.

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

3.2.1 Setting Operating Environment

Before operating the emulator, set the operating environment such as the MCU operation mode, memory mapping, the timer minimum measurement unit, etc. However, each setting has a default. No setup is required if the defaults are used.

MCU Operation Mode

There are two MCU operation modes as follows: Selecting the operation mode changes the emulator debug environment.

- Debugging mode
- Native mode

Debug Area

Sets intensive debugging area in memory space that actual chip can handle.
The extension of break points/data break point count in this area, and the coverage measurement function is enhanced.

Memory Mapping

A memory space can be allocated to the user memory or the emulation memory. In addition, referencing on-the-fly is enabled by converting part of the emulation memory into a mirror of user memory.

Timer Minimum Measurement Unit

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

3.2.1.1 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 3-2-1 are imposed on the debug functions.

Table3-2-1 Restrictions on debug functions

Applicable series	Restrictions on debug functions
F²MC-16/16H	-Memory mapping is disabled and each area is accessed to the MCU specifications. -Traces cannot be disassembled.
Common to all series	-If bus access occurs inside and outside the MCU, external bus access information is not sampled at tracing.

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 FFMC-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.

3.2.1.2 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 512-KB 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 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.

- **Enhancement of Data Break Points**

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

  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 debug 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.

3.2.1.3 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 FFMC-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.

  Up to five areas (including mirror area described below) each with a max. 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.

  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. Re-executing this setup may damage data. The FFMC-16/16H only allows this setup in the debugging mode.

- **Mirror Area**

  The mirror area is an area for storing a copy of user memory access data to the emulator memory. This memory is called mirror memory.

  The mirror area is overlapped either by the user memory area or by the undefined area. This mirror area is enabled by using part of the emulation memory. Up to seven areas, including the emulation memory areas, can be set.

  The mirror area is used to refer to user memory on-the-fly (For further details, see ***On-the-fly Memory Access***).

To set the mirror area, use the **SET MAP** command. If Copy Memory Data is specified at same time as area setting, this mirror area always contains the same data as the user memory data.

<Note>

The FFMC-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.

  Only one internal ROM area with a size up to 128 KB can be specified. To set the internal ROM area, use ***"Setup CPU Information"*** from ***"Setup Project Basics"***. The area attribute is set automatically to **READ/CODE**.

- **Internal ROM Image Area (FFMC-16L, FFMC-16LX, FFMC-16F only)**

  Some types of MCUs have data in a specific area of internal ROM oppearing 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.

- **Internal Instruction RAM Area (FFMC-16H only)**

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

  To set the internal instruction RAM area, use ***"Set up CPU Information"*** from ***"Setup Project Basics"***. The size must be specified to either H'100, H'200, H'400, H'800, H'1000, H'2000 or H'4000 bytes.

- **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 FFMC-16/16H only allows this setup in the debugging mode.

3.2.1.4 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 3-2-2*** 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 3-2-2 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
>
```

3.2.1.5 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 us 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 3-2-3 shows the maximum measurement time length of each timer when 1 us 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 us.

Table 3-2-3 Maximum Measurement Time Length of Each Timer

	1 us selected	100 ns selected
Sequencer timer	About 16 s	About 1.6 s
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
>
```

3.2.2 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, 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.

    FFMC-16L/16LX/16/16H: - **PCB** - **DTB** - **NCC** - **ADB** - **SPB** - **CNR**
    - **MOV ILM,#imm8** - **AND CCR,#imm8**
    - **OR CCR,#imm8** - **POPW PS**

    FFMC-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, 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.

  FFMC-16L/16LX/16/16H: - **MOVS** - **MOVSW** - **SECQ** - **SECQW** - **WBTS**
  - **MOVSI** - **MOVSWI** - **SECQI** - **SECQWI** - **WBTC**
  - **MOVSD** - **MOVSWD** - **SECQD** - **SECQWD**
  - **FILS** - **FILSI** - **FILSW** - **FILSWI**

