To our customers,

Old Company Name in Catalogs and Other Documents

On April 1st, 2010, NEC Electronics Corporation merged with Renesas Technology Corporation, and Renesas Electronics Corporation took over all the business of both companies. Therefore, although the old company name remains in this document, it is a valid Renesas Electronics document. We appreciate your understanding.

Renesas Electronics website: http://www.renesas.com

April 1st, 2010
Renesas Electronics Corporation

Issued by: Renesas Electronics Corporation (http://www.renesas.com)

Send any inquiries to http://www.renesas.com/inquiry.

Notice

1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.
2. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.
4. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
5. When exporting the products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations. You should not use Renesas Electronics products or the technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations.
6. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
7. Renesas Electronics products are classified according to the following three quality grades: "Standard", "High Quality", and "Specific". The recommended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below. You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application categorized as "Specific" without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for an application categorized as "Specific" or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics. The quality grade of each Renesas Electronics product is "Standard" unless otherwise expressly specified in a Renesas Electronics data sheets or data books, etc.

   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots.

   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; safety equipment; and medical equipment not specifically designed for life support.

   "Specific": Aircraft; aerospace equipment; submersible repeaters; nuclear reactor control systems; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implantations, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life.
8. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

User's Manual

R32C Simulator Debugger V.1.00

User's Manual

Renesas Microcomputer Development Environment System

Renesas Electronics
www.renesas.com

Rev.1.00 2006.09

Keep safety first in your circuit designs!

- Renesas Technology Corporation and Renesas Solutions Corporation put the maximum effort into making semiconductor products better and more reliable, but there is always the possibility that trouble may occur with them. Trouble with semiconductors may lead to personal injury, fire or property damage. Remember to give due consideration to safety when making your circuit designs, with appropriate measures such as (i) placement of substitutive, auxiliary circuits, (ii) use of nonflammable material or (iii) prevention against any malfunction or mishap.

Notes regarding these materials

- These materials are intended as a reference to assist our customers in the selection of the Renesas Technology product best suited to the customer's application; they do not convey any license under any intellectual property rights, or any other rights, belonging to Renesas Technology Corporation, Renesas Solutions Corporation or a third party.
- Renesas Technology Corporation and Renesas Solutions Corporation assume no responsibility for any damage, or infringement of any third-party's rights, originating in the use of any product data, diagrams, charts, programs, algorithms, or circuit application examples contained in these materials.
- All information contained in these materials, including product data, diagrams, charts, programs and algorithms represents information on products at the time of publication of these materials, and are subject to change by Renesas Technology Corporation and Renesas Solutions Corporation without notice due to product improvements or other reasons. It is therefore recommended that customers contact Renesas Technology Corporation, Renesas Solutions Corporation or an authorized Renesas Technology product distributor for the latest product information before purchasing a product listed herein. The information described here may contain technical inaccuracies or typographical errors. Renesas Technology Corporation and Renesas Solutions Corporation assume no responsibility for any damage, liability, or other loss rising from these inaccuracies or errors. Please also pay attention to information published by Renesas Technology Corporation and Renesas Solutions Corporation by various means, including the Renesas home page (http://www.renesas.com).
- When using any or all of the information contained in these materials, including product data, diagrams, charts, programs, and algorithms, please be sure to evaluate all information as a total system before making a final decision on the applicability of the information and products. Renesas Technology Corporation and Renesas Solutions Corporation assume no responsibility for any damage, liability or other loss resulting from the information contained herein.
- Renesas Technology semiconductors are not designed or manufactured for use in a device or system that is used under circumstances in which human life is potentially at stake. Please contact Renesas Technology Corporation, Renesas Solutions Corporation or an authorized Renesas Technology product distributor when considering the use of a product contained herein for any specific purposes, such as apparatus or systems for transportation, vehicular, medical, aerospace, nuclear, or undersea repeater use.
- The prior written approval of Renesas Technology Corporation and Renesas Solutions Corporation is necessary to reprint or reproduce in whole or in part these materials.
- If these products or technologies are subject to the Japanese export control restrictions, they must be exported under a license from the Japanese government and cannot be imported into a country other than the approved destination. Any diversion or reexport contrary to the export control laws and regulations of Japan and/or the country of destination is prohibited.
- Please contact Renesas Technology Corporation or Renesas Solutions Corporation for further details on these materials or the products contained therein.

For inquiries about the contents of this document or product, fill in the text file the installer generates in the following directory and email to your local distributor.

\SUPPORT\Product-name\SUPPORT.TXT

Renesas Tools Homepage http://www.renesas.com/en/tools

Overview

The High-performance Embedded Workshop is a Graphical User Interface intended to ease the development and debugging of applications written in C/C++ programming language and assembly language for Renesas microcomputers. Its aim is to provide a powerful yet intuitive way of accessing, observing and modifying the debugging platform in which the application is running.

This help explains the function as a "debugger" of High-performance Embedded Workshop.

Target System

The Debugger operates on the simulator system.

Supported CPU

This help explains the debugging function corresponding to the following CPUs.

- R32C/100 Series

  Note: In this help, the information which depends on this CPU is described as "for R32C".
- M32C/90, M32C/80, M16C/80, M16C/70 Series

  Note: In this help, the information which depends on this CPU is described as "for M32C".
- M16C/60, M16C/30, M16C/Tiny, M16C/20, M16C/10, R8C/Tiny Series

  Note: In this help, the information which depends on this CPU is described as "for M16C/R8C".
- 740 Family

  Note: In this help, the information which depends on this CPU is described as "for 740".

(Blank Page)

Setup of Debugger 1

Setup of Debugger

(Blank Page)

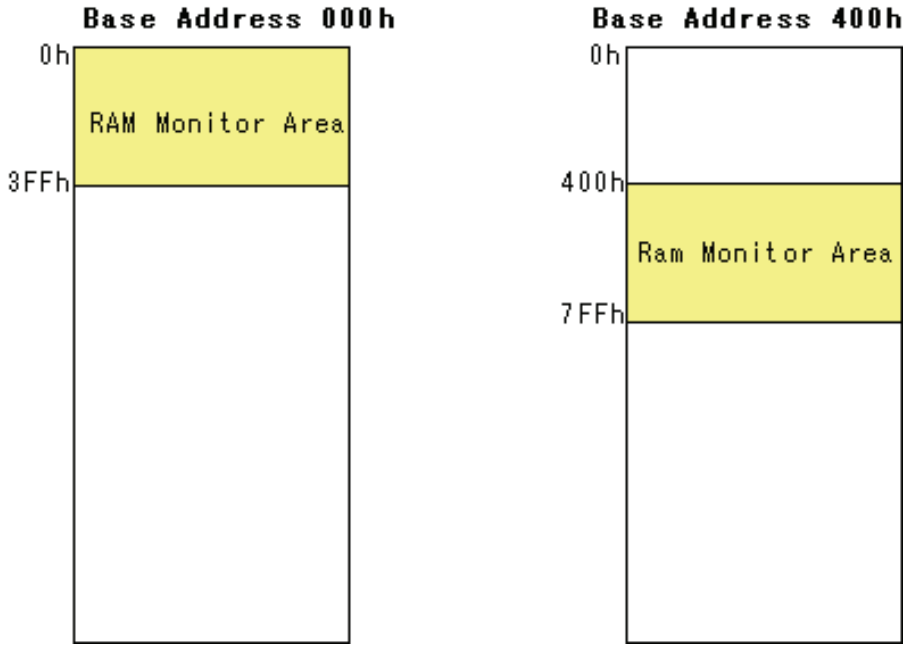
1. Features

1.1 Real-Time RAM Monitor Function

This function allows you to inspect changes of memory contents without impairing the realtime capability of target program execution.
The simulator system contains a 1-Kbyte RAM monitor area (which cannot be divided into smaller areas).

1.1.1 RAM Monitor Area

This debugger provides a 1KB of RAM monitor area, which can be placed at any continuous addresses.



1.1.2 Sampling Period

Sampling cycle means the display update interval.
You can specify this function in any window which supports the RAM monitor. (The interval of 100 ms is set by default.)
The actual sampling cycle may take longer time than the specified cycle depending on the operating environment. (Sampling cycle depends on the following environments.)

- Communication interface
- Number of the RAM Monitor windows displayed
- Size of the RAM Monitor window displayed
- Number of ASM watch points within the RAM monitor area of the ASM Watch window
- Number of C watch points within the RAM monitor area of the C Watch window

1.1.3 Related Windows

The window where the function of the real time RAM monitor function can be used is shown below.

- RAM Monitor Window
- ASM Watch Window
- C Watch Window

1.2 Break Functions

1.2.1 Software Breaks Function

Software Break breaks the target program before execution of the command at the specified address. This break point is called software breakpoint.
The software breakpoint is set/reset in the Editor (Source) window or in the S/W Breakpoint Setting window. You can also disable/enable a software breakpoint temporarily.
You can specify up to 64 software breakpoints. When specifying two or more software breakpoints, the breakpoint combination is based on the OR logic. (Arrival to any one of breakpoints breaks the target program.)

1.2.1.1 Setting of software breakpoint

The software breakpoint can be set by the following windows.

- Editor (Source) Window
- S/W Break Point Setting Window

You can double-click the mouse to set/reset the software breakpoint in the Editor (Source) window.
You can also switch to temporarily disable/enable the software breakpoint in the S/W Breakpoint Setting window.

1.2.1.2 Area where software breakpoint can be set

The area which can be set for software breakpoint varies depending on the product.
For the areas available for software breakpoint, see the following:

Refer to "12.1.2 Area where software breakpoint can be set"

1.2.2 Hardware Breaks Function

This function stops the target program upon detecting data read/writes to memory or instruction execution.

The H/W Break Point Setting dialog box allows you to set hardware break points.

- You can set up to 64 hardware break points.
- You can set one hardware breakpoints with pass counts.
- As hardware break point access types, you can specify writing data to the hardware break point (Write), reading data from the hardware break point (Read), reading or writing data (R/W), and fetching instructions (Fetch).
- You can also specify that execution breaks if the data read from or written to the hardware break point has a specific value.
- If multiple hardware breakpoints have been set, a break occurs when one of the hardware breakpoints is reached.

1.3 Real-Time Trace Function

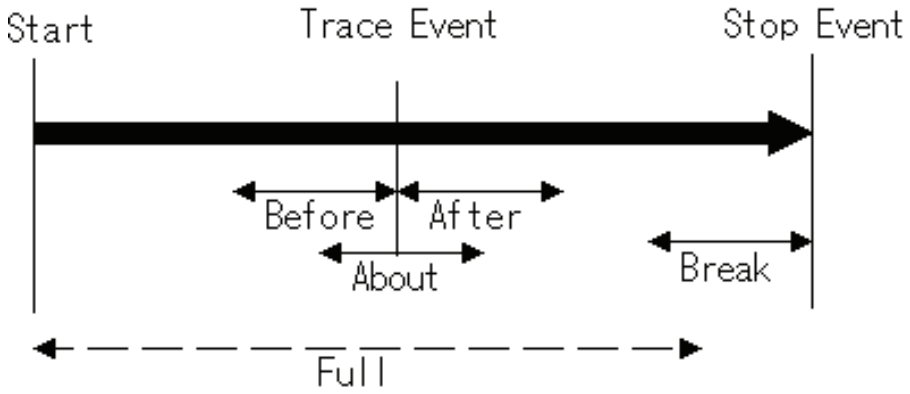
The real-time trace function records the execution history of the target program.
The debugger for 740 doesn't support the real-time trace function.
The execution history is referred to in the tracing window.
The execution history can be referred to in the following mode.

- BUS mode
  This mode allows you to inspect cycle-by-cycle bus information. The display content depends on the MCU and simulator system used. In addition to bus information, this mode allows disassemble, source line or data access information to be displayed in combination.
- Disassemble mode
  This mode allows you to inspect the executed instructions. In addition to disassemble information, this mode allows source line or data access information to be displayed in combination.
- Data access mode
  This mode allows you to inspect the data read/write cycles. In addition to data access information, this mode allows source line information to be displayed in combination.
- Source mode
  This mode allows you to inspect the program execution path in the source program.

1.3.1 Trace Area

For the simulator debugger, as many cycles as specified on the Init dialog box can be recorded. The trace area of the following 5 mode is being supported.

- Break
  the specified cycles before target program stops
- Before
  the specified cycles before trace point
- About
  the specified cycles either side of trace point
- After
  the specified cycles after trace point
- Full
  Until the specified cycles are written in the trace memory



"Break" is set by default. To refer the execution history before stopping the target program, use "Break" (designation of trace event is not required).
To refer the execution history at any position, or to continue execution of the target program, specify the trace event and change the trance range.

1.3.2 Trace Condition Setting

The following designations are available as trace events:

- Address designation
  - Instruction fetch
  - Memory access
  - Bit access
- External trigger designation (eight events)
- Interruption

The number of events that can be specified are six events of all. These break events can be combined as below:

- Trace when all of the valid events are established (AND condition)
- Trace when all of the valid events are established at the same time (And(same time) comdition)
- Trace when one of the valid events is established (OR condition)
- Trace upon entering a break state during state transition (State Transition condition)

You can select "specified task only" (or "other than specified task") as the trace condition to meet the real time OS.

1.3.3 Trace Data Write Condition

Trace data write conditions can be specified.
You can specify the following write conditions:

- Write conditions unlimited (default)
- Cycles from the start event established to the end event established
- Only cycles where the start event is established
- Cycles from the start event established to the start event unestablished
- Other than cycles from the start event established to the end event established
- Other than cycles where the start event is established
- Other than cycles from the start event established to the start event unestablished

1.4 Coverage Function

Coverage Measurement is a function to record the addresses executed (accessed) by the target program (C0 coverage).
After stopping execution of the target program, you can understand which addresses are not executed yet.
By using the coverage measurement function in the test process, you can check for missing test items.

1.4.1 Coverage Measurement Area

On simulator debugger, all the memory space is the coverage measurement area.

1.4.2 Related Windows

Refer to the coverage measurement result in the following windows.

- Editor (Source) Window
- Memory Window
- Coverage Window

1.5 Real-Time OS Debugging Function

This function debugs the realtime OS-dependent parts of the target program that uses the realtime OS.
This function helps to show the status of the realtime OS and inspect a task execution history, etc.

The debugger for R32C doesn't support the real-time OS debugging function.
The debugger for 740 doesn't support the real-time OS debugging function.

1.6 GUI Input/Output Function

This function simulates the user target system's key input panel (buttons) and output panel on a window.
Buttons can be used for the input panel, and labels (strings) and LEDs can be used for the output panel.

1.7 I/O Simulation Function

- Virtual Port Inputs

This function lets you define changes of input port data. Use the I/O Timing Setting Window to define changes of input port data. The contents thus defined can be simulated at the time:

- When the program execution reaches specified cycles
- When the program accesses a specified memory location for read
- When a specified virtual interrupt is generated

This function, when combined with the I/O script function, allows you to simulate at any time such as when the program fetches an instruction or memory is accessed for write.

- Virtual Port Outputs

This function lets you record changes of output port data and the cycles in which changes occurred. The recorded data can be verified in graphic or numeric forms on the I/O Timing Setting Window.

The number of data entries recorded is the number of entries specified on the Init dialog box's I/O Script tab reckoning from when the program started to run. When reexecuted, the previous data is cleared.

- Virtual Interrupts

This function lets you generate an interrupt (e.g., timer interrupt or key input interrupt). An interrupt can be generated at the time:

- When the program execution reaches specified cycle
- When the program fetches an instruction from a specified address
- At a specified time interval

Use the I/O Timing Setting Window to define virtual interrupts. This function, when combined with the I/O script function, allows you to generate an interrupt at any time such as when memory is accessed for write or accessed for read.

- Simulating output ports

This function lets you record changes of output port data. The recorded data can be displayed in the Output Port Window (or can be output to a file).

The number of data entries recorded is the number of entries specified on the Init dialog box's I/O Script tab reckoning from when the program started to run.

- I/O Script

This function lets you execute virtual port input or virtual interrupt settings as a script. It provides a more flexible way to define virtual port inputs and virtual interrupts than can be set from the I/O Timing Setting Window.

Example: Read a set divide-by value from the timer register and generate a timer interrupt periodically

1.8 Time Measurement Function

This function lets you measure an approximate number of cycles and the execution time spent by the program you've run. The number of cycles are represented using the values listed in the microcomputer's software manual.
The execution time is calculated from the cumulative number of cycles of the instructions executed and the MCU clock frequency specified on the Init dialog box's MCU tab.

1.9 Stack Utilization Monitor Function

This function lets you detect the maximum and minimum addresses of the stack which the target program used.

2. Simulation Specifications

The simulation specifications vary with the type of simulator use.

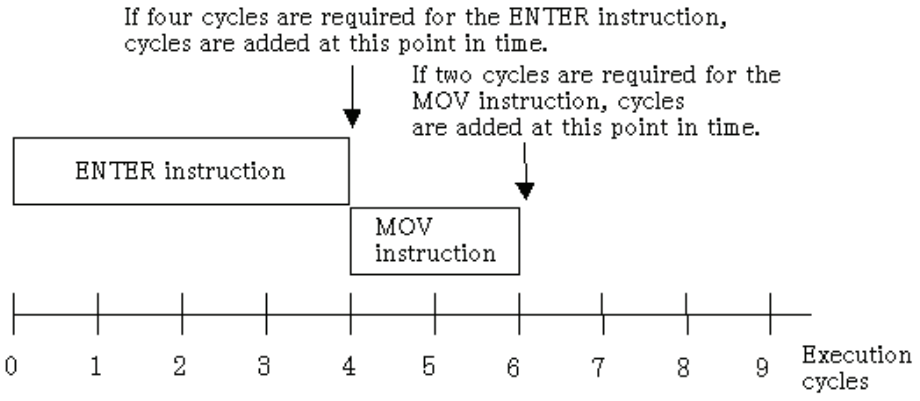
2.1 Simulation Specifications for R32C

2.1.1 Operation of Instructions

- Regarding the number of instruction cycles
  Time management is exercised in units of cycles. The number of cycles is represented by the values listed in the microcomputer's software manual.
  However, this differs from the actual chip in the following points:
  - The bus width, queue, and wait states are not considered when measuring the number of cycles.
  - The executed cycle of an interrupt sequence is not considered when measuring the number of cycles.(When an interrupt occurs, the executed cycle of an interrupt sequence is 0.)
  - This debugger starts counting cycles immediately after a reset. (Cycles immediately after a reset are 0.) The number of cycles needed to execute one machine instruction are added on for each instruction executed. (See Figure shown below.)