  FFMC-16F: Above plus - **MOVM** - **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
```

3.2.3 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 3-2-4*** 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 3-2-4 Major Functions Executable in On-the-fly Mode

Function	Limitations and Restrictions	Major Commands
MCU reset	—	**RESET**
Displaying MCU execution status	—	**SHOW STATUS**
Displaying trace data	Enabled only when trace function disabled	**SHOW TRACE** **SHOW MULTITRACE**
Enable/Disable trace	—	**ENABLE TRACE** **DISABLE TRACE**
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**
Setting event	Disabled in performance mode	**SET EVENT** **SHOW EVENT** **ENABLE EVENT** **DISABLE EVENT** **CANCEL EVENT**

3.2.4 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. However, the emulation memory can be accessed. (The 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 3.2-1***, 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 3.2-2***, access to the user memory by the MCU is reflected "as 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 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**

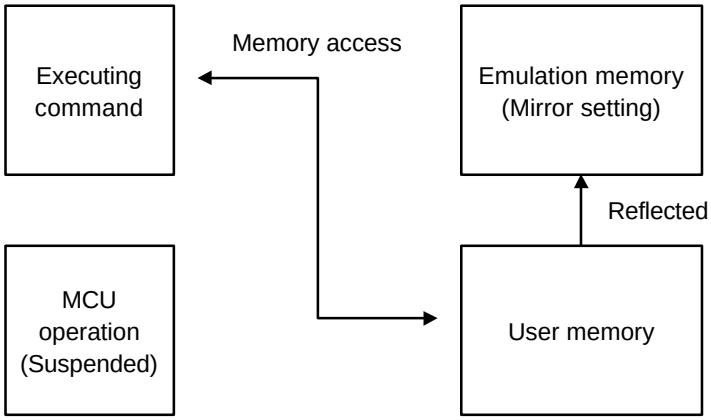


Figure 3.2-1 Access to Mirror Area while MCU Suspended

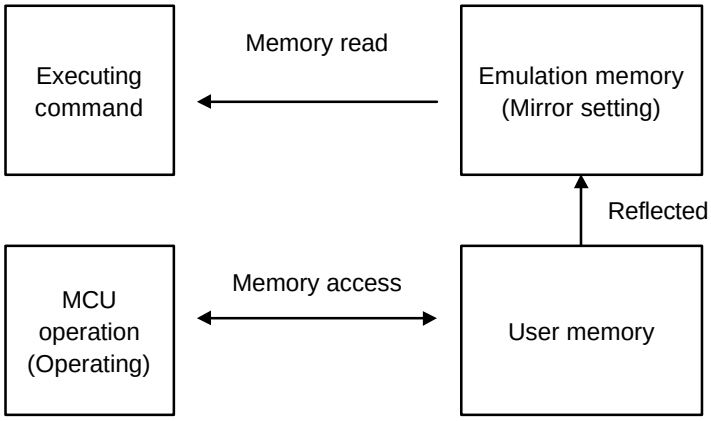


Figure 3.2-2 On-the-fly Access to Mirror Area

<Note>

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

3.2.5 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 for the following functions; to determine which function event triggers are used for depends on event modes.

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

Setting Events

Up to 8 events can be set.

Table 3-2-5 shows the conditions that can be set for events.

Table 3-2-5 Conditions for Events

Condition	Description
Address	Memory location (Address bit masking enabled)
Data	8-bit data (Data bit masking enabled) **NOT** can be specified.
Status	Choose from Data Read, Data Write, Instruction Execution*1, Data Modify*2
External Probe	8-bit data (Bit masking enabled)

*1: Instruction execution allows an event trigger generated only when the instruction has been executed. This cannot be specified together with another status at one time.

*2: Data Modify generates an event trigger when the data at the specified address is rewritten. When Data Modify is selected as the status, the data setting is disregarded. This cannot be specified together with another status at one time.

Use the following commands to set an event.

SET EVENT: Sets event

SHOW EVENT: Display event setup status

CANCEL EVENT: Deletes event

ENABLE EVENT: Enables event

DISABLE EVENT: Disables 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).

- **Multitrace 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.

3.2.5.1 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

Terminating the 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.

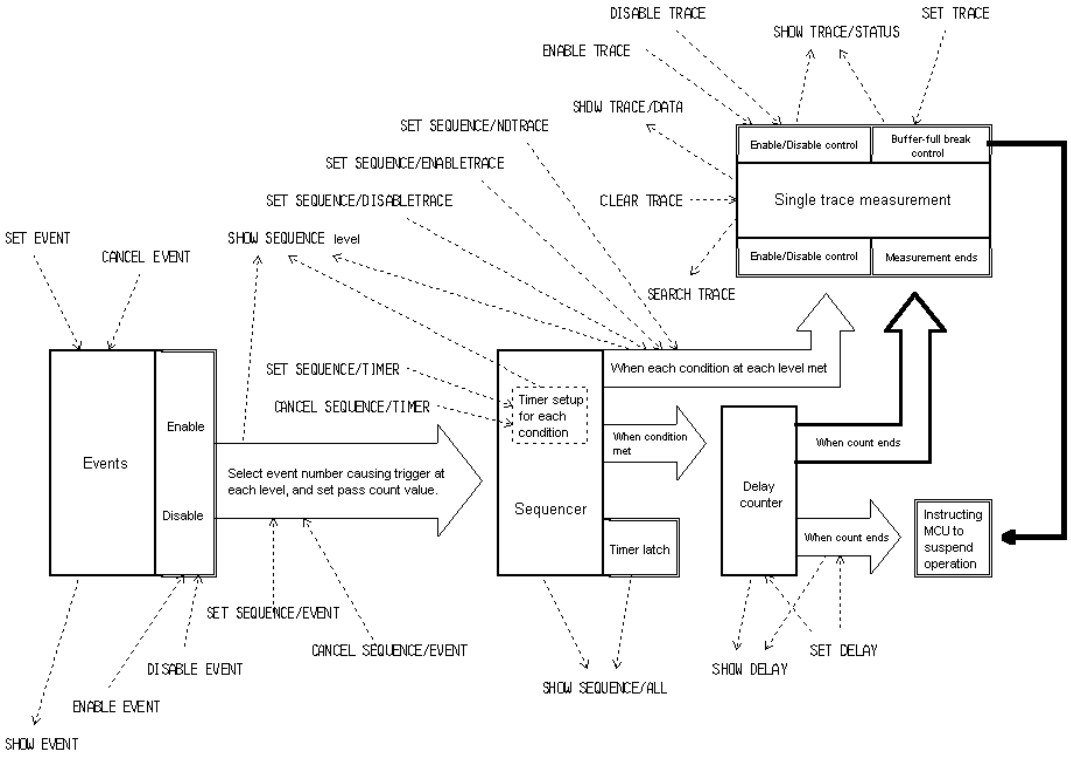


Figure 3.2-3 Operation in Normal Mode

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 the single trace.

Table 3-2-6 shows the event-related commands that can be used in the normal mode.

Table 3-2-6 Event-related Commands in Normal Mode

Mode	Usable Command	Function
Normal Mode	**Set Event**	Set event
	Show Event	Displays event setup status
	Cancel Event	Delete event
	Enable Event	Enables event
	Disable Event	Disables event
	Set Sequence	Sets sequencer
	Show Sequence	Displays sequencer setup status
	Cancel Sequence	Cancels sequencer
	Enable Sequence	Enables sequencer
	Disable Sequence	Disables sequencer
	Set Delay	Sets delay count
	Show Delay	Displays delay count setup status
	Set Trace	Sets trace buffer-full break
	Show Trace	Displays trace data
	Search Trace	Searches trace data
	Enable Trace	Enables trace function
	Disable Trace	Disables trace function
	Clear Trace	Clears trace data

3.2.5.2 Operation in Multitrace 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 Multitrace 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. Multitracing is a trace function that samples data before and after an event trigger occurrence.

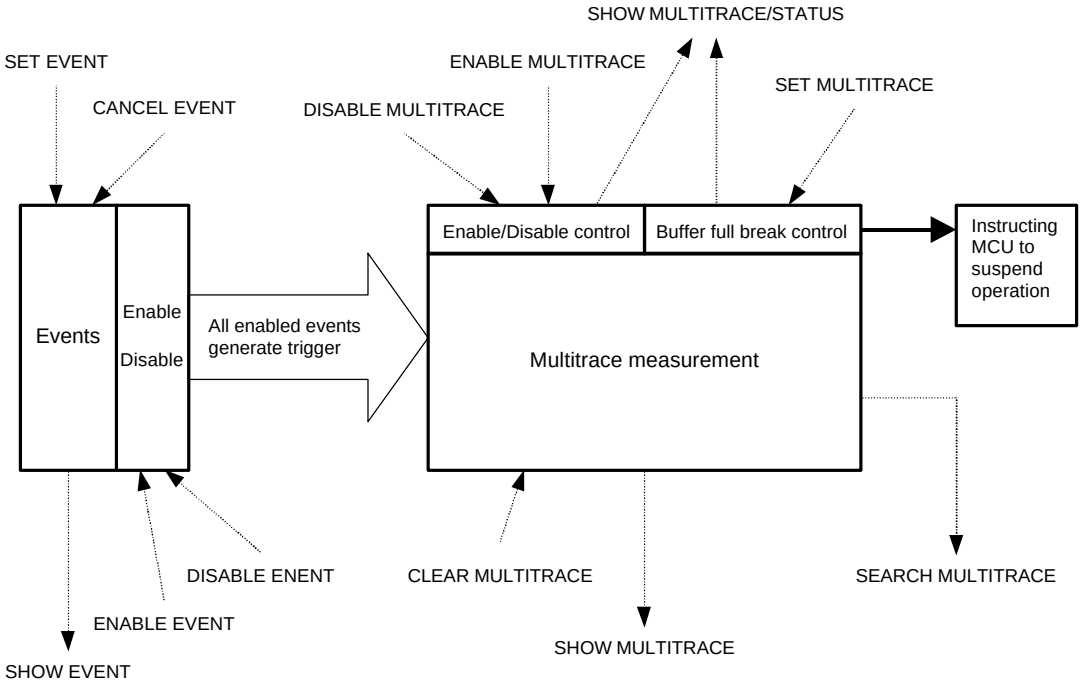


Figure 3.2-4 Operation in Multitrace Mode

Event-related Commands in Multitrace Mode

Table 3-2-7 shows the event-related commands that can be used in the multi-race mode.

Table 3-2-7 Event-related Commands in Multitrace Mode

Mode	Usable Command	Function
Multitrace Mode	**Set Event**	Sets event
	Show Event	Displays event setup status
	Cancel Event	Deletes event
	Enable Event	Enables event
	Disable Event	Disables event
	Set MultiTrace	Sets trace buffer-full break
	Show MultiTrace	Displays trace data
	Search MultiTrace	Searches trace data
	Enable MultiTrace	Enables trace function
	Disable MultiTrace	Disables trace function
	Clear MultiTrace	Clears trace data

3.2.5.3 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.

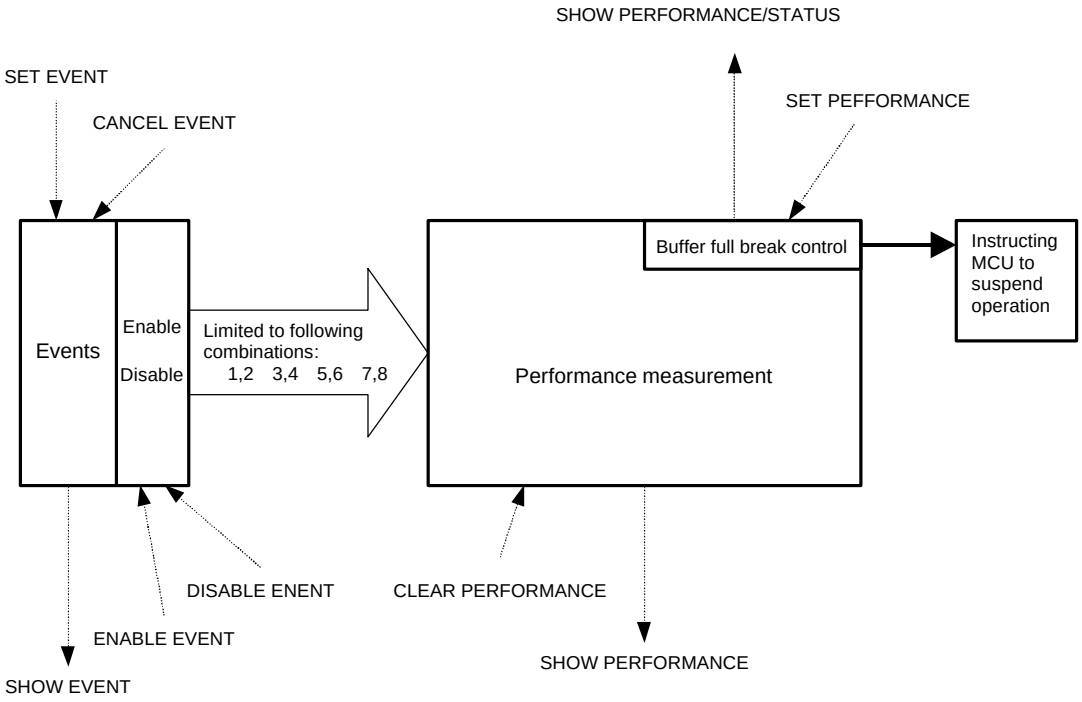


Figure 3.2-5 Operation in Performance Mode

Event-related Commands in Performance Mode

Table 3-2-8 shows the event-related commands that can be used in the performance mode.

Table 3-2-8 Event-related Commands in Performance Mode

Mode	Usable Command	Function
Performance Mode	**Set Event**	Sets event
	Show Event	Displays event setup status
	Cancel Event	Deletes event
	Enable Event	Enables event
	Disable Event	Disables event
	Set Performance	Sets performance
	Show Performance	Displays performance setup status
	Clear Performance	Clears performance measurement data

3.2.6 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 3-2-9***, 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 ORing. 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 3-2-9 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 us 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 us or 100 ns using the SET TIMERSCALE command.

3.2.6.1 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

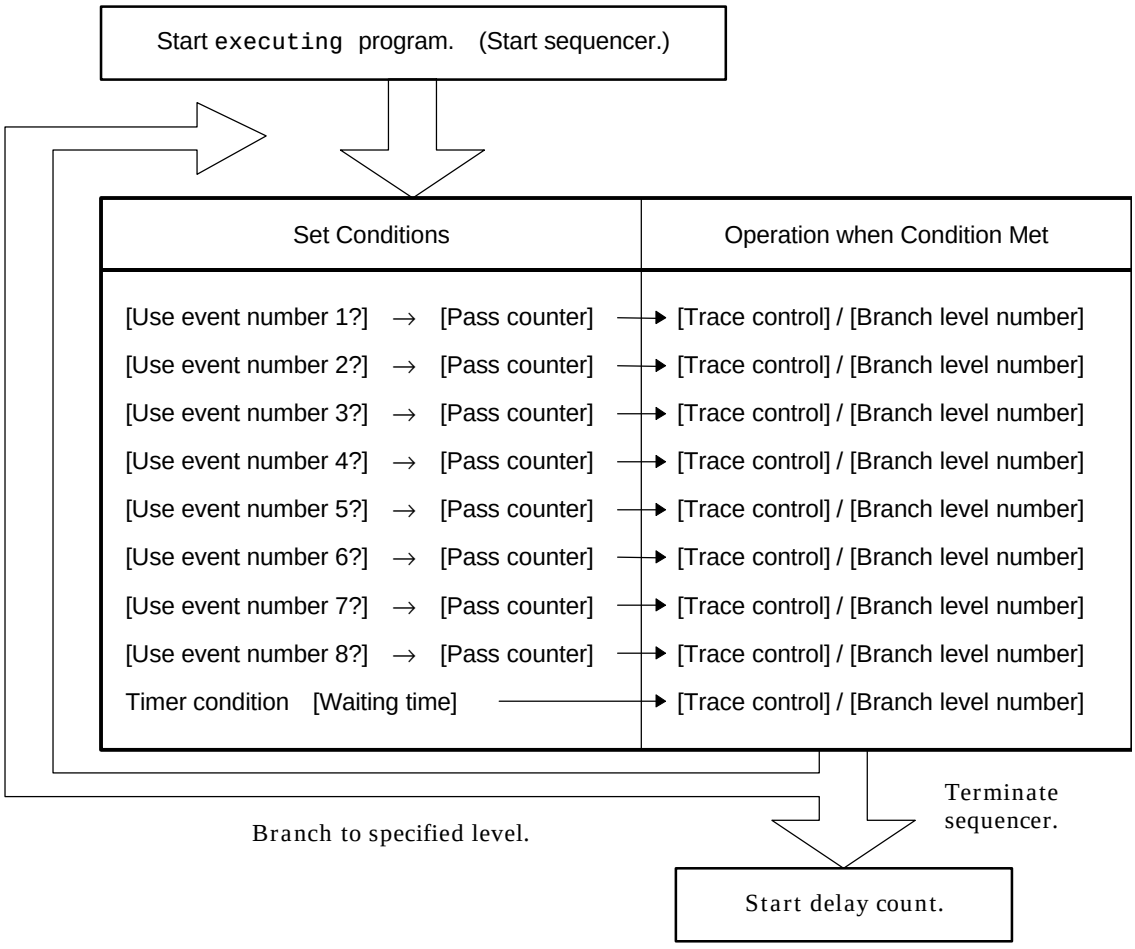


Figure 3.2-6 Operation of Sequencer

[Setup Examples]

- Terminate sequencer when event 1 occurs.

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

- Terminate sequencer when event 2 occurs 16 times.

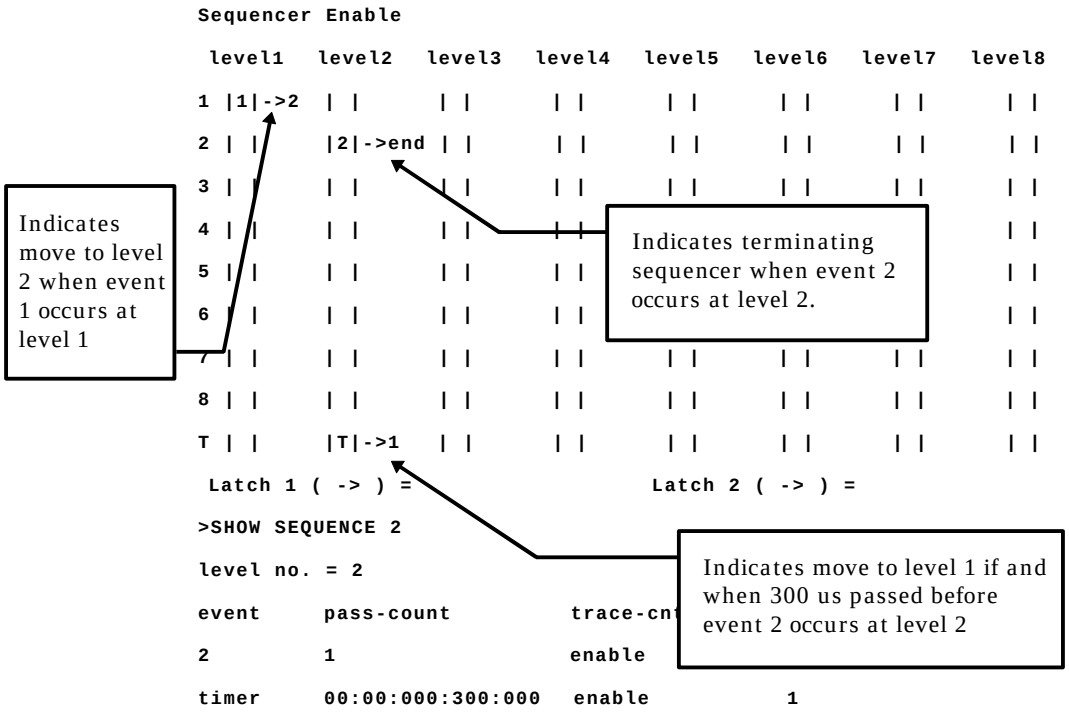
```
>SET SEQUENCE/EVENT 1,2,16,J=0
```

- 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 us 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
```