Note

Because the number of cycles measured by the simulator does not take into account the bus width, queue, wait cycles, etc., it includes some error when compared with the number of cycles in the actual chip.



In the above example, no cycles are added while the ENTER or MOV instruction is being executed. The cycles required for each instruction are added after instruction execution. Note that the virtual port input/output and virtual interrupt functions are processed after instruction execution is completed.

- Target program execution time measurement
  The target program execution time measurement is calculated from the number of cycles described above and the MCU clock and divide-by ratio specified on the MCU tab of the Init dialog box.

Note

Because the simulator's execution time measurement is calculated using the number of cycles described above, it includes some error when compared with the actual chip's execution time.

- STOP, WAIT, BRK2
  Executed as an NOP instruction.
  Other instructions operate the same as those of the actual MCU.
- INT, INTO, UND, BRK
  As with the actual MCU, these instructions generate interrupts. (The INTO instruction only generates an interrupt when the O flag is 1.)

2.1.2 Resetting

- The SFR area is nonexistent in the debugger, so the initialization as in the actual chip is not performed.
- The cycle count is initialized to 0.

Resetting is performed in the same way as the actual MCU.

A reset is also performed when the debugger starts. The value FFFF0000h is set in the reset vector (FFFFFFFCh to FFFFFFFFh) immediately after starting. The program counter is therefore set to FFFF0000h immediately after the debugger starts.

2.1.3 Memory

- Memory Space
  There is no processor mode. If mapped for memory, the whole 4GB of memory from 00000000h to FFFFFFFFh can be read from and written to as RAM.
  Note, however, that in the initial state, memory between 00010000h and 0003FFFFh and memory between 00050000h and FFFEFFFFh is not secured and an error will result if an attempt is made to access this part of memory. If this occurs while a program is running, the program will stop with an illegal memory access error. Use the map function, described later, to map this part of memory.

- Memory Structure and Initial Values Immediately after Starting
  The memory is set up as follows immediately after starting the debugger.

00000000h to 000003FFh (SFR1 area)	Filled with 00h.
00000400h to 0000FFFFh	Filled with FFh.
00010000h to 0003FFFFh	No memory immediately after starting
00040000h to 0004FFFFh (SFR2 area)	Filled with 00h.
00050000h to FFFEFFFFh	No memory immediately after starting
FFFF0000h to FFFFFFFBh	Filled with FFh
FFFFFFFCh to FFFFFFh (reset vector)	Set to FFFF0000h.

- The Map Function: MAP Command
  The simulator divides the memory between 00000000h and FFFFFFFFh into 65536 equal parts, so that the memory space can be mapped in 64KB blocks. Note that the blocks with the lowest address (00000000h to 0000FFFFh) and the next address (00040000h to 0004FFFFh) with the highest address (FFFF0000h to FFFFFFFFh) are already mapped when the simulator starts.
  Use the MAP command to map the simulator memory. Memory mapped using this command is initialized with the value FF16 immediately after being allocated.

When downloading a target program, the memory is mapped automatically.

Note

Memory space that has been mapped cannot be deleted.

- Accessing an Area Without Memory
  There is no actual memory in the memory blocks between 00010000h and 0003FFFFh and between 00050000h and FFFEFFFFh unless memory is secured. If an attempt is made to access this area, an illegal memory access error occurs and execution of the command or program stops.

2.1.4 I/O

- SFR
  The actual chip's peripheral I/Os other than the CPU core, such as the timers, DMAC, and serial I/O, are not supported. The SFR area to which the peripheral I/Os are connected is also handled as RAM by the simulator.
  However, a method is available that allows you to materialize data input to memory such as the SFR or interrupts such as timer interrupt in an artificial manner. For details about this method, see "I/O Script" and "Interrupts" described later.

- I/O Script
  - Virtual Port Input Function
    This function defines changes of the data that is input from external devices to a specified memory address. Using this function you can simulate data inputs to the ports defined in SFR.
    The following shows timings at which data can be input to memory:
    1. When program execution has reached a specified number of cycles
    2. When a specified memory location is accessed for read by a program
    3. When a specified virtual interrupt is generated

    Virtual interrupts at the above timings can be defined from the I/O Timing Setting Window.
    Use of the I/O script function (the function that allows users to define virtual port input or virtual interrupt) makes it possible to specify more elaborate data input timing such as when the program performs fetch or writes to memory or when it executed an instruction a specified number of times.

  - Virtual Port Output Function
    When a data write to some memory address by the program occurs, this function records the written data value and the cycle at which the data was written.
    The recorded data can be verified in graphic or numeric format from the I/O Timing Setting Window.
    The number of data entries that can be recorded by this function equals the number of data entries specified on the Init dialog box's I/O script tab reckoning from the time at which the program started running. When reexecuted, the previous data is cleared.

  - The output port simulate function
    The output port simulate function provides an efficient means of simulation. When data are written to some memory addresses by a program, it allows you to record the written data values. The recorded data can be displayed on a window or output to a file.
    Also, you can verify the data which are output to UARTs by the Printf function.
    The number of data entries that can be recorded by this function equals the number of data entries specified on the Init dialog box's I/O script tab reckoning from the time at which the program started running. When reexecuted, the previous data is cleared.

- Interrupts
  In the actual MCU, peripheral I/O (including external interrupt signals) are generating factors for interrupts. However, the simulator has nothing corresponding to peripheral I/O.
  This simulator provides another method in place of this, which allows you to generate interrupts in a simulated manner (virtual interrupt function). Virtual interrupts can be generated at any time, e.g., in a specified cycle or at an executed address.

  - Virtual Interrupt Function
    This function defines interrupt generation. Using this function you can generate timer interrupts and key input interrupts in a simulated manner without having to actually generate them.
    The following shows timings at which virtual interrupts can be generated:
    1. When program execution has reached a specified number of cycles.
    2. When the program has executed a specified address.
    3. Every specified time interval

    Virtual interrupts at the above timings can be defined from the I/O Timing Setting Window.
    Use of the I/O script function (the function that allows uses to define virtual port input or virtual interrupt) makes it possible to write timer interrupt.

  - Differences between Virtual Interrupts and Interrupts in Actual Chip
    Virtual interrupts differ from interrupts in the actual chip in the following points:
    1. Special hardware interrupts cannot be generated as virtual interrupts.
    2. High Speed interrupt cannot be generated as virtual interrupts.
    3. If virtual interrupts of the same priority occur simultaneously

       If in the actual chip, multiple interrupts of the same priority occur simultaneously,they are resolved according to the priority levels set in hardware so that an interrupt of the highest priority is accepted. For virtual interrupts, however, all interrupts belonging to one interrupt type (e.g., peripheral I/O interrupt) are handled as having the same priority. Therefore, if virtual interrupts of the same priority occur simultaneously, the order in which they are accepted is indeterminate.

    There are following two methods to set virtual interrupts.

    1. By using the I/O Timing Setting Window
    2. By using the I/O script function

With either method, the virtual interrupts are subject to the following limitations.

1. Virtual interrupts set by using the I/O Timing Setting Window
   [Regarding interrupt control for virtual interrupts generated]

- Each Interrupt Control Register's interrupt request bit is not set to 1.
- The priority levels set in each Interrupt Control Register's interrupt priority level select bit are not referenced.
  The priority of virtual interrupts can be specified when setting virtual interrupts on the I/O Timing Setting Window.
- The Flag Register (FLG)'s interrupt enable flag (I flag) and processor interrupt priority level (IPL) are referenced as in the actual chip.

2. Virtual interrupts set by using the I/O script function
   [Regarding interrupt control for virtual interrupts generated]

- A statement can be written so that when an interrupt occurs, each Interrupt Control Register's interrupt request bit is set to 1.
- The priority levels set in each Interrupt Control Register's interrupt priority level select bit can be referenced. However, once a virtual interrupt is generated and registered in the simulator, the priority of the virtual interrupt cannot be altered even when the priority levels specified with the interrupt priority level select bit is changed by the user program.
- The Flag Register (FLG)'s interrupt enable flag (I flag) and processor interrupt priority level (IPL) are referenced as in the actual chip.

- I/O Script Function
  This function allows you to write virtual port input and virtual interrupt settings to a file in script form. Therefore, it provides a more flexible way to define virtual port inputs and virtual interrupts than can be set from the I/O Timing Setting Window. Specifically, this includes, for example, reading the divide-by-N ratios you've set in the timer register and generating a timer interrupt periodically.

- Port input/output
  - GUI input function
    The GUI input function refers to simulating the user target system's simple key input panel on a window. The key input panel is created from the GUI input window.
    The input panel can have the following parts placed on it:

    [Buttons]
    Virtual port input or virtual interrupt can be performed by pressing the button. The following actions can be set for the button:
    - Enter data to a specified memory address (virtual port input)
    - Generate a specified virtual interrupt
    - Generate a specified virtual interrupt and virtual port input at the same time

    [Text]
    Display a text string.

  - GUI output function
    The GUI output function refers to simulating the user target system's simple key output panel on a window. The key output panel is created from the GUI output window.
    The following parts can be arranged on this output panel:

    [Character string]
    User-specified character strings are displayed or erased when some value is written to a specified memory address or according to logic 1 or 0 in bits.
    [LED]
    LEDs are lit when some value is written to a specified memory address or according to logic 1 or 0 in bits.
    [Text]
    Display a text string.

2.1.5 Cycle Count: The CYcle (CY) Command

Use of the CYcle command allows you to know an approximate number of cycles and the execution time of the program you've executed.
The number of cycles are represented using the values listed in the microcomputer's software manual. The execution time refers to the target program's execution time calculated from the cumulative number of cycles of the CPU instructions executed and the MCU clock and divide-by ratio specified on the Init dialog box's MCU tab.

2.1.6 Stack Utilization Monitor: The StackMonitor (SM) Command

Use the StackMonitor command to check the maximum and minimum addresses of the stack, and to determine how much the program has used of what part of the stack.
The stack monitoring continues from the time that a Go or GoFree command is invoked until it is interrupted, the maximum and minimum values being recorded for the two stack pointers (USP and ISP registers).
If, while the program is running, it causes a change in the value of a stack pointer,monitoring of stack utilization of that stack stops at that point.

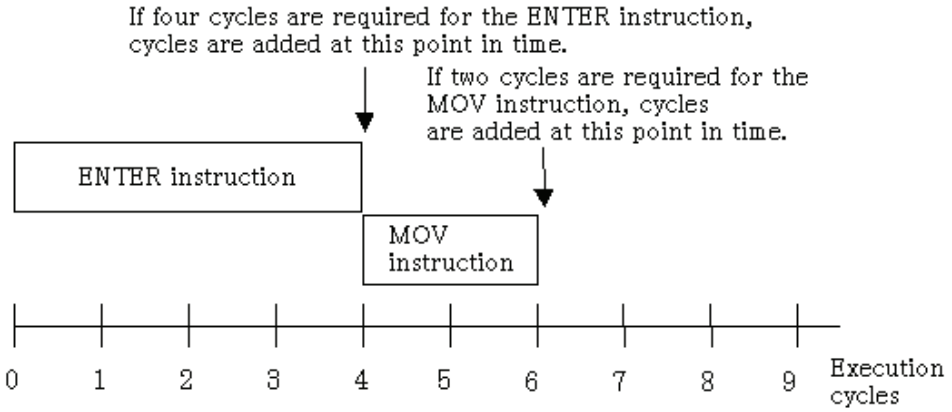
2.2 Simulation Specifications for M32C

2.2.1 Operation of Instructions

- Regarding the number of instruction cycles
  Time management is exercised in units of cycles. The number of cycles is represented by the values listed in the microcomputer's software manual.
  However, this differs from the actual chip in the following points:
  - The bus width, queue, and wait states are not considered when measuring the number of cycles.
  - The executed cycle of an interrupt sequence is not considered when measuring the number of cycles.(When an interrupt occurs, the executed cycle of an interrupt sequence is 0.)
  - This debugger starts counting cycles immediately after a reset. (Cycles immediately after a reset are 0.) The number of cycles needed to execute one machine instruction are added on for each instruction executed. (See Figure shown below.)

Note

Because the number of cycles measured by the simulator does not take into account the bus width, queue, wait cycles, etc., it includes some error when compared with the number of cycles in the actual chip.



In the above example, no cycles are added while the ENTER or MOV instruction is being executed. The cycles required for each instruction are added after instruction execution. Note that the virtual port input/output and virtual interrupt functions are processed after instruction execution is completed.

- Target program execution time measurement
  The target program execution time measurement is calculated from the number of cycles described above and the MCU clock and divide-by ratio specified on the MCU tab of the Init dialog box.

Note

Because the simulator's execution time measurement is calculated using the number of cycles described above, it includes some error when compared with the actual chip's execution time.

- WAIT, BRK2
  Executed as an NOP instruction.
  Other instructions operate the same as those of the actual MCU.
- INT, INTO, UND, BRK
  As with the actual MCU, these instructions generate interrupts. (The INTO instruction only generates an interrupt when the O flag is 1.)

2.2.2 Resetting

- The SFR area is nonexistent in the debugger, so the initialization as in the actual chip is not performed.
- The cycle count is initialized to 0.

Resetting is performed in the same way as the actual MCU.

A reset is also performed when the debugger starts. The value FF0000h is set in the reset vector (FFFFFCh to FFFFFFh) immediately after starting. The program counter is therefore set to FF0000h immediately after the debugger starts.

2.2.3 Memory

- Memory Space
  There is no processor mode. If mapped for memory, the whole 16MB of memory from 000000h to FFFFFFh can be read from and written to as RAM.
  Note, however, that in the initial state, memory between 020000h and FEFFFFh is not secured and an error will result if an attempt is made to access this part of memory. If this occurs while a program is running, the program will stop with an illegal memory access error. Use the map function, described later, to map this part of memory.

- Memory Structure and Initial Values Immediately after Starting
  The memory is set up as follows immediately after starting the debugger.

000000h to 0003FFh (SFR area)	Filled with 00h.
000400h to 01FFFFh	Filled with FFh.
020000h to FEFFFFh	No memory immediately after starting
FF0000h to FFFFFBh	Filled with FFh
FFFFFCh to FFFFFFh (reset vector)	Set to 0000FF00h.

- The Map Function: MAP Command
  The simulator divides the memory between 000000h and FFFFFFh into 256 equal parts, so that the memory space can be mapped in 64KB blocks. Note that the blocks with the lowest address (000000h to 00FFFFh) and the next address (010000h to 01FFFFh) with the highest address (FF0000h to FFFFFFh) are already mapped when the simulator starts.
  Use the MAP command to map the simulator memory. Memory mapped using this command is initialized with the value FF16 immediately after being allocated.

When downloading a target program, the memory is mapped automatically.

Note

Memory space that has been mapped cannot be deleted.

- Accessing an Area Without Memory
  There is no actual memory in the 253 memory blocks between 020000h and FEFFFFh unless memory is secured. If an attempt is made to access this area, an illegal memory access error occurs and execution of the command or program stops.

2.2.4 I/O

- SFR
  The actual chip's peripheral I/Os other than the CPU core, such as the timers, DMAC, and serial I/O, are not supported. The SFR area (000000h to 0003FFh) to which the peripheral I/Os are connected is also handled as RAM by the simulator.
  However, a method is available that allows you to materialize data input to memory such as the SFR or interrupts such as timer interrupt in an artificial manner. For details about this method, see "I/O Script" and "Interrupts" described later.

- I/O Script
  - Virtual Port Input Function
    This function defines changes of the data that is input from external devices to a specified memory address. Using this function you can simulate data inputs to the ports defined in SFR.
    The following shows timings at which data can be input to memory:
    1. When program execution has reached a specified number of cycles
    2. When a specified memory location is accessed for read by a program
    3. When a specified virtual interrupt is generated

    Virtual interrupts at the above timings can be defined from the I/O Timing Setting Window.
    Use of the I/O script function (the function that allows users to define virtual port input or virtual interrupt) makes it possible to specify more elaborate data input timing such as when the program performs fetch or writes to memory or when it executed an instruction a specified number of times.

  - Virtual Port Output Function
    When a data write to some memory address by the program occurs, this function records the written data value and the cycle at which the data was written.
    The recorded data can be verified in graphic or numeric format from the I/O Timing Setting Window.
    The number of data entries that can be recorded by this function equals the number of data entries specified on the Init dialog box's I/O script tab reckoning from the time at which the program started running. When reexecuted, the previous data is cleared.

  - The output port simulate function
    The output port simulate function provides an efficient means of simulation. When data are written to some memory addresses by a program, it allows you to record the written data values. The recorded data can be displayed on a window or output to a file.
    Also, you can verify the data which are output to UARTs by the Printf function.
    The number of data entries that can be recorded by this function equals the number of data entries specified on the Init dialog box's I/O script tab reckoning from the time at which the program started running. When reexecuted, the previous data is cleared.

- Interrupts
  In the actual MCU, peripheral I/O (including external interrupt signals) are generating factors for interrupts. However, the simulator has nothing corresponding to peripheral I/O.
  This simulator provides another method in place of this, which allows you to generate interrupts in a simulated manner (virtual interrupt function). Virtual interrupts can be generated at any time, e.g., in a specified cycle or at an executed address.

  - Virtual Interrupt Function
    This function defines interrupt generation. Using this function you can generate timer interrupts and key input interrupts in a simulated manner without having to actually generate them.
    The following shows timings at which virtual interrupts can be generated:
    1. When program execution has reached a specified number of cycles.
    2. When the program has executed a specified address.
    3. Every specified time interval

    Virtual interrupts at the above timings can be defined from the I/O Timing Setting Window.
    Use of the I/O script function (the function that allows uses to define virtual port input or virtual interrupt) makes it possible to write timer interrupt.