3.2.6.2 Break by Sequencer

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

Break by Sequencer

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

As shown in ***Figure 3.2-7***, 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.

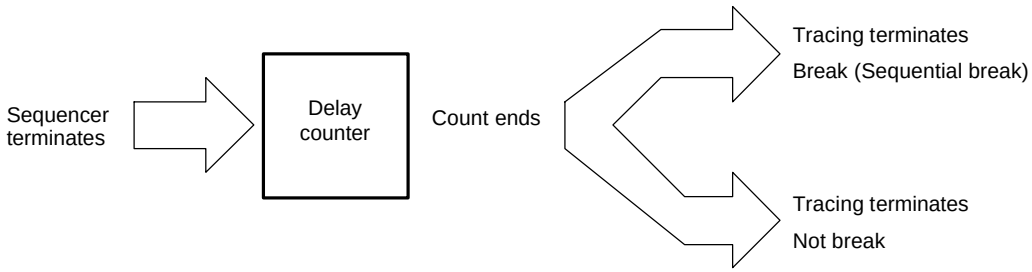


Figure 3.2-7 Operation when sequencer terminates

[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
```

3.2.6.3 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 3.2-8*** shows the trace sampling flow.
Trace data sampling can be restricted. For example, it is possible to suspend trace sampling when event 1 occurs, and then resume trace sampling when event 2 occurs.

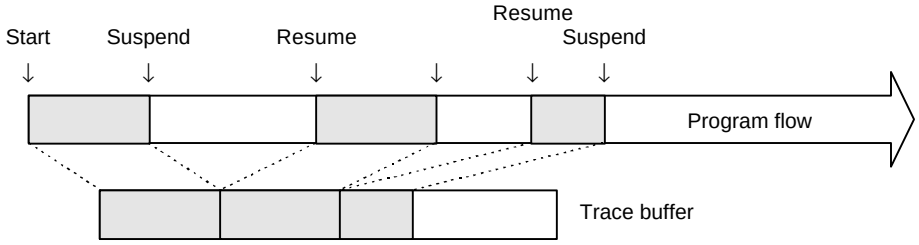


Figure 3.2-8 Controlling Trace Sampling Control (1)

As shown in ***Figure 3.2-9***, 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
```

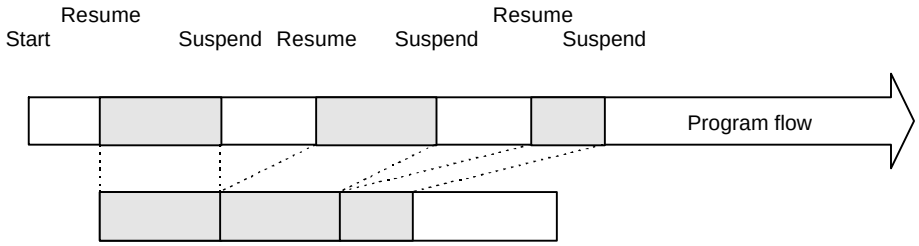
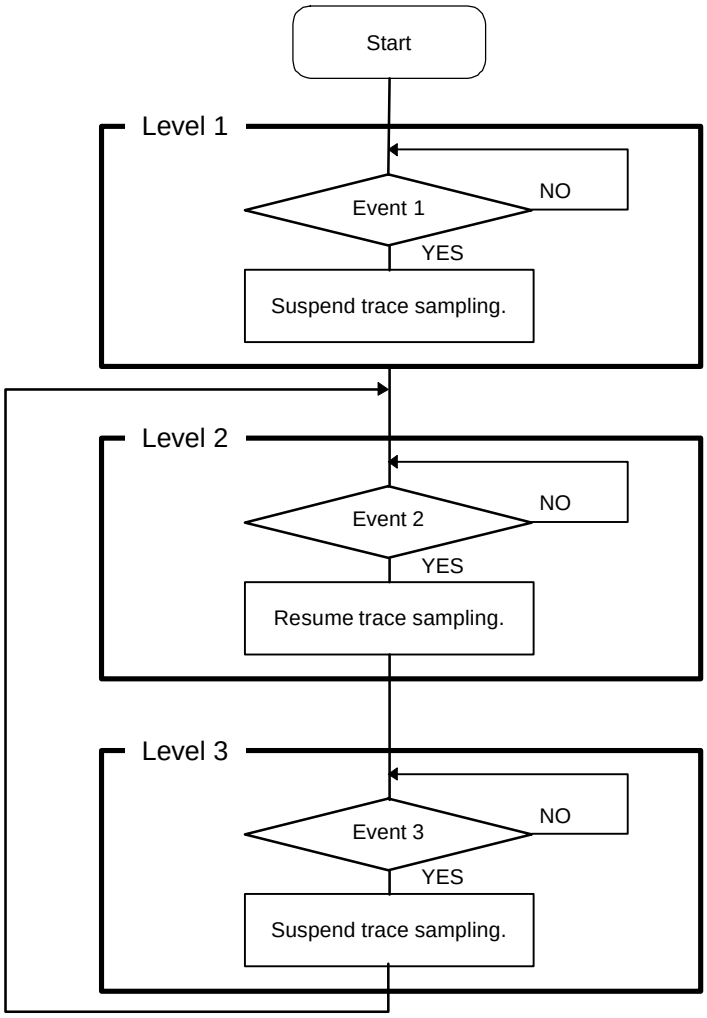


Figure 3.2-9 Controlling Trace Sampling Control(2)