  - Differences between Virtual Interrupts and Interrupts in Actual Chip
    Virtual interrupts differ from interrupts in the actual chip in the following points:
    1. Special hardware interrupts cannot be generated as virtual interrupts.
       Reset, NMI, DBC, watchdog timer, single-step, address match interrupts cannot be generated as virtual interrupts.
    2. High Speed interrupt cannot be generated as virtual interrupts.
    3. If virtual interrupts of the same priority occur simultaneously

       If in the actual chip, multiple interrupts of the same priority occur simultaneously,they are resolved according to the priority levels set in hardware so that an interrupt of the highest priority is accepted. For virtual interrupts, however, all interrupts belonging to one interrupt type (e.g., peripheral I/O interrupt) are handled as having the same priority. Therefore, if virtual interrupts of the same priority occur simultaneously, the order in which they are accepted is indeterminate.

    There are following two methods to set virtual interrupts.

    1. By using the I/O Timing Setting Window
    2. By using the I/O script function

With either method, the virtual interrupts are subject to the following limitations.

1. Virtual interrupts set by using the I/O Timing Setting Window
   [Regarding interrupt control for virtual interrupts generated]

- Each Interrupt Control Register's interrupt request bit is not set to 1.
- The priority levels set in each Interrupt Control Register's interrupt priority level select bit are not referenced.
  The priority of virtual interrupts can be specified when setting virtual interrupts on the I/O Timing Setting Window.
- The Flag Register (FLG)'s interrupt enable flag (I flag) and processor interrupt priority level (IPL) are referenced as in the actual chip.

2. Virtual interrupts set by using the I/O script function
   [Regarding interrupt control for virtual interrupts generated]

- A statement can be written so that when an interrupt occurs, each Interrupt Control Register's interrupt request bit is set to 1.
- The priority levels set in each Interrupt Control Register's interrupt priority level select bit can be referenced. However, once a virtual interrupt is generated and registered in the simulator, the priority of the virtual interrupt cannot be altered even when the priority levels specified with the interrupt priority level select bit is changed by the user program.
- The Flag Register (FLG)'s interrupt enable flag (I flag) and processor interrupt priority level (IPL) are referenced as in the actual chip.

- I/O Script Function
  This function allows you to write virtual port input and virtual interrupt settings to a file in script form. Therefore, it provides a more flexible way to define virtual port inputs and virtual interrupts than can be set from the I/O Timing Setting Window. Specifically, this includes, for example, reading the divide-by-N ratios you've set in the timer register and generating a timer interrupt periodically.

- Port input/output
  - GUI input function
    The GUI input function refers to simulating the user target system's simple key input panel on a window. The key input panel is created from the GUI input window.
    The input panel can have the following parts placed on it:

    [Buttons]
    Virtual port input or virtual interrupt can be performed by pressing the button. The following actions can be set for the button:
    - Enter data to a specified memory address (virtual port input)
    - Generate a specified virtual interrupt
    - Generate a specified virtual interrupt and virtual port input at the same time

    [Text]
    Display a text string.

  - GUI output function
    The GUI output function refers to simulating the user target system's simple key output panel on a window. The key output panel is created from the GUI output window.
    The following parts can be arranged on this output panel:

    [Character string]
    User-specified character strings are displayed or erased when some value is written to a specified memory address or according to logic 1 or 0 in bits.
    [LED]
    LEDs are lit when some value is written to a specified memory address or according to logic 1 or 0 in bits.
    [Text]
    Display a text string.

2.2.5 Cycle Count: The CYcle (CY) Command

Use of the CYcle command allows you to know an approximate number of cycles and the execution time of the program you've executed.
The number of cycles are represented using the values listed in the microcomputer's software manual. The execution time refers to the target program's execution time calculated from the cumulative number of cycles of the CPU instructions executed and the MCU clock and divide-by ratio specified on the Init dialog box's MCU tab.

2.2.6 Stack Utilization Monitor: The StackMonitor (SM) Command

Use the StackMonitor command to check the maximum and minimum addresses of the stack, and to determine how much the program has used of what part of the stack.
The stack monitoring continues from the time that a Go or GoFree command is invoked until it is interrupted, the maximum and minimum values being recorded for the two stack pointers (USP and ISP registers).
If, while the program is running, it causes a change in the value of a stack pointer,monitoring of stack utilization of that stack stops at that point.

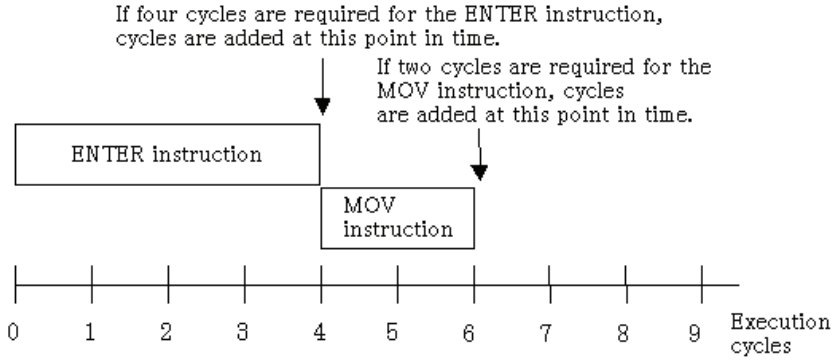
2.3 Simulation Specifications for M16C/R8C

2.3.1 Operation of Instructions

- Regarding the number of instruction cycles
  Time management is exercised in units of cycles. The number of cycles is represented by the values listed in the microcomputer's software manual.
  However, this differs from the actual chip in the following points:
  - The bus width, queue, and wait states are not considered when measuring the number of cycles.
  - The executed cycle of an interrupt sequence is not considered when measuring the number of cycles.(When an interrupt occurs, the executed cycle of an interrupt sequence is 0.)
  - This debugger starts counting cycles immediately after a reset. (Cycles immediately after a reset are 0.) The number of cycles needed to execute one machine instruction are added on for each instruction executed. (See Figure shown below.)

Note

Because the number of cycles measured by the emulator does not take into account the bus width, queue, wait cycles, etc., it includes some error when compared with the number of cycles in the actual chip.



In the above example, no cycles are added while the ENTER or MOV instruction is being executed. The cycles required for each instruction are added after instruction execution. Note that the virtual port input/output and virtual interrupt functions are processed after instruction execution is completed.

- Target program execution time measurement
  The target program execution time measurement is calculated from the number of cycles described above and the MCU clock and divide-by ratio specified on the MCU tab of the Init dialog box.

Note

Because the simulator's execution time measurement is calculated using the number of cycles described above, it includes some error when compared with the actual chip's execution time.

- WAIT
  Executed as an NOP instruction.
  Other instructions operate the same as those of the actual MCU.
- INT, INTO, UND, BRK
  As with the actual MCU, these instructions generate interrupts. (The INTO instruction only generates an interrupt when the O flag is 1.)

2.3.2 Resetting

- The SFR area is nonexistent in the debugger, so the initialization as in the actual chip is not performed.
- The cycle count is initialized to 0.

Resetting is performed in the same way as the actual MCU.
A reset is also performed when the debugger starts. The value 000F000016 is set in the reset vector immediately after starting. The program counter is therefore set to F000016 immediately after the debugger starts.

2.3.3 Memory

- Memory Space
  There is no processor mode. If mapped for memory, the whole 1MB of memory from 00000h to FFFFFh can be read from and written to as RAM.
  Note, however, that in the initial state, memory between 10000h and EFFFFh is not secured and an error will result if an attempt is made to access this part of memory. If this occurs while a program is running, the program will stop with an illegal memory access error. Use the map function, described later, to map this part of memory.
- Memory Structure and Initial Values Immediately after Starting
  The memory is set up as follows immediately after starting the debugger.

00000h to 003FFh	Filled with 00h.
00400h to 0FFFFh	Filled with FFh.
10000h to EFFFFh	No memory immediately after starting
F0000h to FFFFFh	Filled with FFh.
Reset Vector	Set to 000F0000h.

- The Map Function: MAP Command
  The simulator divides the memory between 00000h and FFFFFh into sixteen equal parts, so that the memory space can be mapped in 64KB blocks. Note that the blocks with the lowest address (00000h to 0FFFFh) and with the highest address (F0000h to FFFFFh) are already mapped when the simulator starts.
  Use the MAP command to map the simulator memory. Memory mapped using this command is initialized with the value FFh immediately after being allocated.

When downloading a target program, the memory is mapped automatically.

Note

Memory space that has been mapped cannot be deleted.

- Accessing an Area Without Memory
  There is no actual memory in the 14 memory blocks between 10000h and EFFFFh unless memory is secured. If an attempt is made to access this area, an illegal memory access error occurs and execution of the command or program stops.

2.3.4 I/O

- SFR
  The actual chip's peripheral I/Os other than the CPU core, such as the timers, DMAC, and serial I/O, are not supported. The SFR area to which the peripheral I/Os are connected is also handled as RAM by the simulator.
  However, a method is available that allows you to materialize data input to memory such as the SFR or interrupts such as timer interrupt in an artificial manner. For details about this method, see "I/O Script" and "Interrupts" described later.

- I/O Script
  - Virtual Port Input Function
    This function defines changes of the data that is input from external devices to a specified memory address. Using this function you can simulate data inputs to the ports defined in SFR.
    The following shows timings at which data can be input to memory:

    1. When program execution has reached a specified number of cycles
    2. When a specified memory location is accessed for read by a program
    3. When a specified virtual interrupt is generated

    Virtual interrupts at the above timings can be defined from the I/O Timing Setting Window.
    Use of the I/O script function (the function that allows users to define virtual port input or virtual interrupt) makes it possible to specify more elaborate data input timing such as when the program performs fetch or writes to memory or when it executed an instruction a specified number of times.

  - Virtual Port Output Function
    When a data write to some memory address by the program occurs, this function records the written data value and the cycle at which the data was written.
    The recorded data can be verified in graphic or numeric format from the I/O Timing Setting Window.
    The number of data entries that can be recorded by this function equals the number of data entries specified on the Init dialog box's I/O script tab reckoning from the time at which the program started running. When reexecuted, the previous data is cleared.

  - The output port simulate function
    The output port simulate function provides an efficient means of simulation. When data are written to some memory addresses by a program, it allows you to record the written data values. The recorded data can be displayed on a window or output to a file.
    Also, you can verify the data which are output to UARTs by the Printf function.
    The number of data entries that can be recorded by this function equals the number of data entries specified on the Init dialog box's I/O script tab reckoning from the time at which the program started running. When reexecuted, the previous data is cleared.

- Interrupts

  In the actual MCU, peripheral I/O (including external interrupt signals) are generating factors for interrupts. However, the simulator has nothing corresponding to peripheral I/O.

  This simulator provides another method in place of this, which allows you to generate interrupts in a simulated manner (virtual interrupt function). Virtual interrupts can be generated at any time, e.g., in a specified cycle or at an executed address.

  - Virtual Interrupt Function

    This function defines interrupt generation. Using this function you can generate timer interrupts and key input interrupts in a simulated manner without having to actually generate them.

    The following shows timings at which virtual interrupts can be generated:

    1. When program execution has reached a specified number of cycles.
    2. When the program has executed a specified address.
    3. Every specified time interval

    Virtual interrupts at the above timings can be defined from the I/O Timing Setting Window.

    Use of the I/O script function (the function that allows uses to define virtual port input or virtual interrupt) makes it possible to write timer interrupt.

  - Differences between Virtual Interrupts and Interrupts in Actual Chip

    Virtual interrupts differ from interrupts in the actual chip in the following points:

    1. Special hardware interrupts cannot be generated as virtual interrupts.

       Reset, NMI, DBC, watchdog timer, single-step, address match interrupts cannot be generated as virtual interrupts.

    2. If virtual interrupts of the same priority occur simultaneously

       If in the actual chip, multiple interrupts of the same priority occur simultaneously,they are resolved according to the priority levels set in hardware so that an interrupt of the highest priority is accepted. For virtual interrupts, however, all interrupts belonging to one interrupt type (e.g., peripheral I/O interrupt) are handled as having the same priority. Therefore, if virtual interrupts of the same priority occur simultaneously, the order in which they are accepted is indeterminate.

    There are following two methods to set virtual interrupts.

    1. By using the I/O Timing Setting Window
    2. By using the I/O script function

With either method, the virtual interrupts are subject to the following limitations.

1. Virtual interrupts set by using the I/O Timing Setting Window
   [Regarding interrupt control for virtual interrupts generated]
   - Each Interrupt Control Register's interrupt request bit is not set to 1.
   - The priority levels set in each Interrupt Control Register's interrupt priority level select bit are not referenced.
     The priority of virtual interrupts can be specified when setting virtual interrupts on the I/O Timing Setting Window.
   - The Flag Register (FLG)'s interrupt enable flag (I flag) and processor interrupt priority level (IPL) are referenced as in the actual chip.
2. Virtual interrupts set by using the I/O script function
   [Regarding interrupt control for virtual interrupts generated]
   - A statement can be written so that when an interrupt occurs, each Interrupt Control Register's interrupt request bit is set to 1.
   - The priority levels set in each Interrupt Control Register's interrupt priority level select bit can be referenced. However, once a virtual interrupt is generated and registered in the simulator, the priority of the virtual interrupt cannot be altered even when the priority levels specified with the interrupt priority level select bit is changed by the user program.
   - The Flag Register (FLG)'s interrupt enable flag (I flag) and processor interrupt priority level (IPL) are referenced as in the actual chip.

- I/O Script Function
  This function allows you to write virtual port input and virtual interrupt settings to a file in script form. Therefore, it provides a more flexible way to define virtual port inputs and virtual interrupts than can be set from the I/O Timing Setting Window. Specifically, this includes, for example, reading the divide-by-N ratios you've set in the timer register and generating a timer interrupt periodically.

- Port input/output
  - GUI input function
    The GUI input function refers to simulating the user target system's simple key input panel on a window. The key input panel is created from the GUI input window.
    The input panel can have the following parts placed on it:

    [Buttons]
    Virtual port input or virtual interrupt can be performed by pressing the button. The following actions can be set for the button:
    - Enter data to a specified memory address (virtual port input)
    - Generate a specified virtual interrupt
    - Generate a specified virtual interrupt and virtual port input at the same time

    [Text]
    Display a text string.

  - GUI output function
    The GUI output function refers to simulating the user target system's simple key output panel on a window. The key output panel is created from the GUI output window.
    The following parts can be arranged on this output panel:

    [Character string]
    User-specified character strings are displayed or erased when some value is written to a specified memory address or according to logic 1 or 0 in bits.
    [LED]
    LEDs are lit when some value is written to a specified memory address or according to logic 1 or 0 in bits.
    [Text]
    Display a text string.

2.3.5 Cycle Count: The CYcle (CY) Command

Use of the CYcle command allows you to know an approximate number of cycles and the execution time of the program you've executed.
The number of cycles are represented using the values listed in the microcomputer's software manual. The execution time refers to the target program's execution time calculated from the cumulative number of cycles of the CPU instructions executed and the MCU clock and divide-by ratio specified on the Init dialog box's MCU tab.

2.3.6 Stack Utilization Monitor: The StackMonitor (SM) Command

Use the StackMonitor command to check the maximum and minimum addresses of the stack, and to determine how much the program has used of what part of the stack.
The stack monitoring continues from the time that a Go or GoFree command is invoked until it is interrupted, the maximum and minimum values being recorded for the two stack pointers (USP and ISP registers).
If, while the program is running, it causes a change in the value of a stack pointer,monitoring of stack utilization of that stack stops at that point.

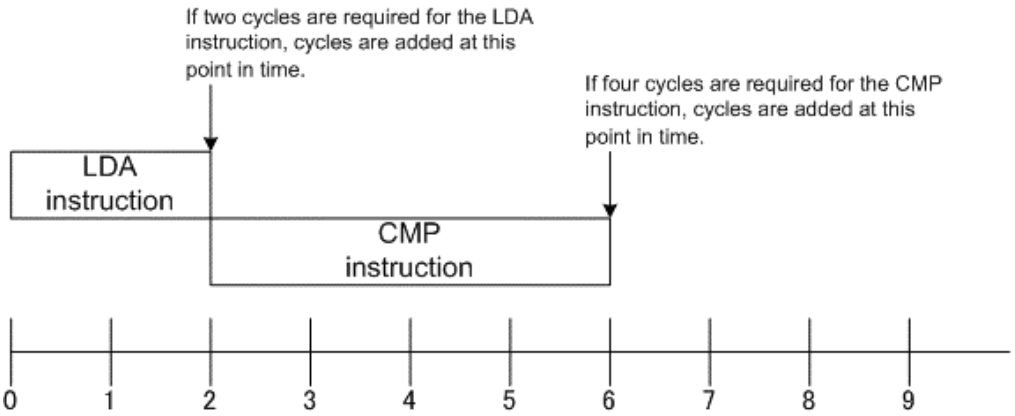
2.4 Simulation Specifications for 740

2.4.1 Operation of Instructions

- Regarding the number of instruction cycles
  Time management is exercised in units of cycles. The number of cycles is represented by the values listed in the microcomputer's software manual.
  However, this differs from the actual chip in the following points:
  - The 740 debugger starts counting cycles immediately after a reset. (Cycles immediately after a reset are 0.) The number of cycles needed to execute one machine instruction are added on for each instruction executed. (See Figure shown below.)

Note

Because the number of cycles measured by the emulator does not take into account the bus width, queue, wait cycles, etc., it includes some error when compared with the number of cycles in the actual chip.



In the above example, no cycles are added while the LDA or CMP instruction is being executed. The cycles required for each instruction are added after instruction execution. Note that the virtual port input/output and virtual interrupt functions are processed after instruction execution is completed.
Target program execution time measurement
The target program execution time measurement is calculated from the number of cycles described above and the MCU clock and divide-by ratio specified on the MCU tab of the Init dialog box.