[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
```

3.2.6.4 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 diplayed using the **SHOW SEQUENCE** command.

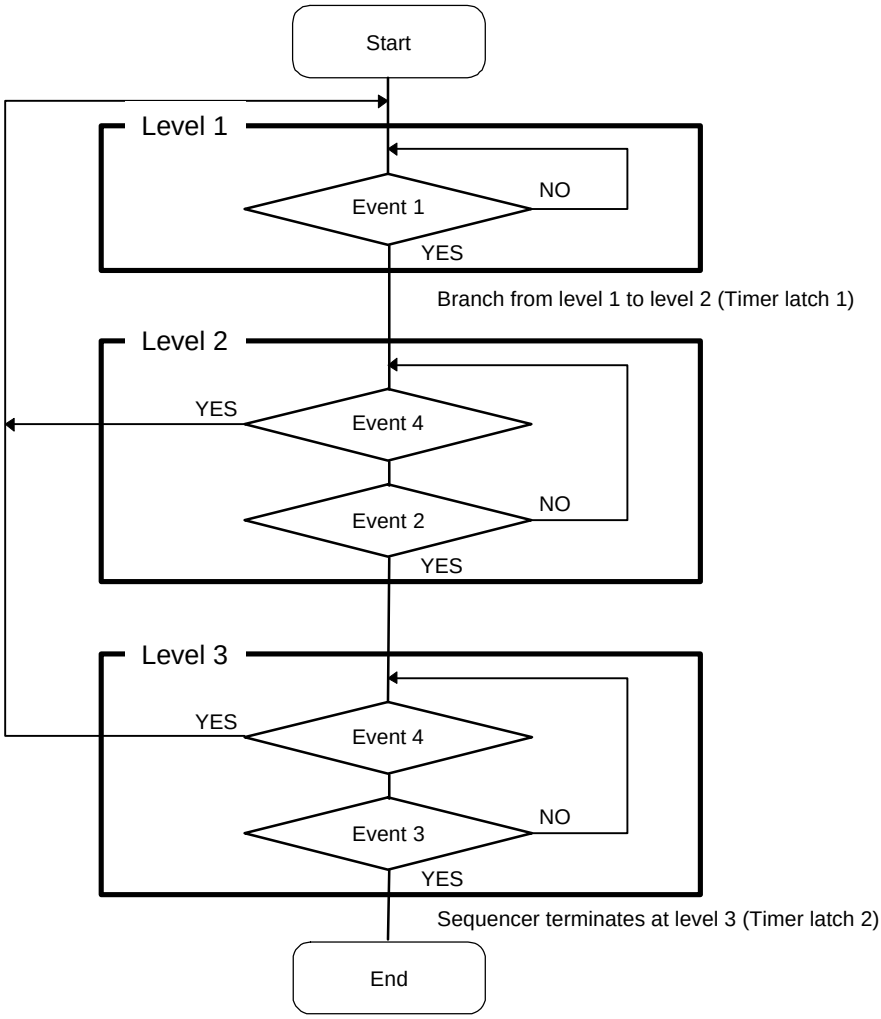
When a program starts execution, the emulation timer is initialized and then starts counting. Select either 1 us or 100 ns as the minimum measurement unit for the emulation timer. Set the measurement unit using the **SET TIMESCALE** command.

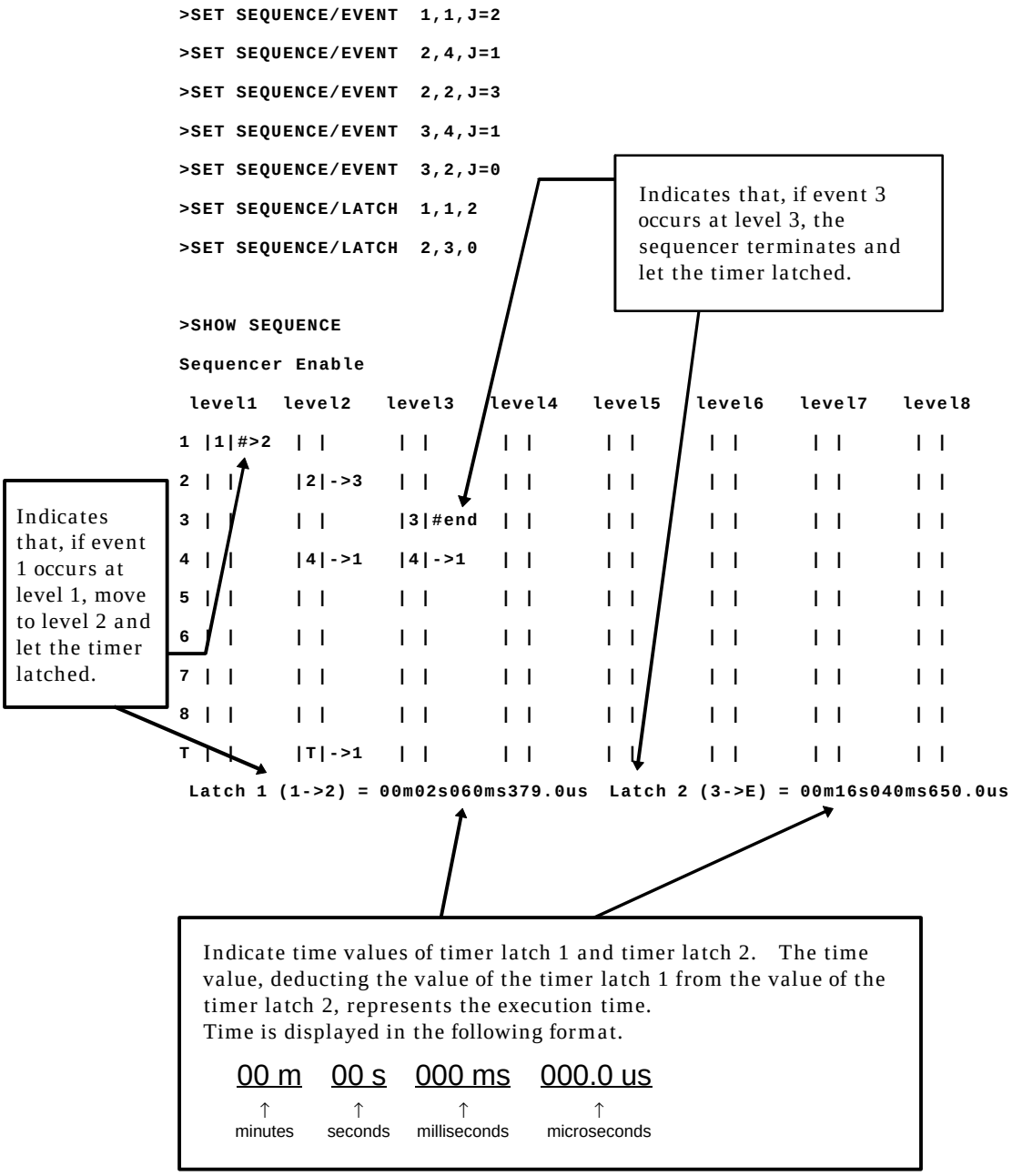
When 1 us is selected, the maximum measured time is about 70 minutes; when 100 ns is selected, the maximum measured time is about 7 minutes. If the timer overflows during measurement, a warning message is displayed when the timer value is displayed using the **SHOW SEQUENCE** command.

3.2.6.5 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




>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
T |T|->1
Latch 1 (1->2) = 00m02s060ms379.0us 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

3.2.7 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 32 Kframes (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 FFMC-16L/16LX/16/16H/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 in native mode.**

  When operating in the native mode, the FFMC-16L/16LX/16/16H/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).

3.2.7.1 Function of Single Trace

The single trace function 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 function 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 3.2-10***, 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.

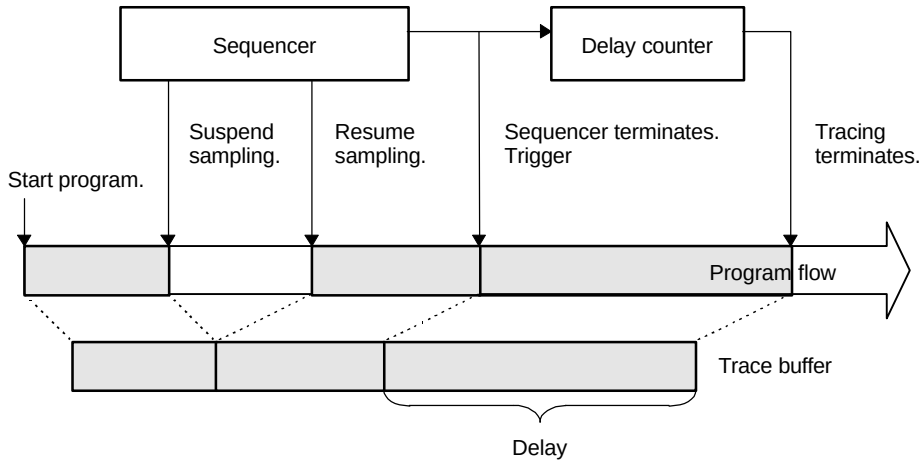


Figure 3.2-10 Sampling in Single Trace

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 ***(Figure 3.2-11)***.
If there is no sequencer termination trigger point available, the trace data sampled last is numbered 0.

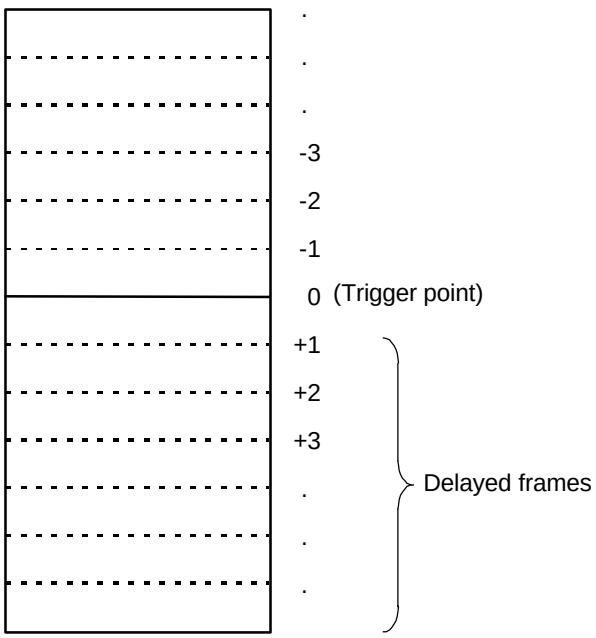


Figure 3.2-11 Frame Number in Single Trace

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.

3.2.7.2 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 3-2-10 lists trace-related commands that can be used in the single trace function.

Table 3-2-10 Single Trace Function Commands

Usable Command	Function
Set Event	Sets events
Show Event	Displays event setup status
Cancel Event	Deletes event
Enable Event	Enables event
Disable Event	Disables event
Set Sequence	Sets sequencer.
Show Sequence	Displays sequencer setting status
Cancel Sequence	Cancels sequencer
Enable Sequence	Enables sequencer
Disable Sequence	Disables sequencer
Set Delay	Sets delay count value, and operation after delay
Show Delay	Displays delay count setting status
Set Trace	Set traces-buffer-full break
Show Trace	Displays trace data
Search Trace	Searchs trace data
Enable Trace	Enables trace function
Disable Trace	Disables trace function
Clear Trace	Clears trace data

3.2.7.3 Multitrace Function

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

Multitrace Function

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

The multitrace function 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 32 Kframes, up to 2048 blocks can be sampled. Multitrace sampling terminates when the trace buffer becomes full. At this point, a executing program can be allowed to break if necessary.

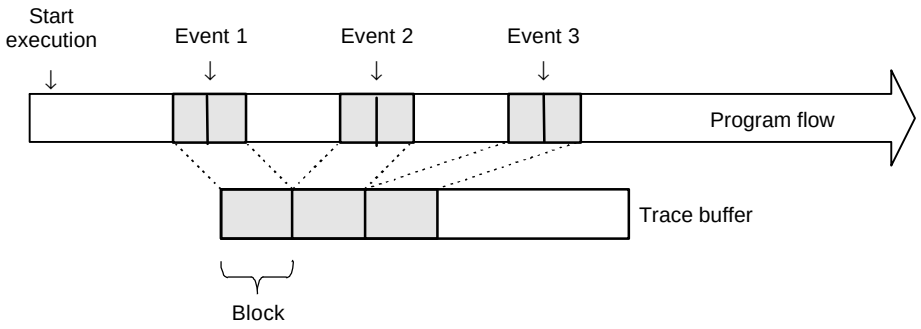


Figure 3.2-12 Multitrace Sampling

Multitrace Frame Number

Sixteen frames of data are sampled each time an event trigger 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. 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 3.2-13***).

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 32 Kframes, frames are numbered 1 to 32758 (***See Figure 3.2-13***).

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

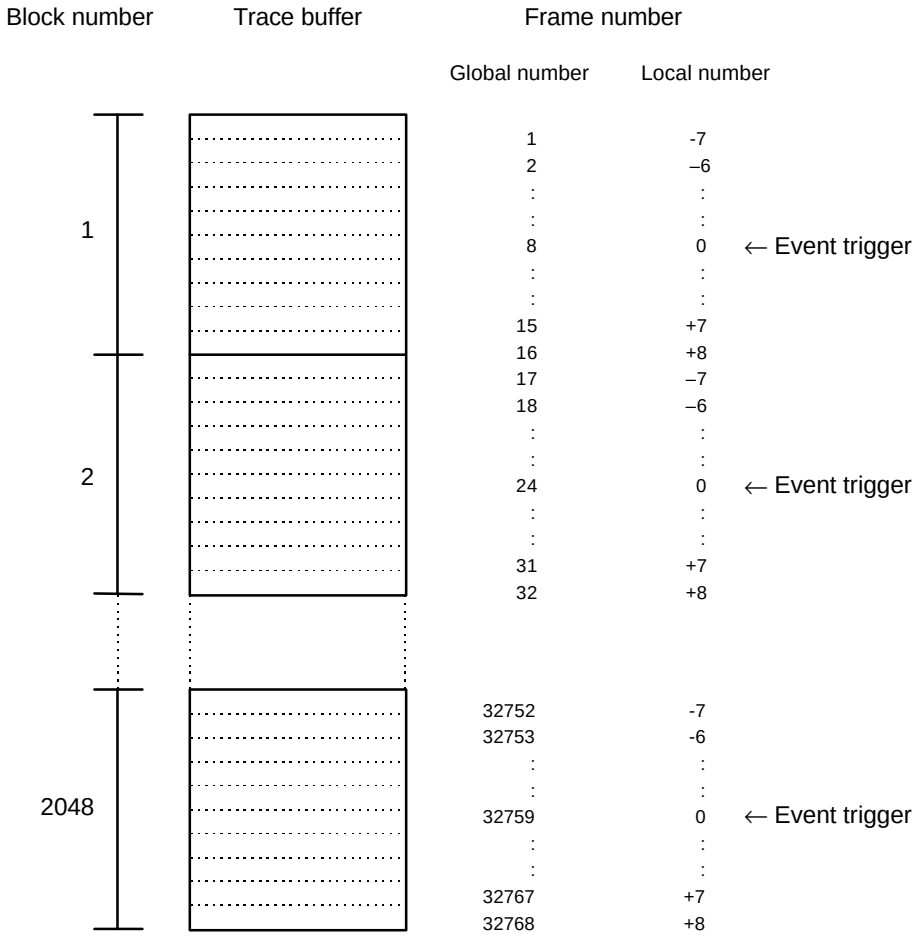


Figure 3.2-13 Frame Number in Multitrace Function

3.2.7.4 Setting Multitrace

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

- Set event mode to multitrace mode.
- Enable trace function.
- Set event.
- Set trace-buffer-full break.

Setting Multitrace

Before executing the multitrace function, 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 sampiling. 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 3-2-j shows the list of trace-related commands that can be used in multitrace mode.

Table 3-2-11 Multitrace Mode Commands

Usable Command	Function
Set Event	Sets events
Show Event	Displays event setup status
Cancel Event	Deletes event
Enable Event	Enables event
Disable Event	Disables event
Set Multitrace	Sets trace-buffer-full break
Show Multitrace	Displays trace data
Search Multitrace	Searches trace data
Enable Multitrace	Enables trace function
Disable Multitrace	Disables trace function
Clear Multitrace	Clears trace data

3.2.7.5 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 adding **/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 adding **/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.

[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 Multitrace

```
>SHOW MULTITRACE/STATUS
en/dis      = enable            Multitrace 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)
```

3.2.7.6 Specifying 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 or step 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 or step 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 Multitrace

>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

3.2.7.7 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.

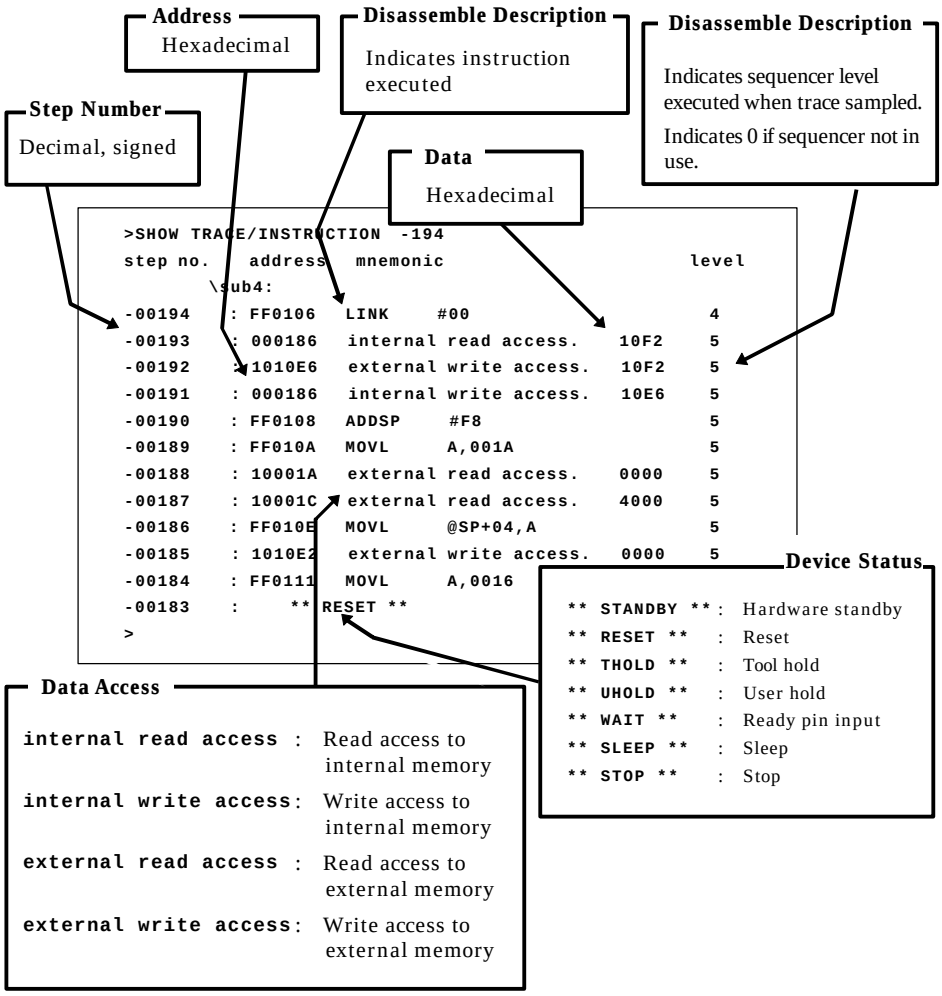
There are three formats to display trace data:

- 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.


Address
Hexadecimal
Disassemble Description
Indicates 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/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 **
>
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 (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
```

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;
```

3.2.7.8 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, **%TRCS**.

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 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.

Reading Trace Data On-the-fly in the Multitrace

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
               .               .              .
               .               .              .
```

3.2.8 Measuring Performance

It is possible to measure the time and pass count between two events. Repetitive 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.

3.2.8.1 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 us or 100 ns as the minimum measurement unit for the timer used to measure performance. The default is 1 us.
When the minimum measurement unit is changed, the performance measurement values are cleared.

[Example]

```
>SET TIMERSCALE/1U       <- Set 1 us 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]

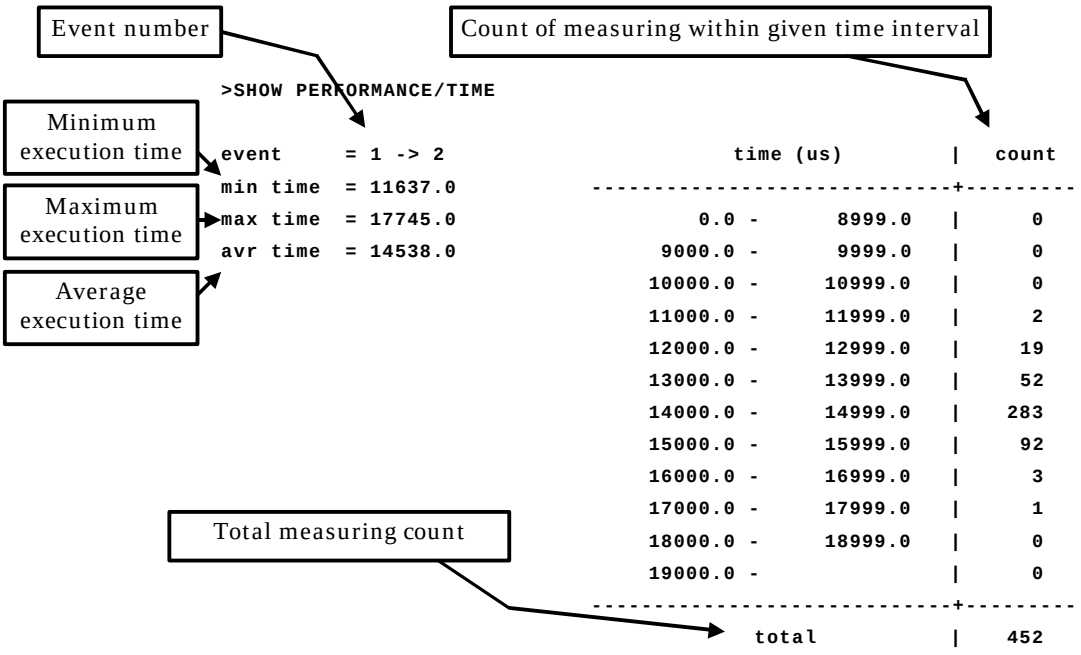
```
>CLEAR PERFORMANCE
>
```

3.2.8.2 Displaying Performance Measurement Data

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

Displaying Measured Time

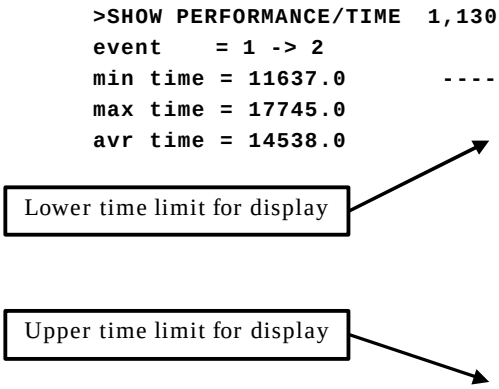
To display the time measured, specify the starting event number or the ending event number.




```
>SHOW PERFORMANCE/TIME
event     = 1 -> 2
min time  = 11637.0
max time  = 17745.0
avr time  = 14538.0

                time (us)          |   count
-------------------------------+---------
        0.0 -      8999.0      |      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 1us, when the minimum measurement unit is set to 1 us by the **SET TIMESCALE** command, and in 100 ns when the minimum is set to 100 ns.




```
>SHOW PERFORMANCE/TIME  1,13000,16999,500
event    = 1 -> 2                time (us)            |   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
```

3.2.9 Measuring Coverage

This emulator has the Co 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 Co 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**

3.2.9.1 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 0FF00..0FFFF
```

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 Measurement Result

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

- Coverage ratio of total measurement area
- Summary of 16 addresses as one block
- Details indicating access status of each address

- Coverage Ratio of Total Measurement Area (Specify **/TOTAL** for command qualifier.)

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

- Summary (Specify **/GENERAL** for command qualifier.)

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

Indicates access status of 16 addresses in one block

- `.` : No access
- `1` to `F` : Displays hexadecimal count of access to 16 addresses
- `*` : All 16 addresses accessed

- Details (Specify **/DETAIL** for command qualifier.)

Indicates coverage ratio for one line