Note

Because the simulator's execution time measurement is calculated using the number of cycles described above, it includes some error when compared with the actual chip's execution time.

- WIT,STP
  Executed as an NOP instruction.
  Other instructions operate the same as those of the actual MCU.

2.4.2 Resetting

- S register is set to FF16, only 'I' flag is set to 1 about PS register, and Program Counter is set to the value of the reset vector. Other registers are set to 0.
- SFR area is not initialized.
- Executed cycle count is set to 0.

Resetting is performed in the same way as the actual MCU. A reset is also performed when the debugger of 740 starts. The value 0000h is set in the reset vector immediately after starting. The program counter is therefore set to 0000h immediately after the debugger of 740 starts.

2.4.3 Memory

- Memory Space
  The entire memory space (0000h to FFFFh) functions as RAM. At startup, memory is allocated for the entire area.
- Memory Structure and Initial Values Immediately after Starting
  The memory is set up as follows immediately after starting the debugger.

0000h to FFFFh	Filled with 00h

2.4.4 I/O

- SFR
  No peripheral I/Os such as timers and serial I/O in the actual chip are supported. Only the CPU core is supported. Consequently, the 740 debugger handles the SFR area to which peripheral I/Os are connected simply as RAM.
  However, the CPU Mode Register's stack page select bit and the Interrupt Control Register each are handled as SFR. (For details about each register location, refer to the user's manual of your chip.) When the stack page select bit is set to 1, one page of RAM can be used as a stack area. When a bit of the Interrupt Control Register is set to 1, the corresponding interrupt of that bit is enabled. Moreover, a method is provided that allows you to accomplish interrupts, e.g., a timer interrupt, and data input to SRF or other memory by simulating these operations without having to actually execute them. For details about this method, refer to the virtual port input/output and virtual interrupt functions described later in this manual.

- I/O Script
  - Virtual Port Input Function
    This function defines changes of the data that is input from external devices to a specified memory address. Using this function you can simulate data inputs to the ports defined in SFR. The following shows timings at which data can be input to memory:
    1. When program execution has reached a specified number of cycles
    2. When a specified memory location is accessed for read by a program
    3. When a specified virtual interrupt is generated

    The input data at the above timings can be defined from the I/O Window. Furthermore, this function can be used in combination with the I/O script function, a function that allows you to define virtual port inputs and virtual interrupts. (For details, refer to "High-end Debugging" described later in this manual.) Using this I/O script function, you can specify more precise data input timings such as when the program fetches an instruction, when the program writes to memory, or when the program has executed instructions a specified number of times.

  - Virtual Port Output Function
    When a data write to some memory address by the program occurs, this function records the written data value and the cycle at which the data was written. The recorded data can be verified in graphic or numeric format from the I/O Window. The maximum number of data that can be recorded by this function is 30,000 entries counted from the beginning of program execution.

- Interrupts

  This function defines interrupt generation. Using this function you can generate timer interrupts and key input interrupts in a simulated manner without having to actually generate them. The following shows timings at which virtual interrupts can be generated:

  - Virtual Interrupt Function

    This function defines interrupt generation. Using this function you can generate timer interrupts and key input interrupts in a simulated manner without having to actually generate them.

    The following shows timings at which virtual interrupts can be generated:

    1. When program execution has reached a specified number of cycles.
    2. When the program has executed a specified address.
    3. Every specified time interval

    Virtual interrupts at the above timings can be defined from the I/O Timing Setting Window. Furthermore, this function can be used in combination with the I/O script function, a function that allows you to define virtual port inputs and virtual interrupts. (For details, refer to "High-end Debugging" described later in this manual.) Using this I/O script function, you can specify more precise interrupt generation timings such as when the program reads or writes to memory or when the program has executed instructions a specified number of times.

  - Differences between Virtual Interrupts and Interrupts in Actual Chip

    Virtual interrupts differ from interrupts in the actual chip in the following points:

    About the Interrupt Control and Interrupt Request Registers

    When a virtual interrupt is generated, PD38SIM looks up the Interrupt Control Register's interrupt control bit as it simulates virtual interrupt generation. If a virtual interrupt occurs when interrupt generation is disabled, the interrupt request is saved inside the simulator so that a virtual interrupt is generated after interrupt generation is enabled. However, since the interrupt request bit is not simulated, the interrupt request bit is not set even when an interrupt request is saved. Nor can the virtual interrupts that have been saved be deleted by clearing the interrupt request bit. (The virtual interrupts saved in the simulator are deleted when the device is reset.) Note that you can use the I/O script function to write a statement to the effect that the interrupt request bit is set when an interrupt occurs.

    Reset interrupts cannot be generated.

    There are following two methods to set virtual interrupts.

    1. By using the I/O Timing Setting Window
    2. By using the I/O script function

  - I/O Script Function

    This function allows you to write virtual port input and virtual interrupt settings to a file in script form. Therefore, it provides a more flexible way to define virtual port inputs and virtual interrupts than can be set from the I/O Timing Setting Window. Specifically, this includes, for example, reading the divide-by-N ratios you've set in the timer register and generating a timer interrupt periodically.

- Port input/output
  - GUI input function
    The GUI input function refers to simulating the user target system's simple key input panel on a window. The key input panel is created from the GUI input window.
    The input panel can have the following parts placed on it:
    [Buttons]
    Virtual port input or virtual interrupt can be performed by pressing the button. The following actions can be set for the button:
    - Enter data to a specified memory address (virtual port input)
    - Generate a specified virtual interrupt
    - Generate a specified virtual interrupt and virtual port input at the same time

    [Text]
    Display a text string.

  - GUI output function
    The GUI output function refers to simulating the user target system's simple key output panel on a window. The key output panel is created from the GUI output window.
    The following parts can be arranged on this output panel:
    [Character string]
    User-specified character strings are displayed or erased when some value is written to a specified memory address or according to logic 1 or 0 in bits.
    [LED]
    LEDs are lit when some value is written to a specified memory address or according to logic 1 or 0 in bits.
    [Text]
    Display a text string.

2.4.5 Cycle Count: The CYcle (CY) Command

Use of the CYcle command allows you to know an approximate number of cycles and the execution time of the program you've executed.
The number of cycles are represented using the values listed in the microcomputer's software manual. The execution time refers to the target program's execution time calculated from the cumulative number of cycles of the CPU instructions executed and the MCU clock and divide-by ratio specified on the Init dialog box's MCU tab.

2.4.6 Stack Utilization Monitor: The StackMonitor (SM) Command

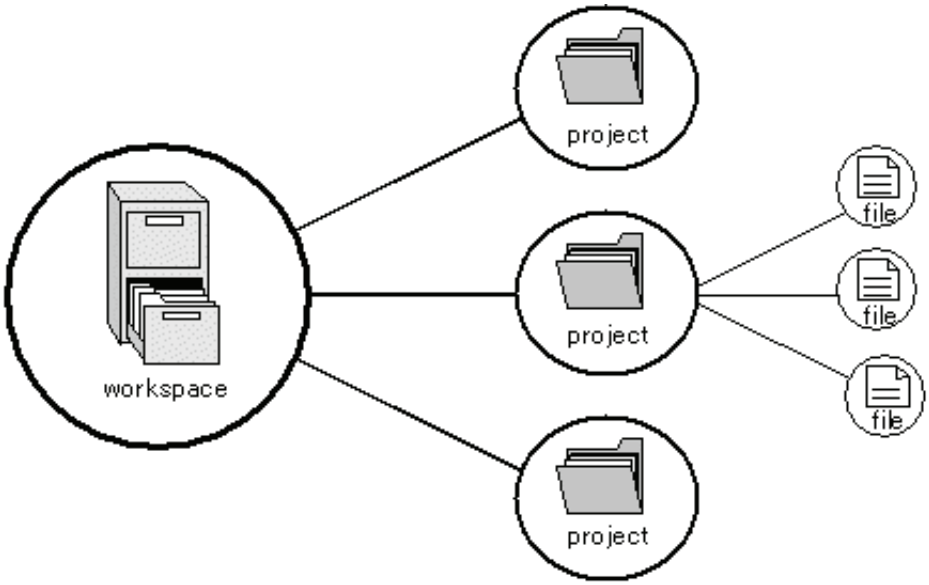
Use the StackMonitor command to check the maximum and minimum addresses of the stack, and to determine how much the program has used of what part of the stack.
The stack monitoring continues from the time that a Go or GoFree command is invoked until it is interrupted, the maximum and minimum values being recorded for the two stack pointers (USP and ISP registers).
If, while the program is running, it causes a change in the value of a stack pointer,monitoring of stack utilization of that stack stops at that point.

3. Preparation before Use

Please run the High-performance Embedded Workshop and connect the emulator .
In addition, in order to debug with this product, it is necessary to create a workspace.

3.1 Workspaces, Projects, and Files

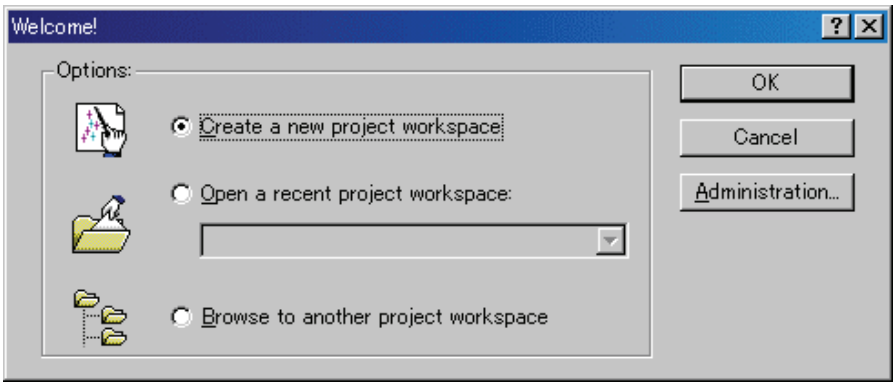
Just as a word processor allows you to create and modify documents, this product allows you to create and modify workspaces.
A workspace can be thought of as a container of projects and, similarly, a project can be thought of as a container of project files. Thus, each workspace contains one or more projects and each project contains one or more files.



Workspaces allow you to group related projects together. For example, you may have an application that needs to be built for different processors or you may be developing an application and library at the same time. Projects can also be linked hierarchically within a workspace, which means that when one project is built all of its "child" projects are built first.
However, workspaces on their own are not very useful, we need to add a project to a workspace and then add files to that project before we can actually do anything.

3.2 Starting the High-performance Embedded Workshop

Activate the High-performance Embedded Workshop from [Programs] in the [Start] menu.
The [Welcome!] dialog box is displayed.



In this dialog box, A workspace is created or displayed.

- [Create a new project workspace] radio button:
  Creates a new workspace.
- [Open a recent project workspace] radio button:
  Uses an existing workspace and displays the history of the opened workspace.
- [Browse to another project workspace] radio button:
  Uses an existing workspace;
  this radio button is used when the history of the opened workspace does not remain.

In the case of Selecting an Existing Workspace, select [Open a recent project workspace] or [Browse to another project workspace] radio button and select the workspace file (.hws).

Please refer to the following about the method to create a new workspace.

- Refer to "3.2.1 Creating a New Workspace (Toolchain Used)"
- Refer to "3.2.2Creating a New Workspace (Toolchain Not Used)"

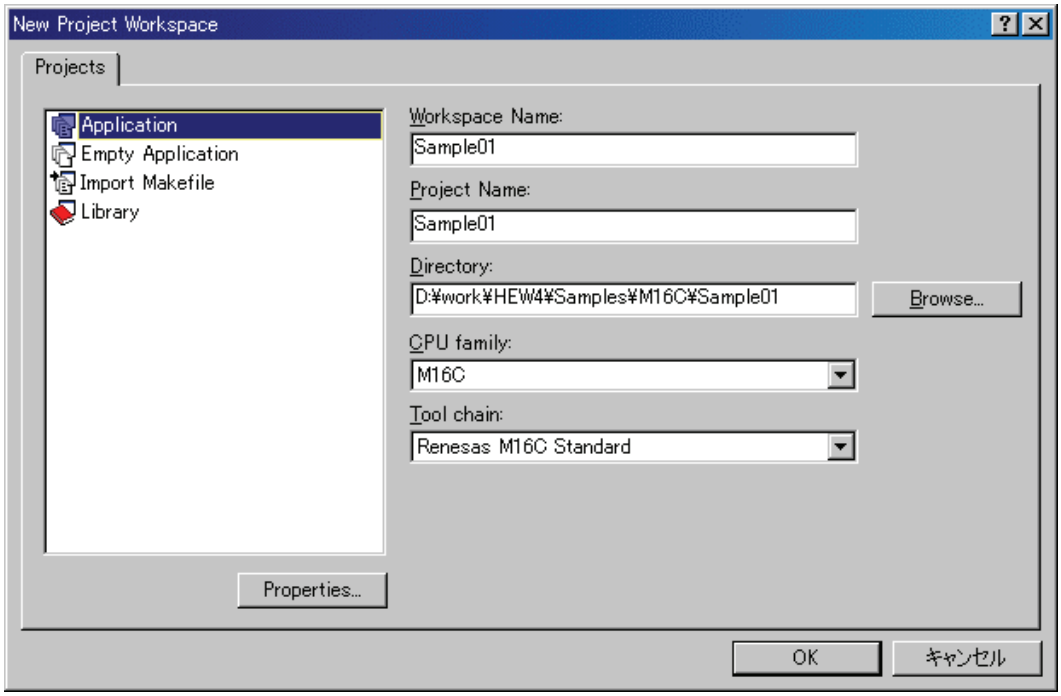
* When debugging the existing load module file with this product, a workspace is created by this method.

The method to create a new workspace depends on whether a toolchain is or is not in use. Note that this product does not include a toolchain. Use of a toolchain is available in an environment where the C/C++ compiler package for the CPU which you are using has been installed.
For details on this, refer to the manual attached to your C/C++ compiler package.

3.2.1 Creating a New Workspace (Toolchain Used)

3.2.1.1 Step1 : Creation of a new workspace

In the [Welcome!] dialog box that is displayed when the High-performance Embedded Workshop is activated, select the [Create a new project workspace] radio button and click the [OK] button.
Creation of a new workspace is started.
The following dialog box is displayed.

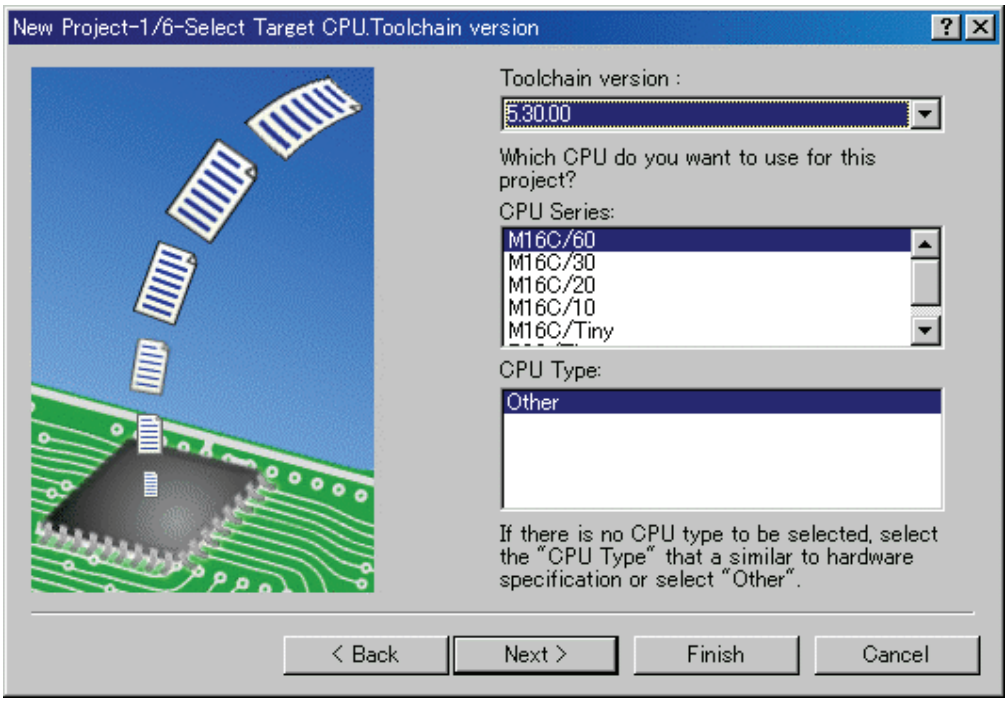


1. Select the target CPU family
   In the [CPU family] combo box, select the target CPU family.
2. Select the target toolchain
   In the [Tool chain] combo box, select the target toolchain name when using the toolchain.
3. Select the project type
   In the [Project type] list box, select the project type to be used.
   In this case, select "Application" .
   (Please refer to the manual attached to your C/C++ compiler package about the details of the project type which can be chosen.)
4. Specify the workspace name and project name
   - In the [Workspace Name] edit box, enter the new workspace name.
   - In the [Project Name] edit box, enter the project name. When the project name is the same as the workspace name, it needs not be entered.
   - In the [Directory] edit box, enter the directory name in which the workspace will be created. Click the [Browse...] button to select a directory.

After a setting, click the [OK] button.

3.2.1.2 Step2 : Setting for the Toolchain

A wizard for the project creation starts.



Here, the following contents are set.