```
>SHOW COVERAGE/DETAIL 0FF00

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
-----------------------------------------------------TOTAL   56.9
```

Indicates access status of each address

- `.` : Not accessed
- `-` : Accessed

Indicates coverage ratio whole displayed lines

3.2.10 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 us or 100 ns as the minimum measurement unit for the emulation timer and set the measurement unit using the **SET TIMESCALE** command.

When 1 us is selected, the maximum is about 70 minutes; when 100 ns is selected, the maximum is about 7 minutes.

The default is 1 us.

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 timer

[Example]

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

3.2.11 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 3-2-12)***. 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 3-2-12 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 3-2-13 shows the commands that can be used to set or display external probe data.

Table 3-2-13 Commands for External Probe Data

Usable Command	Function
Set Sampling	Sets sampling timing for external probe
Show Sampling	Samples external probe data
Set Event	Enables to specify external probe data as condition for event 1
Show Event	Displays event setup status
Show Trace	Displays external probe trace-sampled (single trace)
Show Multitrace	Displays external probe trace-sampled (multi-trace)

3.3 Monitor Debugger

This section describes the functions of the monitor debugger for the FFMC-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.

3.3.1 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 band
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 us units.

3.4 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 emulator 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

3.4.1 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.

    FFMC-16L/16LX/16/16H: - **PCB** - **DTB** - **NCC** - **ADB** - **SPB** - **CNR**
    - **MOV ILM,#imm8** - **AND CCR,#imm8**
    - **OR CCR,#imm8** - **POPW PS**

    FFMC-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, 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.

FFMC-16L/16LX/16/16H:	- **MOVS**	- **MOVSW**	- **SECQ**	- **SECQW**	- **WBTS**
	- **MOVSI**	- **MOVSWI**	- **SECQI**	- **SECQWI**	- **WBTC**
	- **MOVSD**	- **MOVSWD**	- **SECQD**	-	**SECQWD**
	- **FILS**	- **FILSI**	- **FILSW**	- **FILSWI**	
FFMC-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, 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.

3.4.2 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 points

SHOW DATABREAK: Displays current break point setup status

CANCEL DATABREAK: Cancels break point

ENABLE DATABREAK: Enables break points

DISABLE DATABREAK: Temporarily cancels break points

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>

In the emulator, if the debug area is set up again, the break points in the area are all cleared.

3.4.3 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.

- ***[Run]-[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.

3.4.4 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.)

3.4.5 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 FFMC-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 FFMC-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.

3.4.6 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 ***[Setup]-[Trace]*** command in the short-cut menu of the ***[Analyze]-[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`**

3.4.7 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**

3.4.8 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 E Embedding the REALOS Debug Module***.

To control the task dispatch break, use either of the following commands.

- ***[Run]-[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**`>`**

3.4.9 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 E Embedding the REALOS Debug Module***.

To control the system call break, use either of the following commands.

- ***[Run]-[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**`>`

3.4.10 Forced Break (SIM, EML)

A executing program can be forcibly aborted by using the ***[Run]-[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 mode or hold state. When a forced break is requested by the ***[Run]-[Abort]*** command while executing a program, the command is disregarded if the MCU is in the power-save mode or hold state. If a break must occur, then reset the cause at user system side, or reset the cause by using the ***[Run]-[Reset of MCU]*** command, after inputting the ***[Run]-[Abort]*** command.

When the MCU enters the power-save mode or hold state while executing, the status is displayed on the Status Bar.

Chapter 4 FFMC-8L Family

This chapter describes the FFMC-8L family functions depending on MCUs.

4.1 Simulator

This section describes the functions of the simulator for the FFMC-8LFamily

Simulator

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

Simulation Range

The simulator simulates the MCU operations (instruction operations, memory space, I/O ports, interrupts, reset, power-save mode, etc.) using software to execute operations. It does not support built-in resources and related registers not described in the manual.

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

4.1.1 Instruction Simulation

This section describes the instruction simulation executed by SOFTUNE WORKBENCH.

Instruction Simulation

This simulates the operations of all instructions supported by the FMC-8L. It also simulates the changes in memory and register values due to such instructions.

4.1.2 Memory Simulation

This section describes the memory simulation executed by SOFTUNE WORKBENCH.

Memory Simulation

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

- 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 3.-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 4-1-1 Types of Access Attributes

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

4.1.3 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 Inport`** command in the Command window.
Up to 16 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 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 16 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.

4.1.4 Interrupt Simulation

This section describes interrupt simulation executed by SOFTUNE WORKBENCH.

Interrupt Simulation

Simulate the operation of the MCU in response to an interrupt request.

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)

4.1.5 Reset Simulation

This section describes the reset simulation executed by SOFTUNE WORKBENCH.

Reset Simulation

The simulator simulates the operation when a reset signal is input to the MCU using the [Debug]-[Run]-[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..

4.1.6 Power-Save Consumption Mode Simulation

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

Power-Save Consumption Mode Simulation

The MCU enters the power 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 ***[Run] - [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.

4.2 Emulator

This section describes the functions of the emulator for the FFMC-8L family.

Emulator

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

Before using this emulator, the ICE must be initialized.

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

4.2.1 Setting Operating Environment

Before operating the emulator, set the operating environment such as the MCU operation mode, memory mapping, the timer minimum measurement unit, etc. However, each setting has a default. No setup is required if the defaults are used.

MCU Operation Modes

The operation mode setting varies when a regular MCU is used and when a FFMC-8L piggy back/evaluation chip is used.

- **MCU mode**
  - Single chip mode (MCU Mode 0)
  - External ROM mode (MCU Mode 1)
  - Internal ROM mode with external access function (MCU Mode 2)

- **When FFMC-8L piggy back/evaluation chip used**

  There are two modes as follows:
  - Debugging mode
  - Native mode

Memory Mapping

A memory space can be allocated to the user memory or the emulation memory. However, certain restrictions apply to the space that can be set, depending on the selected MCU mode.

Timer Minimum Measurement Unit

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

4.2.1.1 MCU Operation Mode

There are three MCU operation modes as follows:

- Single chip mode (MCU Mode 0)
- External ROM mode (MCU Mode 1)
- Internal ROM mode with external access function (MCU Mode 2)

Setting MCU Operation Mode

The MCU operation mode varies depending on the product type; refer to the ***User Manual*** for each MCU for further details.

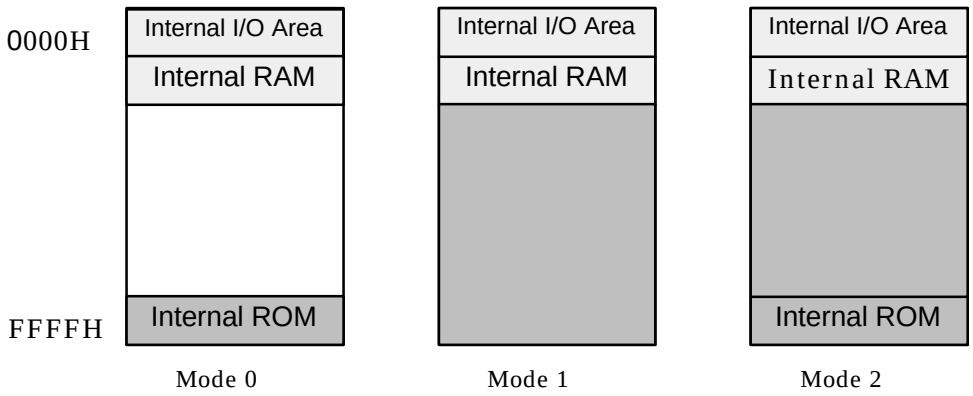


Figure 4.2-1 MCU Modes and Memory Mapping

As shown in ***Figure 4.2-1***, memory mapping operation varies depending on MCU mode. Internal RAM area (internal RAM, internal register and internal I/O) cannot map to the emulation memory because it accesses internal MCU regardless of mapping setup.

4.2.1.2 Operation Mode with Piggy back/Evaluation Chip

The operation mode must be set when a piggy back/evaluation chip is used.
There are two operation modes: debugging mode, and native mode as well as differences in memory mapping. To set the operation mode, use the **SET RUNMODE** command.

Operation Mode with Piggy back/Evaluation Chip

The emulator default is native mode. To change the operation mode, use the **SET RUNMODE** command.

- **Debugging Mode**

  In the debugging mode, the emulation memory is substituted for all spaces, except the internal RAM area.

  [Example]

  ```
  >SET RUNMODE /DEBUG
  ```

- **Native Mode**

  Becomes the same state with MCU mode. The emulation memory is substituted for the internal ROM area, and the external access area accesses the user memory.

  [Example]

  ```
  >SET RUNMODE /NATIVE
  ```

4.2.1.3 Memory Area Types

A unit to allocate memory is called an area. There are three different area types.

Memory Area Types

A unit to allocate memory is called an area.
Up to 20 areas can be set in 1-byte units. There is no limit on the size of an area.
An access attribute can be set for each area.
There are three 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.
  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.
  The user system bus master (DMAC, etc.) cannot access emulation memory.
  To set the emulation memory area, use the **SET MAP** command.

- **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.

4.2.1.4 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 4-2-1*** 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 4-2-1 Types of Access Attributes

Area	Attribute	Description
User Memory	READ	Data Read Enabled
Emulation Memory	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 are. However, if the access target is the emulation memory are, 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 Displaying Memory Map

Use the following commands for memory mapping.

SET MAP: Sets memory map
SHOW MAP: Displays memory map
CANCEL MAP: Changes memory map setting to undefined

[Example]

```
>SET MAP /USER H'0..H'1FFF
>SET MAP /READ/EMULATION H'FF00..H'FFFF
```

```
>SET MAP/GUARD
>SHOW MAP
address          attribute    type
0000 .. 1FFF     read write   user
2000 .. FEFF     guard
FF00 .. FFFF     read         emulation
```

4.2.1.5 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 us 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 4-2-2 shows the maximum measurement time length of each timer when 1 us 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 us.

Table 4-2-2 Maximum Measurement Time Length of Each Timer

	1 us selected	100 ns selected
Sequencer timer	About 16 s.	About 1.6 s.
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
>
```

4.2.2 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 "MCU busy error" occurs. ***Table 4-2-3*** 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.,

Table 4-2-3 Major Functions Executable in On-the-fly Mode

Function	Limitations and Restrictions	Major Commands
MCU reset	—	**RESET**
Displaying MCU execution status	—	**SHOW STATUS**
Displaying trace data	Enabled only when trace function disabled	**SHOW TRACE** **SHOW MULTITRACE**
Enable/Disable trace	—	**ENABLE TRACE** **DISABLE TRACE**
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**
Setting event	Disabled in performance mode	**SET EVENT** **SHOW EVENT** **ENABLE EVENT** **DISABLE EVENT** **CANCEL EVENT**

4.2.3 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. However, the emulation memory can be accessed. (The 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.
However, at least one time access must be allowed before the emulation memory with the mirror setting has the same data as the user memory. The following copy types allow the emulation memory with the mirror setting to have the same data as the user memory.