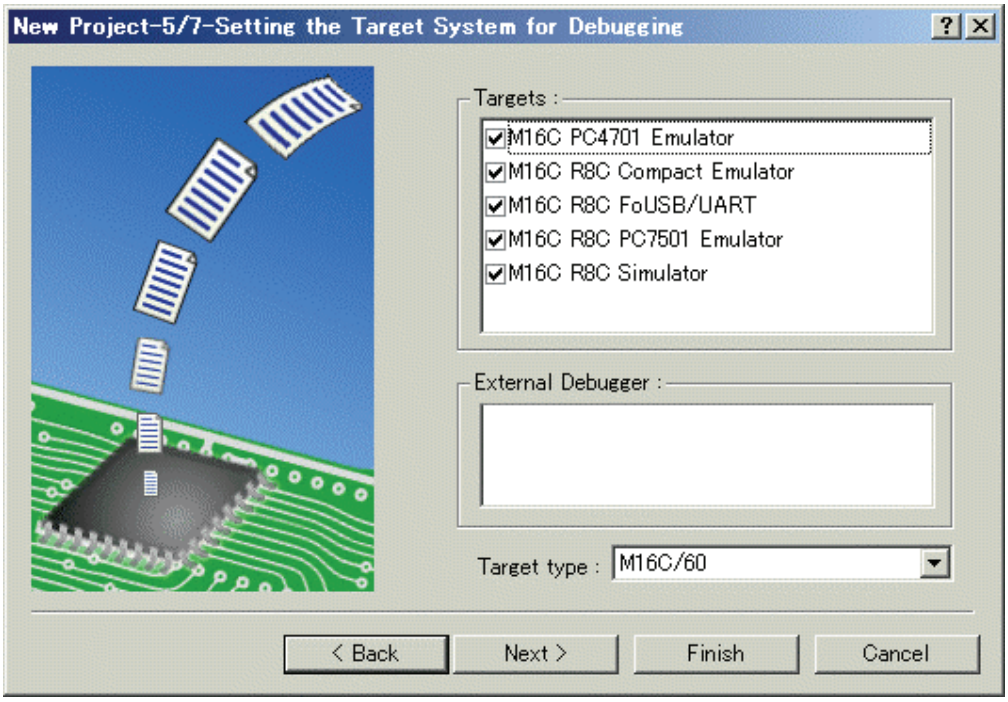
- toolchain
- the setting for the real-time OS (when using)
- the setting for the startup file, heap area, stack area, and so on

Please set required information and click the [Next] button.

The contents of a setting change with C/C++ compiler packages of use. Please refer to the manual attached to your C/C++ compiler package about the details of the contents of a setting.

3.2.1.3 Step 3: Selecting of the Target Platform

Select the target system used for your debugging (emulator, simulator).
When the setting for the toolchain has been completed, the following dialog box is displayed.

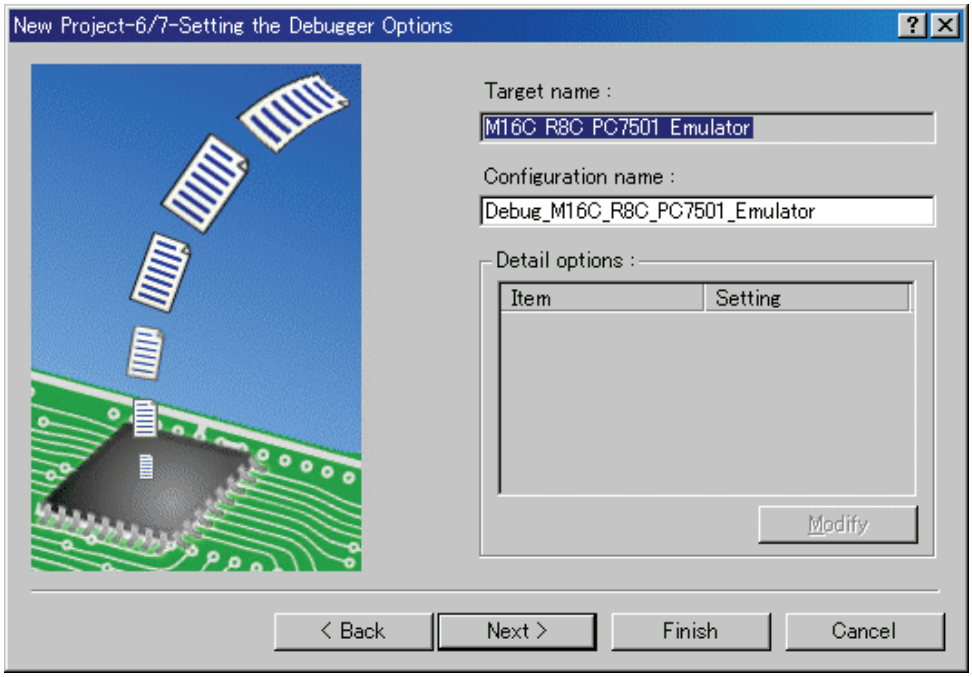


1. Selecting of the Target type
   In the [Target type] list box, select the target CPU type.
2. Selecting of the Target Platform
   In the [Targets] area, the target for the session file used when this debugger is activated must be selected here.
   Check the box against the target platform. (And choose other target as required.)

And click the [Next] button.

3.2.1.4 Step4 : Setting the Configuration File Name

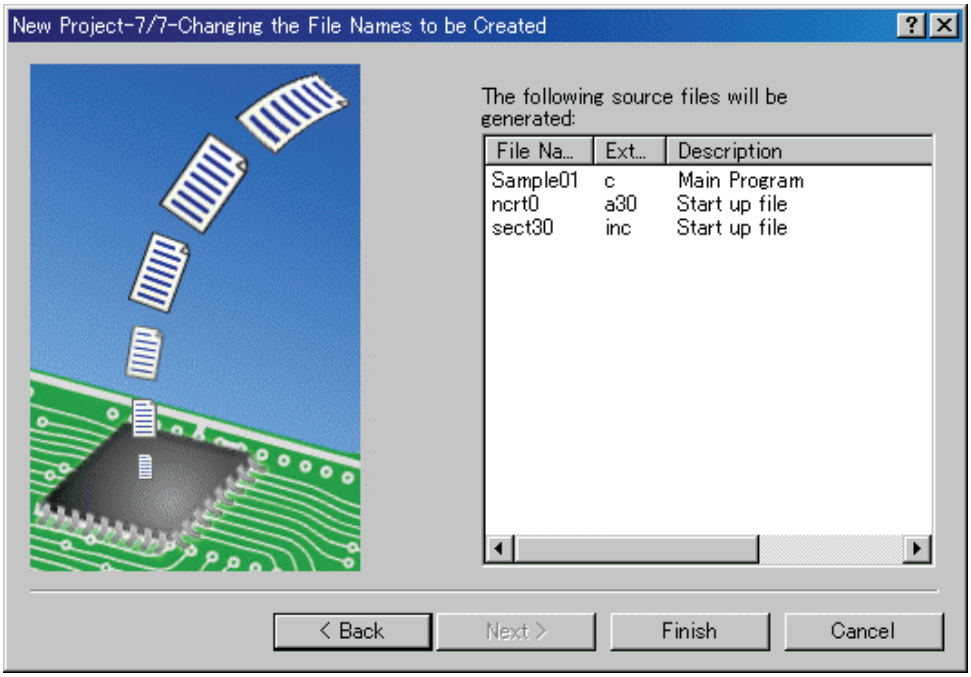
Set the configuration file name for each of the all selected target.
The configuration file saves the state of High-performance Embedded Workshop except for the target (emulator, simulator).



The default name is already set. If it is not necessary to change, please click the [next] button as it is.

3.2.1.5 Step5 : The check of a created file name

Finally, confirm the file name you create. The files which will be generated by the High-performance Embedded Workshop are displayed If you want to change the file name, select and click it then enter the new name.



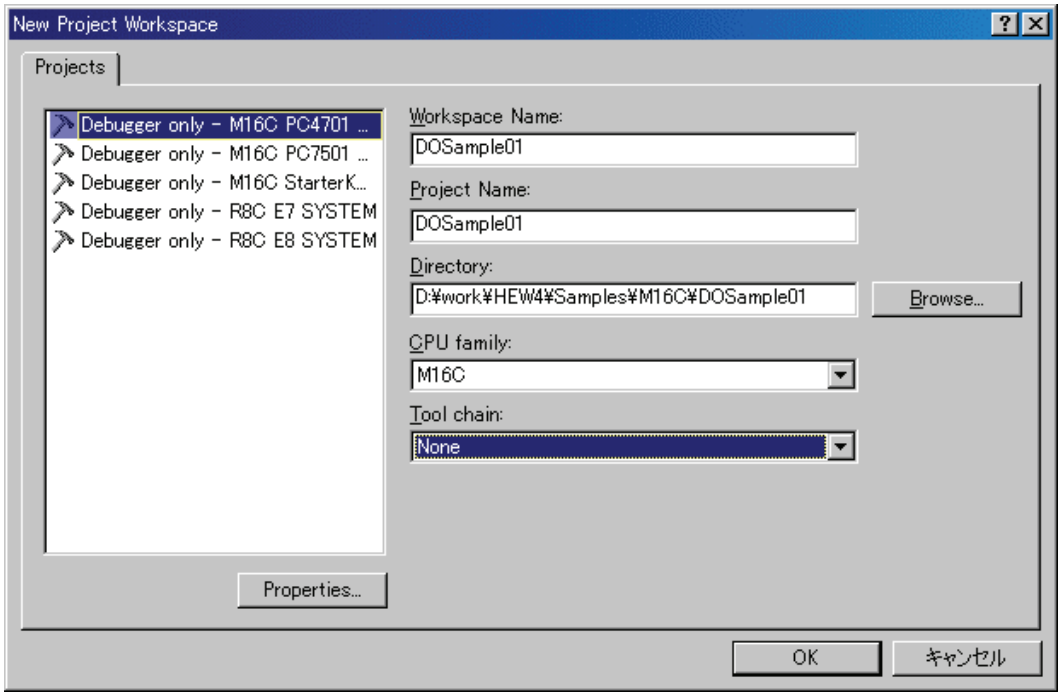
This is the end of the emulator settings.
Exit the Project Generator following the instructions on the screen.

3.2.2 Creating a New Workspace (Toolchain Not Used)

When debugging the existing load module file with this product, a workspace is created by this method.(It can work even if the tool chain is not installed.)

3.2.2.1 Step1 : Creation of a new workspace

In the [Welcome!] dialog box that is displayed when the High-performance Embedded Workshop is activated, select the [Create a new project workspace] radio button and click the [OK] button.
Creation of a new workspace is started. The following dialog box is displayed.

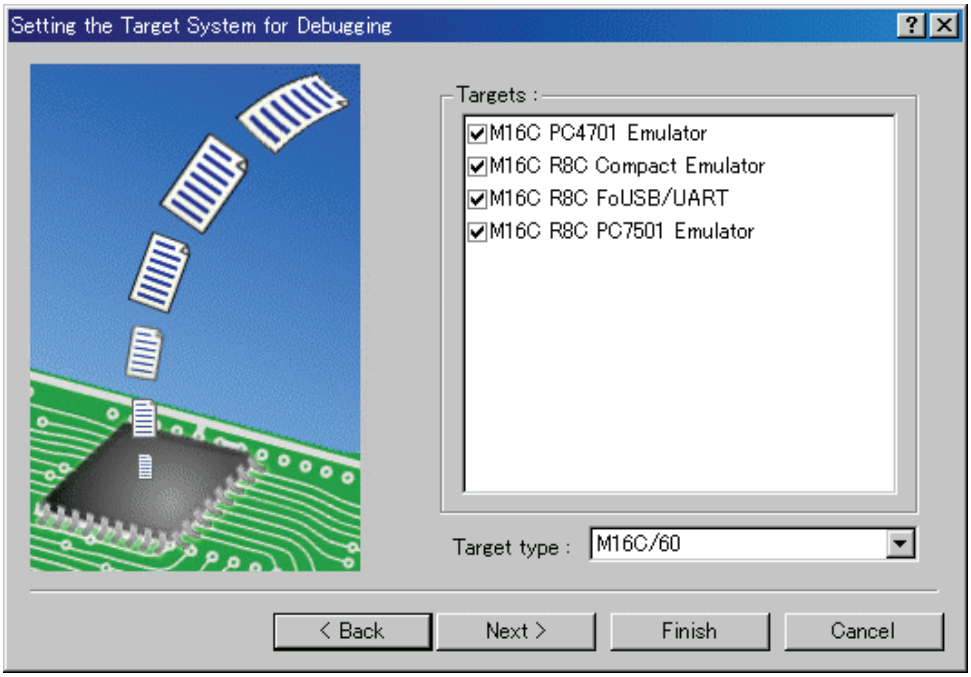


1. Select the target CPU family
   In the [CPU family] combo box, select the target CPU family.
2. Select the target toolchain
   In the [Tool chain] combo box, select "None". In this case, toolchain is not used.
   (When the toolchain has not been installed, the fixed information is displayed in this combo box.)
3. Select the project type
   (When the toolchain is not used, it is displayed on a [Project Type] list box as "Debugger only - Target Name". Select it. (When two or more project types are displayed, please select one of them.)
4. Specify the workspace name and project name
   - In the [Workspace Name] edit box, enter the new workspace name.
   - In the [Project Name] edit box, enter the project name. When the project name is the same as the workspace name, it needs not be entered.
   - In the [Directory] edit box, enter the directory name in which the workspace will be created. Click the [Browse...] button to select a directory.

After a setting, click the [OK] button.

3.2.2.2 Step 2: Selecting of the Target Platform

Select the target system used for your debugging (emulator, simulator).
A wizard starts and the following dialog box is displayed.

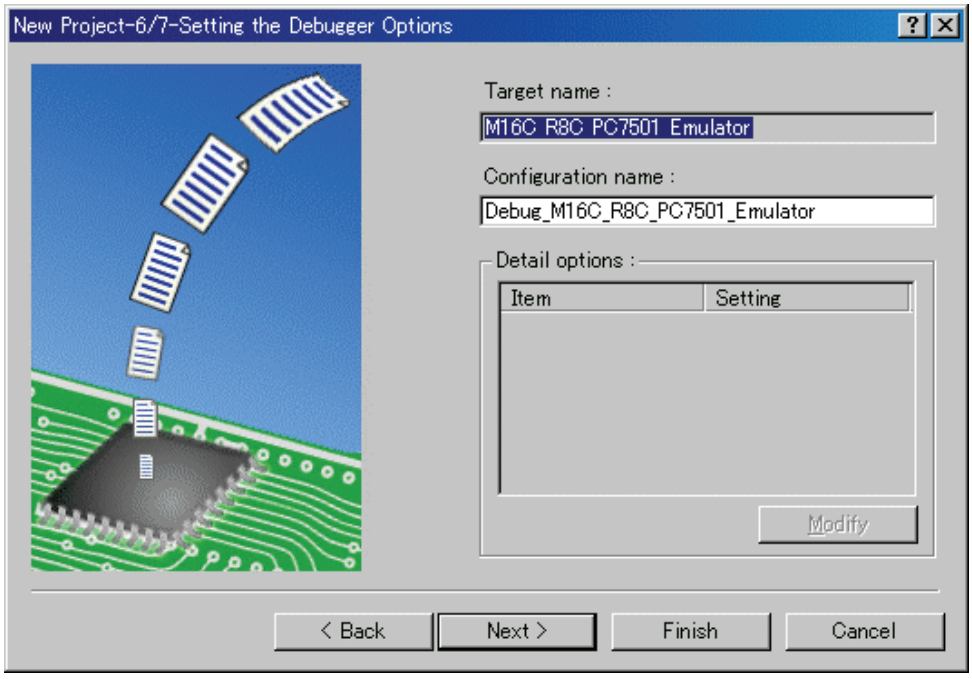


1. Selecting of the Target type
   In the [Target type] list box, select the target CPU type.
2. Selecting of the Target Platform
   In the [Targets] area, the target for the session file used when this debugger is activated must be selected here.
   Check the box against the target platform. (And choose other target as required.)

And click the [Next] button.

3.2.2.3 Step3 : Setting the Configuration File Name

Set the configuration file name for each of the all selected target.
The configuration file saves the state of High-performance Embedded Workshop except for the target (emulator, simulator).

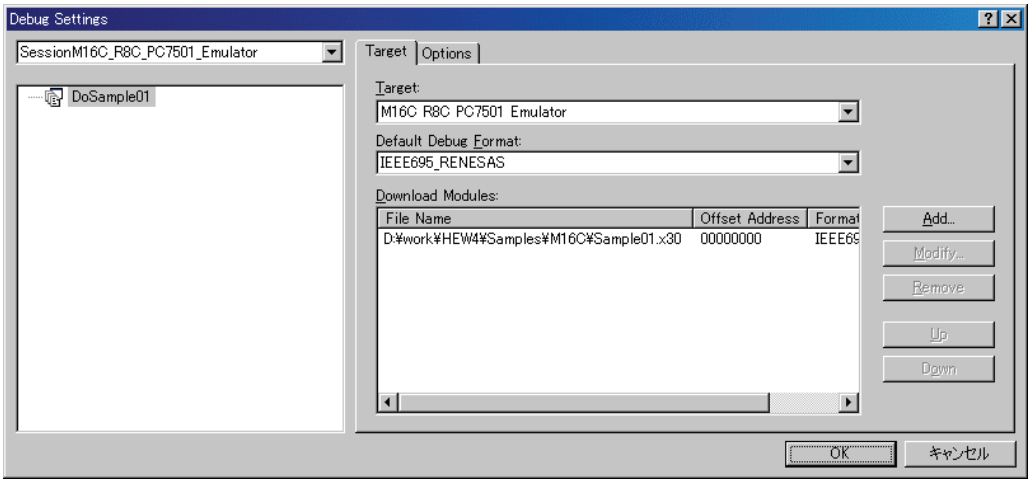


The default name is already set. If it is not necessary to change, please click the [next] button as it is.
This is the end of the emulator settings.
Exit the Project Generator following the instructions on the screen.
And the dialog for the setup of a debugger is also displayed at this time . If preparation of an emulator is completed, set up the debugger in this dialog box and connect with an emulator.

3.2.2.4 Step4 : Registering the Load modules to be downloaded

Finally, register the load module file to be used.

Select [Debug Settings...] from the [Debug] menu to open the [Debug Settings] dialog box.

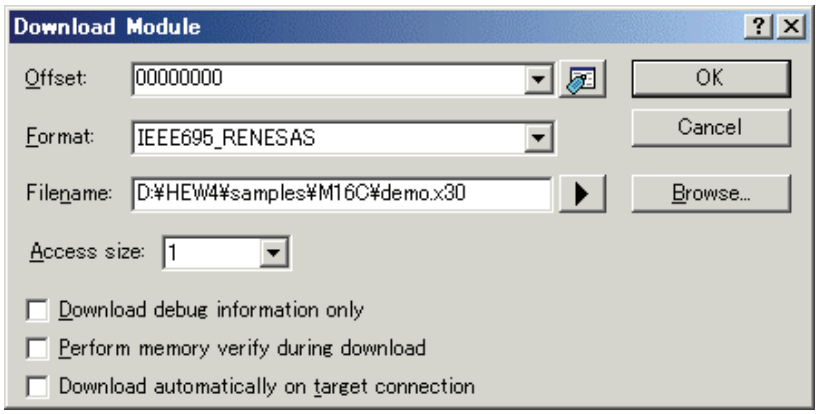


1. Select the product name to be connected in the [Target] drop-down list box.
2. Select the format of the load module to be downloaded in the [Default Debug Format] drop-down list box.

Format Name	Contents
IEEE695_RENESAS	IEEE-695 format file (When Using NCxx)
IEEE695_IAR	IEEE-695 format file (When Using IAR cross tool)
IEEE695_TASKING	IEEE-695 format file (When Using Tasking cross tool)
ELF/DWARF2_IAR	ELF/DWARF2 format file (When Using IAR cross tool)
ELF/DWARF2_TASKING	ELF/DWARF2 format file (When Using Tasking cross tool)
Intel-Hex+Sym	Intel Hex format file with Symbol format file (When Using SRA74)
IEEE695_ICC740	IEEE-695 format file (When Using ICC740)

This debugger does not support the object formats, which are not shown in the drop down list.

3. Then register the corresponding download module in the [Download Modules] list box.
   A download module can be specified in the dialog opened with a [Add...] button.



- Enter the offset at which to load the download module in the [Offset] edit box.
- Select the format of the download module in the [Format] edit box. Please refer to the upper table about the format name of a download module.
- Enter the full path and filename of the download module in the [Filename] edit box.
- Specifies the access size for the current download module in the [Access size] list box.

After that, click the [OK] button.

ATTENTION

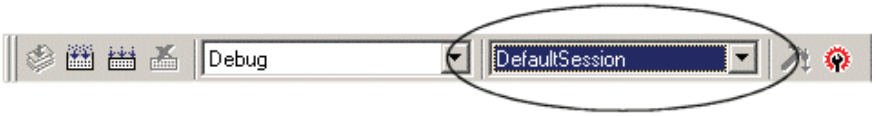
"Access size" and "Perform memory verify during download" is ignored.
The access size is always set to 1 and the verification does not work.

3.3 Starting the Debugger

The debugging can be started by connecting with an emulator.

3.3.1 Connecting the Emulator

Connect the emulator by simply switching the session file to one in which the setting for the emulator use has been registered.
The session file is created by default. The session file has information about the target selected when a project was created.
In the circled list box in the following tool bars, select the session name including the character string of the target to connect.



After the session name is selected, the dialog box for setting the debugger is displayed and the emulator will be connected.

3.3.2 Ending the Emulator

The emulator can be exited by using the following methods:

1. Selecting the "DefaultSession"
   Select the "DefaultSession" in the list box that was used at the time of emulator connection.
2. Exiting the High-performance Embedded Workshop
   Select [Exit] from the [File] menu. High-performance Embedded Workshop will be ended.

The message box, that asks whether to save a session, will be displayed when an emulator is exited. If necessary to save it, click the [Yes] button. If not necessary, click the [No] button.

4. Setup the Debugger

4.1 Init Dialog

The Init dialog box is provided for setting the items that need to be set when the debugger starts up. The contents set from this dialog box are also effective the next time the debugger starts. The data set in this dialog remains effective for the next start.

The tabs available on this dialog box vary with each product used. For details, click the desired tab name shown in the table below.

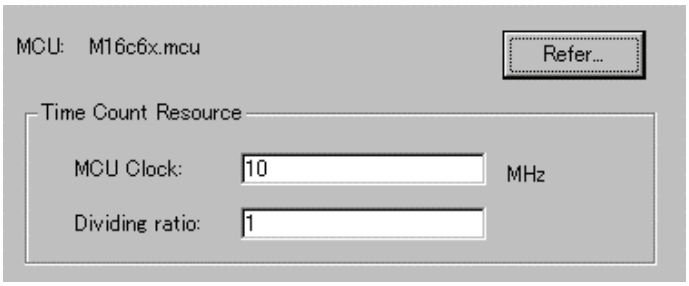
Tab Name	Product Name			
	The debugger for R32C	The debugger for M32C	The debugger for M16C/R8C	The debugger for 740
MCU	exist	exist	exist	exist
Debugging Information	exist	exist	exist	exist
Script	exist	exist	exist	exist
Trace	exist	exist	exist	---
I/O Script	exist	exist	exist	---

You can open the Init dialog using either one of the following methods:

- After the debugger gets started, select Menu - [Setup] -> [Simulator] -> [System...].
- Start Debugger while holding down the Ctrl key.

4.1.1 MCU Tab

The specified content becomes effective when the next being start.



4.1.1.1 Specifying the MCU file

Click the "Refer" button.
The File Selection dialog is opened. Specify the corresponding MCU file.

- An MCU file contains the information specific to the target MCU.
- The specified MCU file is displayed in the MCU area of the MCU tab.

If the corresponding MCU file is not contained in the debugger/emulation pod, you must create a new MCU file. To do this, see the following:

Refer to "4.3 Method of making MCU file"

4.1.1.2 Specifying Clock Frequency

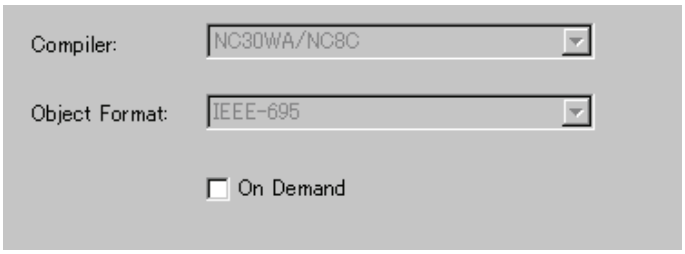
Specify the operation clock of the target MCU within the MCU Clock field in the Time Count Resource group (in units of MHz).

Specify the MCU clock and the clock divide ratio.
If you are using the MCU at 10 MHz divided by 4, for example, enter "10" on the left side and "4" on the right side of the text box.

If no values are set in the clock divide ratio specifying area, it is assumed that the clock is not divided (i.e., the same as you would specify the value 1).

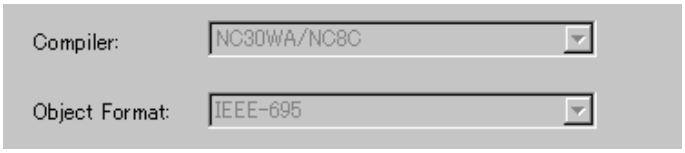
4.1.2 Debugging Information Tab

The specified content becomes effective when the next being download.



4.1.2.1 display the compiler used and its object format

Display the compiler used and its object file format.



Please specify the compiler used and its object file format in the dialog opened by menu [Debug] -> [Debug Settings...].

4.1.2.2 Specify the Storing of Debugging Information

There are two methods for storing debugging information: on-memory and on-demand.
Select one of these two methods. (The on-memory method is selected by default.)
To select the on-demand method, click the On Demand check box.

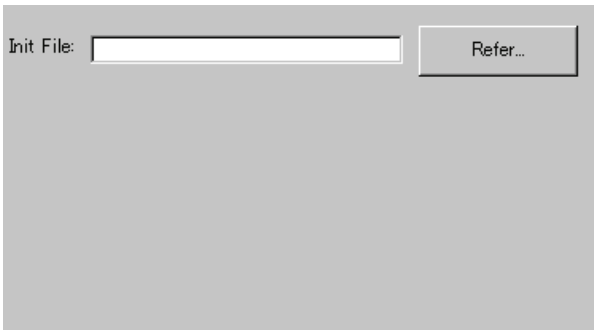
- On-memory method
  Debugging information is stored in the internal memory of your computer.
  This method is suitable when the load module (target program) size is small.
- On-demand method
  Debugging information is stored in a reusable temporary file on the hard disk of your computer.
  Because the stored debugging information is reused, the next time you download the same load module it can be downloaded at high speed.
  This method is suitable when the load module (target program) size is large.

Notes

- If the load module size is large, the on-memory method may be inefficient because it requires a very large amount of time for downloading. In such a case, select the on-demand method.
- In the on-demand method, a folder in which to store a reusable temporary file is created in the folder that contains the downloaded load module. This folder is named after the load module name by the word "~INDEX_" to it. If the load module name is "sample.abs", for example, the folder name is "~INDEX_sample". This folder is not deleted even after quitting the debugger.

4.1.3 Script Tab

The specified content becomes effective when the next being start.



4.1.3.1 Automatically Execute the Script Commands

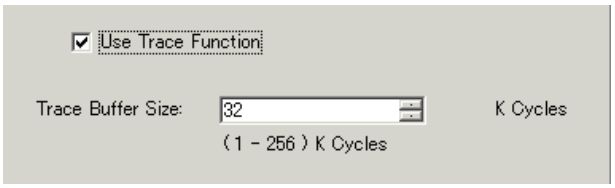
To automatically execute the script command at start of Debugger, click the "Refer" button to specify the script file to be executed.

By clicking the "Refer" button, the File Selection dialog is opened.
The specified script file is displayed in the "Init File:" field.
To disable auto-execution of the script command, erase a character string displayed in the "Init File:" field.

4.1.4 Trace Tab

Specify whether or not to enable trace measurement, and when you chose to enable, specify the trace buffer size.
In the 740 debugger, this tab is not displayed.

The contents you specified here are also effective the next time you start the debugger.

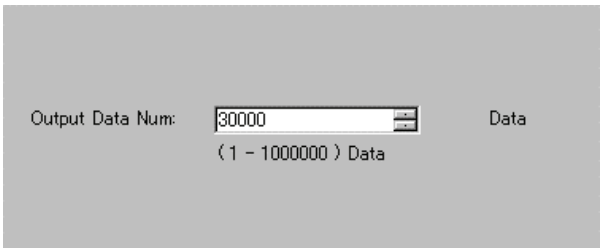


To perform trace measurement, check Use Trace Function.
If trace measurement is disabled, you cannot open the Trace Window and Trace Point Setting Window.
In the Trace Buffer Size area, specify the size of the buffer in which to store the traced data (in K Cycles).

4.1.5 I/O Script Tab

Specify the number of data to be recorded by the I/O Window or Output Port Window's port output function.
In the 740 debugger, this tab is not displayed and the maximum number of data is 30000.

The contents you specified here are also effective the next time you start the debugger.

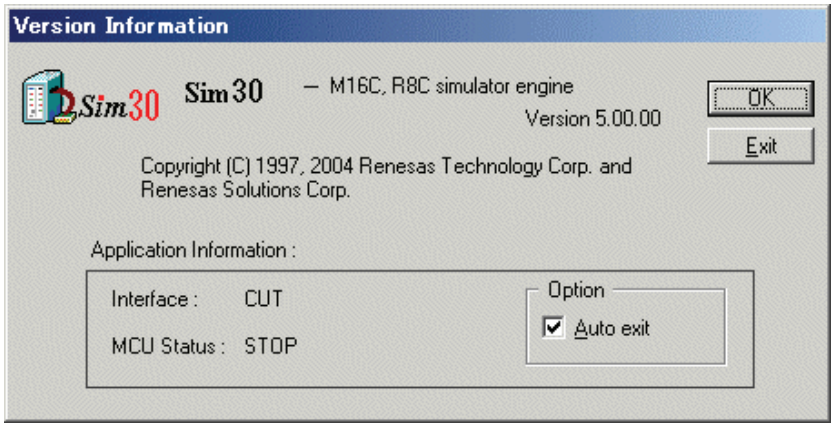


In the Output Data Num area, specify the number of output data to be recorded.

4.2 Simulator engine setup

When Simulator engine simxx starts up, it is registered in the system tray.

Right-clicking on the running simxx and selecting [Version...] from the menu bar will open up the Version Information dialog box. Use this Version Information dialog box to set up simxx.
(Shown below is the Version Information dialog box of the simulator engine for the M16C and M32C series debugger.)



- Auto Exit Switch Setting
  By checking the Auto exit check box, simxx can be terminated at the same time the simulator debugger front-end finishes.
- Communications Connection Status
  CONNECT is displayed when connected to the simulator debugger. CUT is displayed when there is no connection.
- Simulator MCU Status (RUN/STOP)
  RUN is displayed when the simulator MCU is running, STOP when stopped.
- OK button
  Closes the Version Information dialog box
- Exit button
  Exits simxx.

4.3 Method of making MCU file

4.3.1 Method of making MCU file (the R32C Debugger)

In the MCU file, write the following contents in the order listed below.
For the file name, specify the MCU name. For the extension, specify ".mcu".

1. UART0 Transmit/Receive Control Register 1 address
2. UART1 Transmit/Receive Control Register 1 address
3. UART0 Transmit Buffer Register address
4. UART1 Transmit Buffer Register address

Write each address in hexadecimal. Do not add the prefix that represents the radix.

4.3.1.1 Example

```
365
36D
362
36A
```

4.3.2 Method of making MCU file (the M32C Debugger)

In the MCU file, write the following contents in the order listed below.
For the file name, specify the MCU name. For the extension, specify ".mcu".

1. MCU type ("0" or "1")
2. UART0 Transmit/Receive Control Register 1 address
3. UART1 Transmit/Receive Control Register 1 address
4. UART0 Transmit Buffer Register address
5. UART1 Transmit Buffer Register address

The MCU type only needs to be specified for the M32C Debugger.

"0" ... Selects the M16C/8x
"1" ... Selects the M32C/8x

Write each address in hexadecimal. Do not add the prefix that represents the radix.

4.3.2.1 Example

```
0
3A5
3AD
3A2
3AA
```

4.3.3 Method of making MCU file (the M16C/R8C Debugger)

In the MCU file, write the following contents in the order listed below.
For the file name, specify the MCU name. For the extension, specify ".mcu".

1. UART0 Transmit/Receive Control Register 1 address
2. UART1 Transmit/Receive Control Register 1 address
3. UART0 Transmit Buffer Register address
4. UART1 Transmit Buffer Register address
5. Reset Vector address
6. Undefined Instruction Interrupt Vector address
7. Overflow Interruput Vector address
8. BRK Instruction Interrupt Vector address

Write each address in hexadecimal. Do not add the prefix that represents the radix.

4.3.3.1 Example

```
3A5
3AD
3A2
3AA
FFFFC
FFFDC
FFFE0
FFFE4
```

4.3.4 Method of making MCU file (the 740 Debugger)

In the MCU file, write the following contents in the order listed below. For the file name, specify the MCU name. For the extension, specify ".mcu". Write each address in hexadecimal. Do not add the prefix that represents the radix.
Please describe information on 3-6 referring to the data book on MCU used.

1. MCU name
2. Reserved number
3. CPU mode register address and stack page select bit number
4. Reset vector address information
5. BRK vector address information
6. Interrupt vector address information

- MCU name and Reserved number
  Always be sure to add a semicolon (;) before the CPU name and the reserved number.
- CPU mode register address and stack page select bit number
  Separate the CPU mode register address and the stack page select bit number with a colon (:).
- Reset vector address information
  Add the word ":RST" after the reset vector address.
- BRK vector address information
  Add the word ":BRK" after the BRK vector address.
- Interrupt vector address information
  Separate between the interrupt vector address and interrupt control register address, and between the interrupt control register address and interrupt control bit number with a colon (:). Interrupt vector information can be writennen for up to 32 points.

4.3.4.1 Example

```
;M38000
;1
3B:2
FFFC:RST
FFDC:BRK
FFFA:3E:0
FFF8:3E:1
FFF6:3E:2
FFF4:3E:3
FFF2:3E:4
FFF0:3E:5
FFEE:3E:6
FFEC:3E:7
FFEA:3F:0
FFE8:3F:1
FFE6:3F:2
FFE4:3F:3
FFE2:3F:4
FFE0:3F:5
```

Tutorial

(Blank Page)

5. Tutorial

5.1 Introduction

This section describes the main functions of this debugger by using a tutorial program. The tutorial programs are installed to the directory ¥WorkSpace¥Tutorial of the drive you installed High-performance Embedded Workshop. There are workspaces for each targets and each MCUs. Please select the corresponding one to your system, and open the workspace file (*.hws) from the menu [Open Workspace...].
The tutorial program is based on the C program that sorts ten random data items in ascending or descending order.
The tutorial program performs the following actions:

- The tutorial function generates random data to be sorted.
- The sort function sorts the generated random data in ascending order.
- The change function then sorts the data in descending order.

Note

After recompilation, the addresses may differ from those given in this section.

When using the assembler package for 740 family

The tutorial program for the assembler package for 740 family is prepared. If you use the assembler package for 740 family, please use it.

- Please read this tutorial with replacing function names with subroutine name. (e.g. replace "function sort()" with "subroutine sort")
- About the source file name, also please replace it with the corresponding one.
- The diagrams in this tutorial are for C program. The displayed diagram for the assembler program may different from them.
- Step9 and Step12 are descriptions of C program.

5.2 Usage

Please follow these instructions:

5.2.1 Step1 : Starting the Debugger

5.2.1.1 Preparation before Use

To run the High-performance Embedded Workshop and connect the emulator, refer to "3 Preparation before Use".

5.2.1.2 Setup the Debugger

If it connects with an emulator, the dialog box for setting up a debugger will be displayed. Please set up the debugger in this dialog box.
To setup the debugger in this dialog box, refer to "4 Setup the Debugger".
After the setup of a debugger, it will function as a debugger.