- 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**

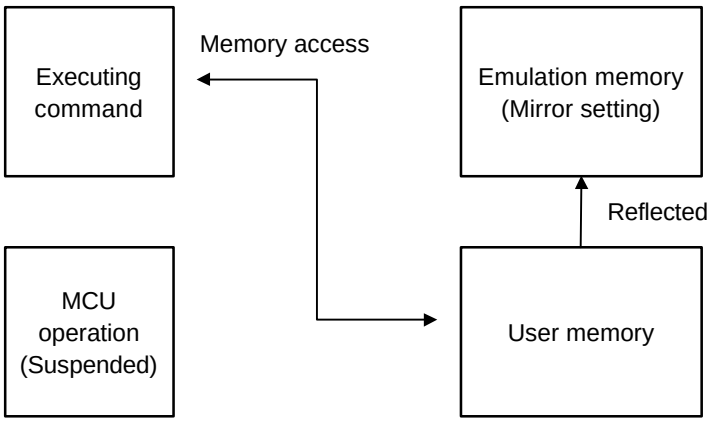


Figure 4.2-b Access to Mirror Area while MCU Suspended

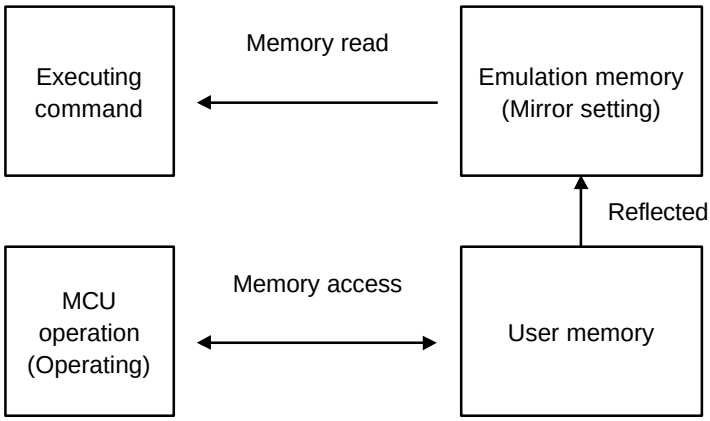


Figure 4.2-3 On-the-fly Access to Mirror Area

<Note>

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

4.2.4 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 for the following functions; use of function for event trigger depends on event modes.

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

Events

The FFMC-8L family has the same event function as the FFMC-16 family. See the ***FFMC-16 family description***.

Reference Section

Chapter 3 FFMC-16 family 3.2.5 EVENT

4.2.5 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.

Sequencer

The FFMC-8L family has the same Sequencer function as the FFMC-16 family. See the ***FFMC-16 family description***.

Reference Section

Chapter 3 FFMC-16 family 3.2.6 Control by Sequencer

4.2.6 Real-time Trace

While executing a program, the address, data and status information, and the data sampled by an external probe can be sampled in machine cycle units and recorded 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.

Real-time Trace

The FFMC-8L family has the same Real-time Trace function as the FFMC-16 family. See the ***FFMC-16 family description***.

Reference Section

Chapter 3 FFMC-16 family 3.2.7 Real-time Trace

4.2.7 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.

Measuring Performance

The FFMC-8L family has the same Measuring Performance function as the FFMC-16 family. See the ***FFMC-16 family description***.

Reference Section

Chapter 3 FFMC-16 family 3.2.8 Measuring Performance

4.2.8 Measuring Coverage

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

Measuring Coverage

The FFMC-8L family has the same Measuring Coverage function as the FFMC-16 family. See the ***FFMC-16 family description***.

Reference Section

Chapter 3 FFMC-16 family 3.2.9 Measuring Coverage

4.2.9 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.

Emulation Timer

The FFMC-8L family has the same Emulation Timer function as the FFMC-16 family. See the ***FFMC-16 family description***.

Reference Section

Chapter 3 FFMC-16 family 3.2.10 Measuring Execution Time using Emulation Timer

4.2.10 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.

External Probe

The FFMC-8L family has the same External Probe function as the FFMC-16 family. See the ***FFMC-16 family description***.

Reference Section

Chapter 3 FFMC-16 family 3.2.11 Sampling by External Probe

4.3 Monitor Debugger

The monitor debugger is not supported by the FFMC-8L family.

4.4 Abortion of Program Execution (SIM, EML)

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 aborted, 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, the following sources can abort program execution.

- Instruction Execution Breaks
- Data Access Breaks
- Guarded Access Breaks
- Task Dispatch Break
- System Call Break
- Forced Break

4.4.1 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

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 cancel 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 that can be set is 65535 both in the simulator and emulator.

Notes on Instruction Execution Breaks

A break occurs before executing the instruction if the break point is set immediately after the instructions listed in ***Table 4-4-1***. The debugger is designed to perform step execution internally, and then to break after such execution. Therefore, the last instruction is not executed in real-time.

If an instruction execution break is set following the 1-byte branch instruction shown below, it occurs immediately after the instruction is executed, because the 1-byte branch instruction is affected by prefetch of the next instruction when executed. Instructions when the instruction execution break is set are just prefetched but not executed.

RET	RET1	JMP @A	CALLV #vct

To avoid this, set the instruction execution break shifted one byte or set a breakpoint using the SET EVENT/CODE command, which is unaffected by prefetch.

Table 4-4-1 Instructions affecting instruction execution breaks

ADDC	A,@EP	ADDC	A, Ri	ADDC	A	ADDCW	A
AND	A,@EP	AND	A, Ri	AND	A	ANDW	A
CALLV	#n	CMP	A,@EP	CMP	A, Ri	CMP	A
CMPW	A	DAA		DAS		DEC	Ri
DECW	A	DECW	EP	DECW	IX	DECW	SP
DIVU	A	INC	Ri	INCW	A	INCW	EP
INCW	IX	INCW	SP	MOV	@A,T	MOV	@EP,A
MOV	A, A@	MOV	A,@EP	MOV	A, Ri	MOV	Ri, A
MOVW	@A, T	MOVW	A, @A	MOVW	A,@EP	MOVW	A, EP
MOVW	A, IX	MOVW	A,PC	MOVW	A, PS	MOVW	SP,A
MOVW	EP,A	MOVW	IX,A	MOVW	PS,A	MOVW	SP,A
MULU	A	OR	A, @EP	OR	A,Ri	OR	A
ORW	A	POPW	A	POPW	IX	PUSHW	A
PUSHW	IX	ROLC	A	RORC	A	SUBC	A,@EP
SUBC	A,RI	SUBC	A	SUBCW	A	SWAP	
XCH	A, T	XCHW	A,EP	XCHW	A,IX	XCHW	A,SP
XCHW	A, T	XOR	A,@EP	XOR	A,RI	XOR	A
XORW	A						

4.4.2 Data Access Breaks (SIM, EML)

A data access break is a function to abort a executing 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 suspend a running program when the MCU accesses data at the specified address.

Data access breaks can be controlled using the following commands:

SET DATABREAK:	Sets break points
SHOW DATABREAK:	Displays current break point setup status
CANCEL DATABREAK:	Cancels break point
ENABLE DATABREAK:	Enables break points
DISABLE DATABREAK:	Temporarily cancels break points

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 is 65535.

4.4.3 Sequential Break (EML)

A sequential break is a function to suspend 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.)

4.4.4 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

4.4.5 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 ***[Setup]-[Trace]*** command in the short-cut menu of the ***[Analyze]-[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`**

4.4.6 Performance-Buffer-Full Break (EML)

A performance-buffer-full break is a function to abort a 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**

4.4.7 Task Dispatch Break (SIM, EML)

A task dispatch break is a break that happens 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 running state. If the dispatch destination task is currently in the execution state, then the break occurs when the task enters the executing 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 refer to, ***Operation Manual Appendix E Embedding the REALOS Debug Module***.

To control the task dispatch break, use either of the following commands.

- ***[Run]-[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**`>`

4.4.8 System Call Break (SIM, EML)

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, refer to ***Operation Manual Appendix E Embedding the REALOS Debug Module***.

To control the system call break, use either of the following commands.

- ***[Run]-[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**`>`

4.4.9 Forced Break (SIM, EML)

A executing program can be forcibly aborted by using the ***[Run]-[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 mode or hold state. When a forced break is requested by the ***[Run]-[Abort]*** command while executing a program, the command is disregarded if the MCU is in the power-save mode or hold state. If a break must occur, then reset the cause at user system side, or reset the cause by using the ***[Run]-[Reset of MCU]*** command, after inputting the ***[Run]-[Abort]*** command.
When the MCU enters the power-save mode or hold state while executing, the status is displayed on the Status Bar.

CM81-00306-2E

FUJITSU SEMICONDUCTOR • CONTROLLER MANUAL

FR FAMILY F^{2}MC FAMILY
32/16/8-BIT MICROCONTROLLER
SOFTUNE Workbench USER'S MANUAL

November 1999 the second edition

Published **FUJITSU LIMITED** Electronic Devices

Edited Technical Communication Dept.

FUJITSU

CM81-00306-2E

FUJITSU SEMICONDUCTOR FR FAMILY F^2MC FAMILY 32/16/8-BIT MICROCONTROLLER SOFTUNE Workbench USER'S MANUAL