5.2.2 Step2 : Checking the Operation of RAM

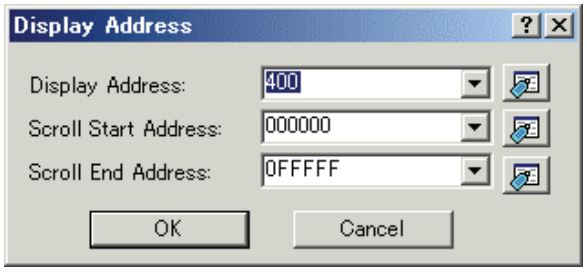
Check that RAM is operating correctly. Display and edit the contents of the memory in the [Memory] window to check that the memory is operating correctly.

Note

The memory can be installed on the board in some microcomputers. In this case, however, the above way of checking the operation of memory may be inadequate. It is recommended that a program for checking the memory be created.

5.2.2.1 Checking the Operation of RAM

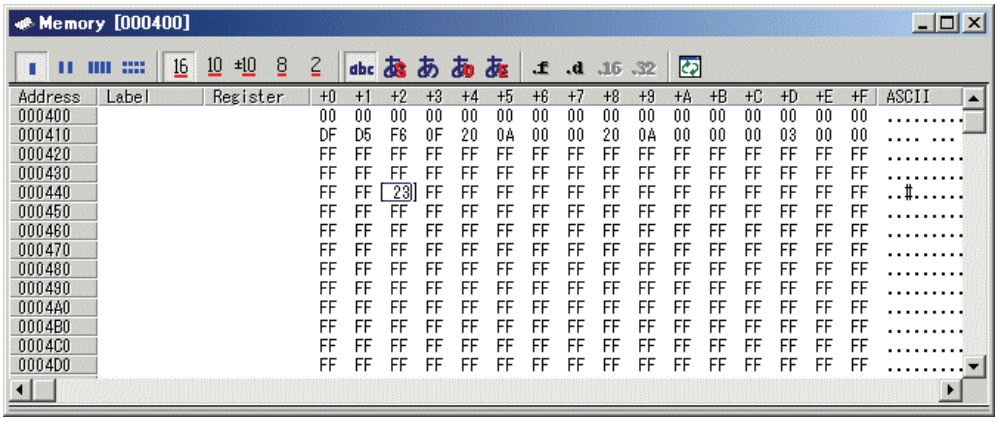
Select [Memory] from the [CPU] submenu of the [View] menu and enter the RAM address (Here, enter H'400) in the [Display Address] edit boxes. The [Scroll Start Address] and [Scroll End Address] editing box is left to a default setting. (By default, the scroll range is set to 0h to the maximum address of MCU.)



Note

The settings of the RAM area differ depending on the product. For details, refer to the hardware manual.

Click the [OK] button. The [Memory] window is displayed and shows the specified memory area.



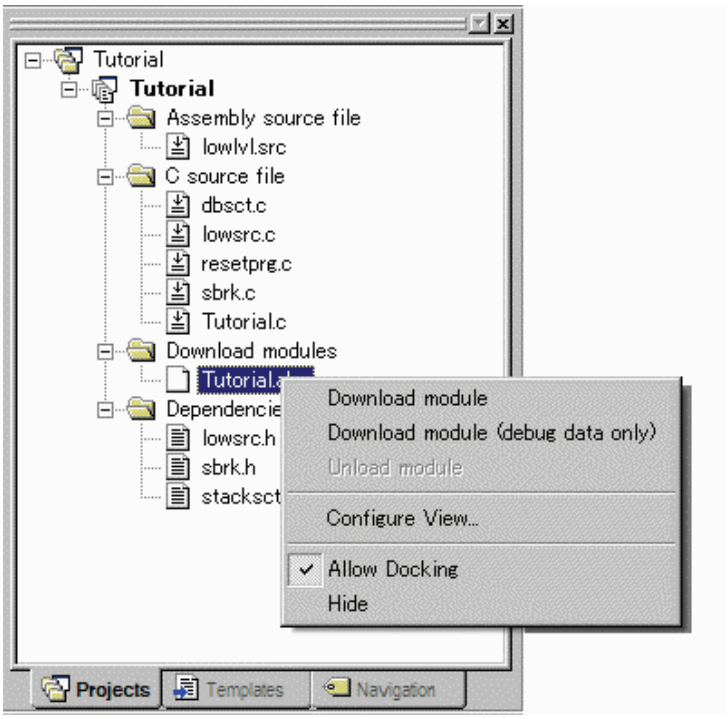
Placing the mouse cursor on a point in the display of data in the [Memory] window and double-clicking allows the values at that point to be changed.

5.2.3 Step3 : Downloading the Tutorial Program

5.2.3.1 Downloading the Tutorial Program

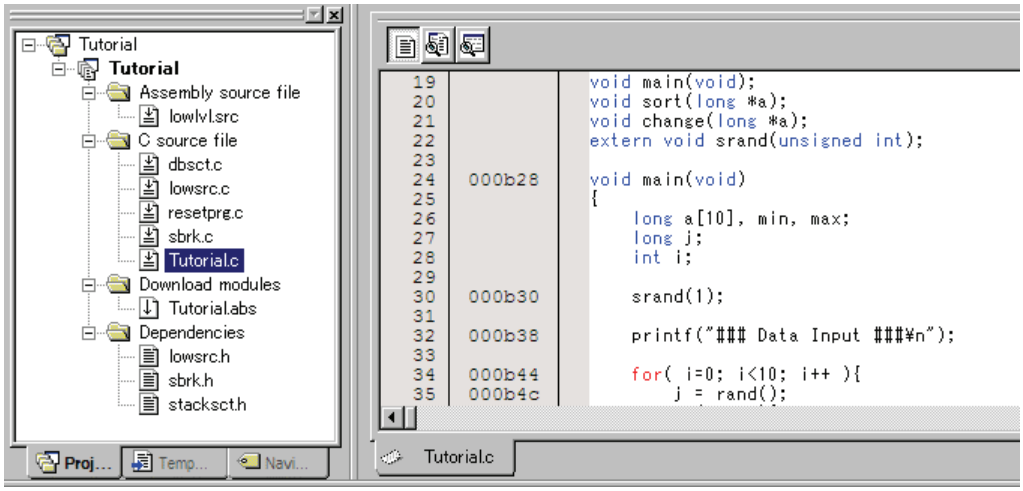
Download the object program to be debugged. The download file and the address to be downloaded will depends on the target mcu you uses. Please replace the screen image and addresses with corresponding one to your target mcu.

- The Debugger for M16C/R8C or M32C
  Select [Download module] from [Tutorial.x30] under [Download modules].
- The Debugger for 740
  If you use the C Compiler Package for 740 Family, select [Download module] from [Tutorial.695] under [Download modules].
  If you use the Assembler Package for 740 Family, select [Download module] from [Tutorial.hex] under [Download modules].



5.2.3.2 Displaying the Source Program

This debugger allows the user to debug a user program at the source level.
Double-click [tutorial.c] under [C source file]. A [Editor(Source)] window opens and the contents of a "Tutorial.c" file are displayed.



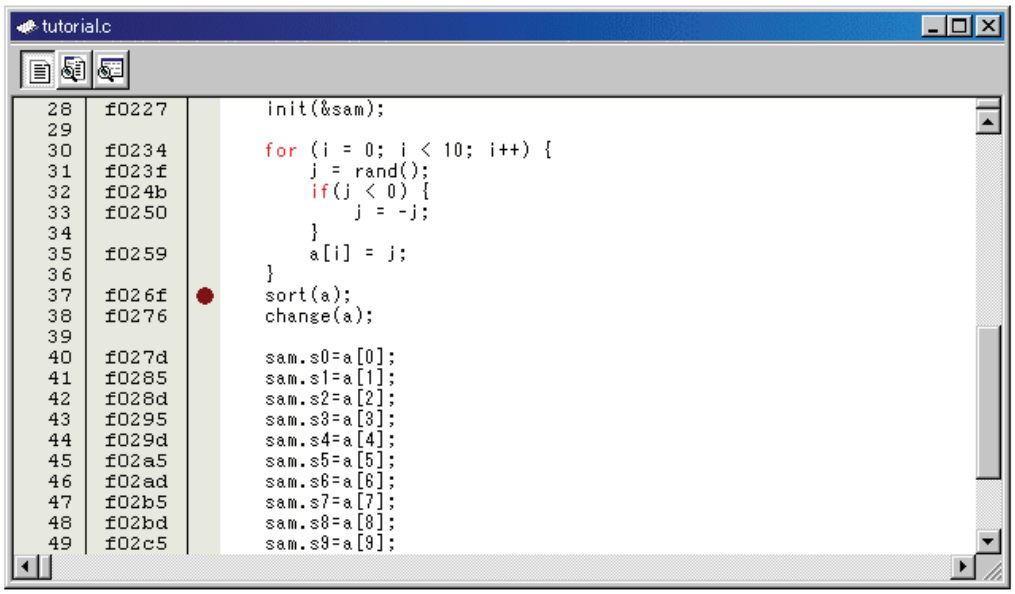
Select the [Format Views...] option from the [Setup] menu to set a font and size that are legible, if necessary.
Initially the [Editor(Source)] window shows the start of the user program, but the user can use the scroll bar to scroll through the user program and look at the other statements.

5.2.4 Step4 : Setting a Breakpoint

A software breakpoint is a basic debugging function.
The [Editor(Source)] window provides a very simple way of setting a software breakpoint at any point in a program.

5.2.4.1 Setting a Software Breakpoint

For example, to set a software breakpoint at the sort function call:
Double-click the [S/W breakpoints] column on the line containing the sort function call.



The red symbol will appear on the line containing the sort function call. This shows that a softwarebreak breakpoint has been set.

5.2.5 Step5 : Executing the Program

Execute the program as described in the following:

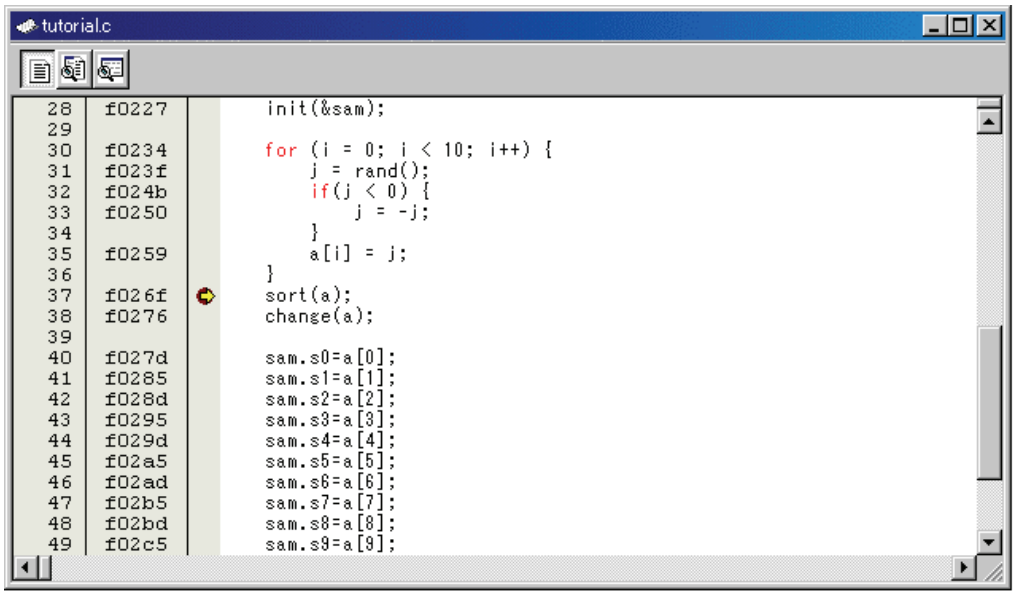
5.2.5.1 Resetting of CPU

By default, CPU is not reset after downloading a program.

To reset the CPU, select [Reset CPU] from the [Debug] menu, or click the [Reset CPU] button  on the toolbar.

5.2.5.2 Executing the Program

To execute the program, select [Go] from the [Debug] menu, or click the [Go] button  on the toolbar.
The program will be executed up to the breakpoint that has been set, and an arrow will be displayed in the [S/W Breakpoints] column to show the position that the program has halted.

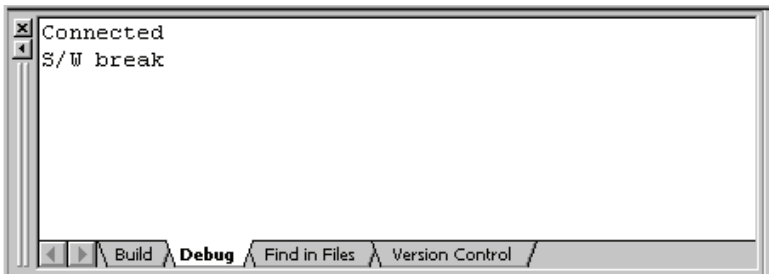


Note

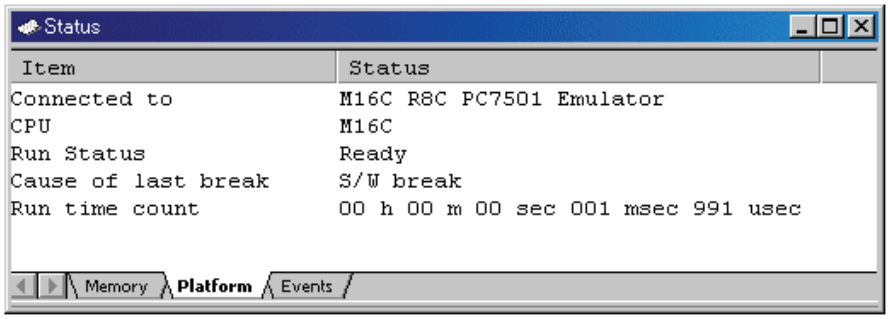
When the source file is displayed after a break, a path of the source file may be inquired. In this case, please specify the location of a source file.

5.2.5.3 Reviewing Cause of the Break

The break factor is displayed in the [Output] window.



The user can also see the cause of the break that occurred last time in the [Status] window.
Select [Status] from the [CPU] submenu of the [View] menu. After the [Status] window is displayed, open the [Platform] sheet, and check the Status of Cause of last break.
The debugger for 740 doesn't support this function.



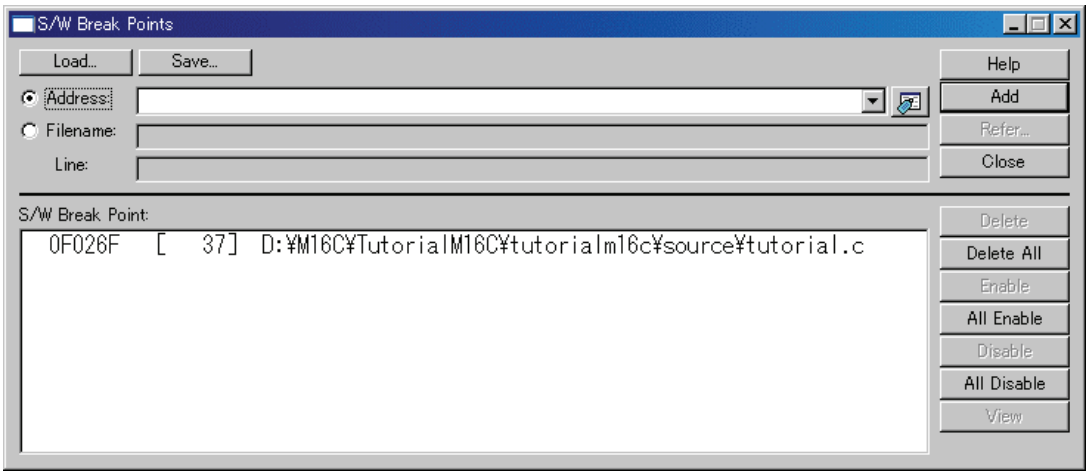
Please refer to "11 Display the Cause of the Program Stoppage" about the notation of a break factor.

5.2.6 Step6 : Reviewing Breakpoints

The user can see all the breakpoints set in the program in the [S/W Break Points] window.

5.2.6.1 Reviewing Breakpoints

Select [S/W Break Points] from the [Break] submenu of the [View] menu. The [S/W Break Points] window is displayed.



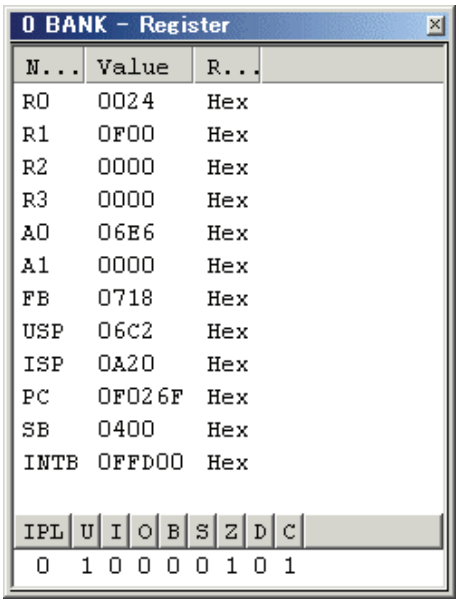
This window allows the user to set or change breakpoints, define new breakpoints, and delete, enable, or disable breakpoints.

5.2.7 Step7 : Viewing Register

The user can see all registers/flags value in the [Register] window.

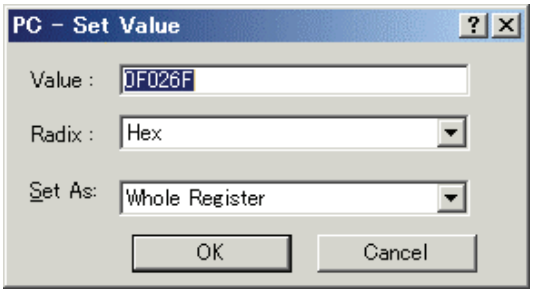
5.2.7.1 Viewing Register

Select [Registers] from the [CPU] submenu of the [View] menu. The [Register] window is displayed. The figure below shows a Register window of the debugger for M16C/R8C.



5.2.7.2 Setting the Register Value

You can change a register/flag value from this window.
Double-click the register line to be changed. The dialog is opened. Enter the value to be changed.

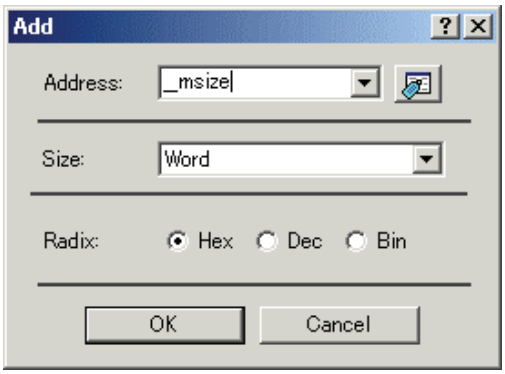


5.2.8 Step8 : Viewing Memory

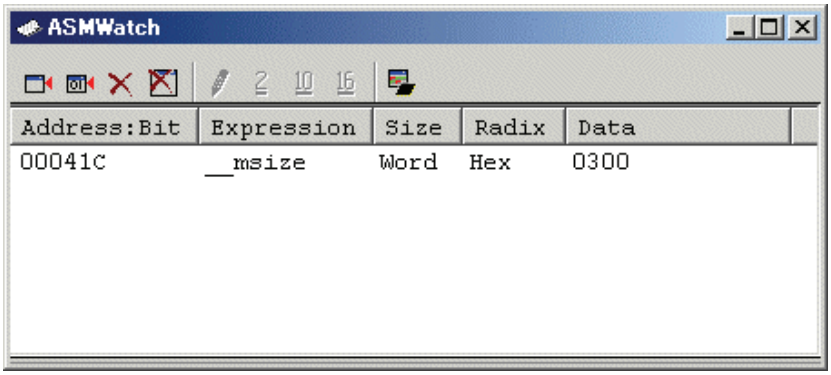
When the label name is specified, the user can view the memory contents that the label has been registered in the [ASM Watch] window.

5.2.8.1 Viewing Memory

For example, to view the memory contents corresponding to __msize in word size:
Select [ASM Watch] from the [Symbol] submenu of the [View] menu, open the [ASM Watch] window. And click the [ASM Watch] window with the right-hand mouse button and select [Add...] from the popup menu, enter __msize in the [Address] edit box, and set Word in the [Size] combo box.



Click the [OK] button. The [ASM Watch] window showing the specified area of memory is displayed.

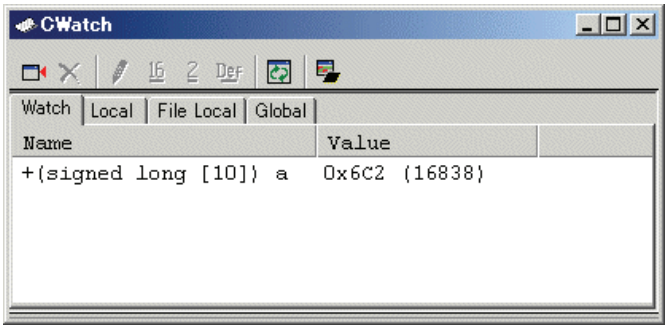


5.2.9 Step9 : Watching Variables

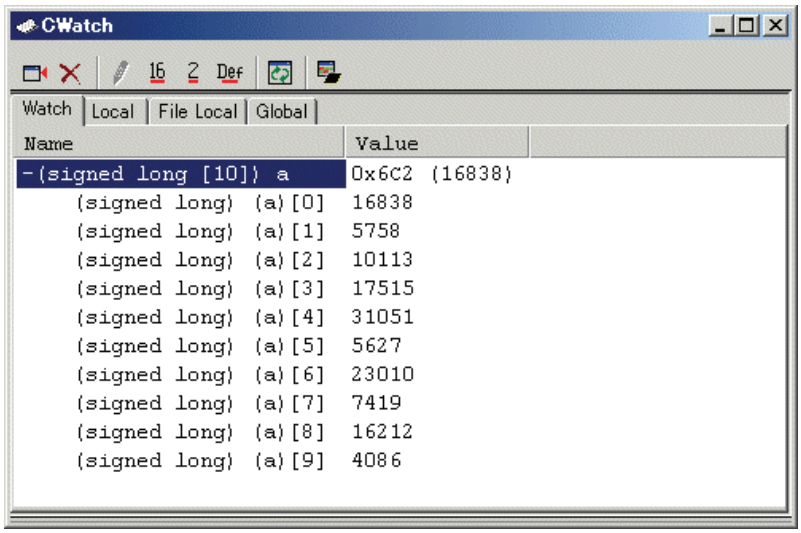
As the user steps through a program, it is possible to watch that the values of variables used in the user program are changed.
If the downloaded program is the program generated by the assembler package for 740 family, you can not watch variables in C watch window.

5.2.9.1 Watching Variables

For example, set a watch on the long-type array a declared at the beginning of the program, by using the following procedure:
Click the left of displayed array a in the [Editor(Source)] window to position the cursor, and select [Add C Watch...] with the right-hand mouse button. The [Watch] tab of [C watch] window in which the variable is displayed opens.



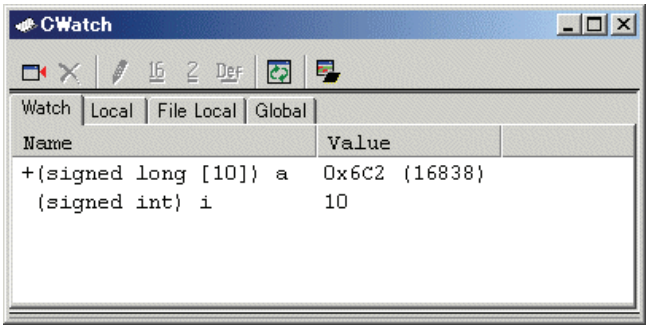
The user can click mark '+' at the left side of array a in the [C Watch] window to watch all the elements.



5.2.9.2 Registering Variable

The user can also add a variable to the [C Watch] window by specifying its name.
Click the [C Watch] window with the right-hand mouse button and select [Add...] from the popup menu.
The following dialog box will be displayed. Enter variable i.

Click the [OK] button. The [C Watch] window will now also show the int-type variable i.



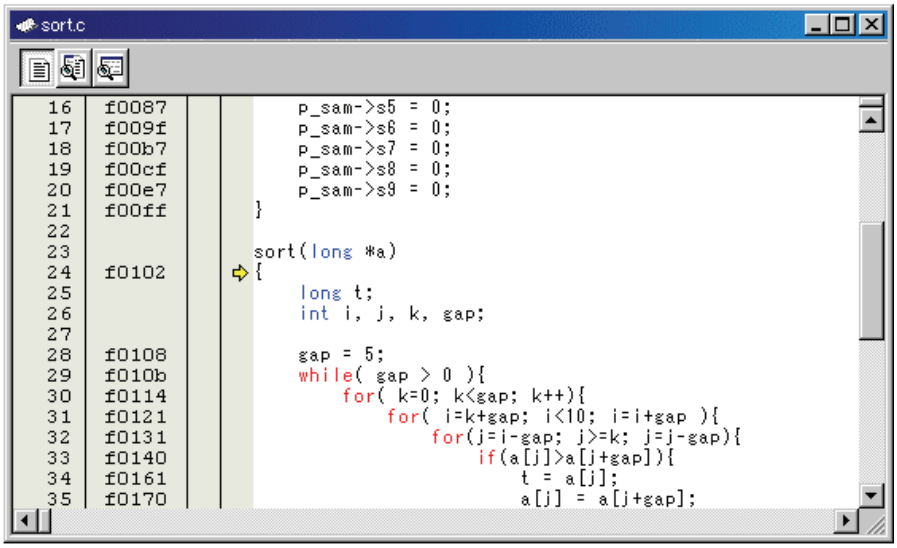
5.2.10 Step10 : Stepping Through a Program

This debugger provides a range of step menu commands that allow efficient program debugging.

1. Step In
   Executes each statement, including statements within functions(subroutines).
2. Step Out
   Steps out of a function(subroutine), and stops at the statement following the statement in the program that called the function(subroutine).
3. Step Over
   Executes a function(subroutine) call in a single step.
4. Step...
   Steps the specified times repeatedly at a specified rate.

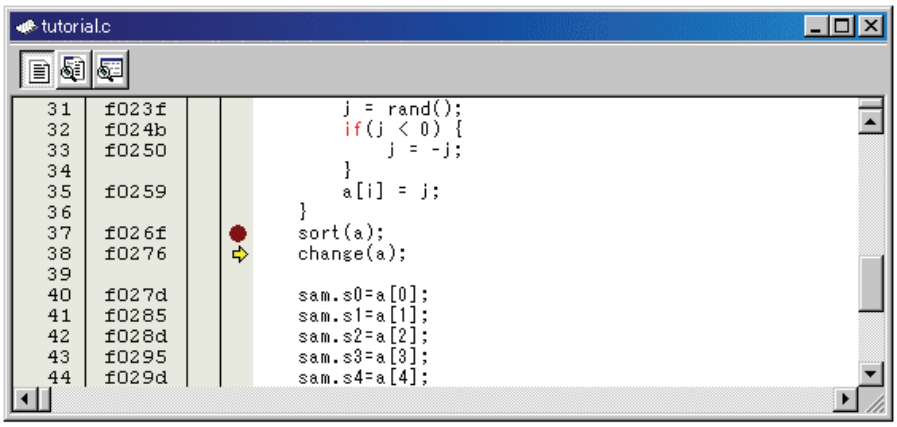
5.2.10.1 Executing [Step In] Command

The [Step In] command steps into the called function(subroutine) and stops at the first statement of the called function(subroutine).
To step through the sort function, select [Step In] from the [Debug] menu, or click the [Step In] button  on the toolbar.
The PC cursor moves to the first statement of the sort function in the [Editor(Source)] window.



5.2.10.2 Executing [Step Out] Command

The [Step Out] command steps out of the called function(subroutine) and stops at the next statement of the calling statement in the main function.
To step out of the sort function, select [Step Out] from the [Debug] menu, or click the [Step Out] button  on the toolbar.
The PC cursor slips out of a sort function, and moves to the position before a change function.

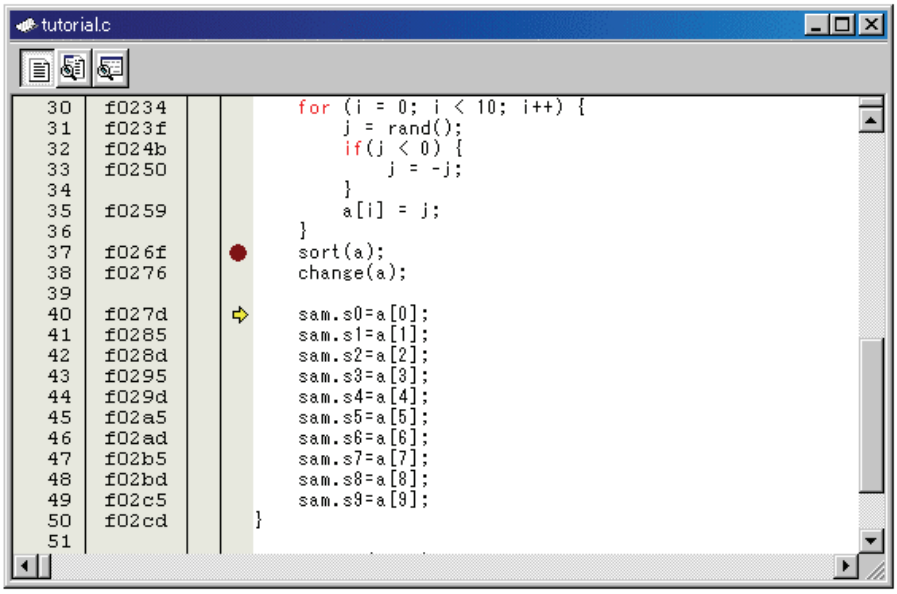


Note

It takes time to execute this function. When the calling source is clarified, use [Go To Cursor].

5.2.10.3 Executing [Step Over] Command

The [Step Over] command executes a function(subroutine) call as a single step and stops at the next statement of the main program.
To step through all statements in the change function at a single step, select [Step Over] from the [Debug] menu, or click the [Step Over] button  on the toolbar.
The PC cursor moves to the next position of a change function.



5.2.11 Step11 : Forced Breaking of Program Executions

This debugger can force a break in the execution of a program.

5.2.11.1 Forced Breaking of Program Executions

Cancel all breaks.
To execute the remaining sections of the main function, select [Go] from the [Debug] menu or the [Go] button  on the toolbar.
The program goes into an endless loop. To force a break in execution, select [Halt Program] from the [Debug] menu or the [Halt] button  on the toolbar.

5.2.12 Step12 : Displaying Local Variables

The user can display local variables in a function using the [C Watch] window.
If the downloaded program is the program generated by the assembler package for 740 family, you can not watch variables in C watch window.

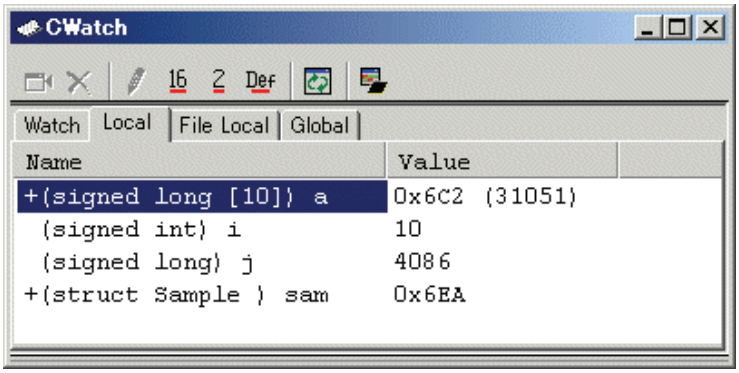
5.2.12.1 Displaying Local Variables

For example, we will examine the local variables in the tutorial function, which declares four local variables: a, j, i, and sam.

Select [C Watch] from the [Symbol] submenu of the [View] menu. The [C Watch] window is displayed. By default, [C watch] window has four tabs as following:

- [Watch] tab
  Only the variable which the user registered is displayed.
- [Local] tab
  All the local variables that can be referred to by the scope in which the the PC exists are displayed. If a scope is changed by program execution, the contents of the [Local] tab will also change.
- [File Local] tab
  All the file local variables of the file scope in which the PC exists are displayed. If a file scope is changed by program execution, the contents of the [File Local] tab will also change.
- [Global] tab
  All the global variables currently used by the downloaded program are displayed.

Please choose the [Local] tab, when you display a local variable.



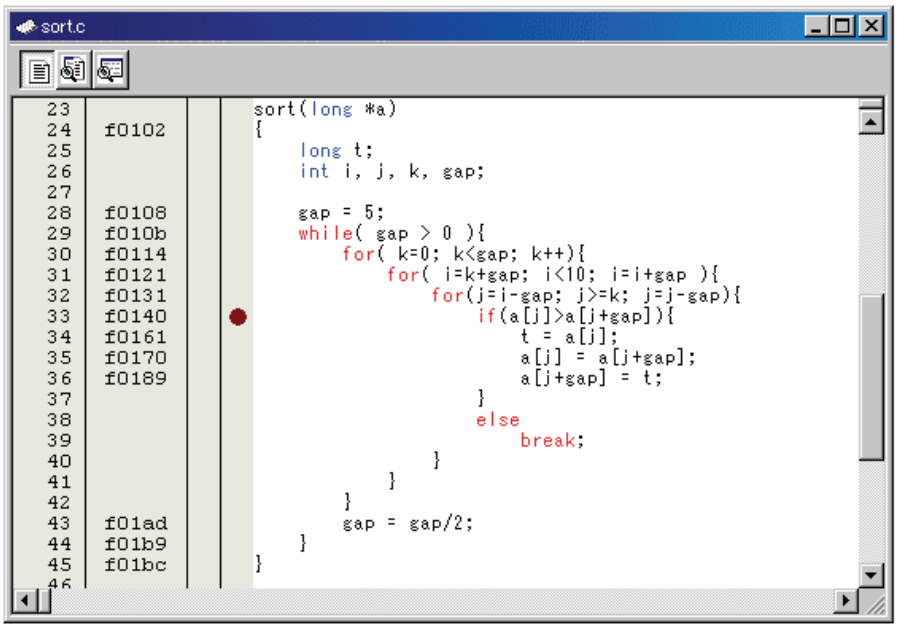
Click mark '+' at the left side of array a in the [Locals] window to display the elements.
When the user refers to the elements of array a before and after the execution of the sort function, it is clarified that random data is sorted in descending order.

5.2.13 Step13 : Stack Trace Function

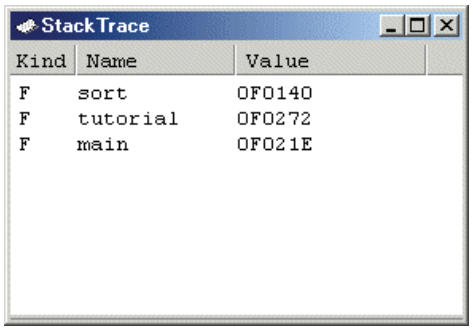
The debugger uses the information on the stack to display the names of functions in the sequence of calls that led to the function to which the program counter is currently pointing.
The debugger for 740 doesn't support the stack trace function.

5.2.13.1 Reference the function call status

Double-click the [S/W Breakpoints] column in the sort function and set a software breakpoint.



To executes the user program from the reset vector address, select [Reset Go] from the [Debug] menu, or click the [Reset Go] button  on the toolbar.
After the break in program execution, select [Stack Trace] from the [Code] submenu of the [View] menu to open the [Stack Trace] window.



The upper figure shows that the position of the program counter is currently at the selected line of the sort() function, and that the sort() function is called from the tutorial() function.

5.2.14 What Next?

This tutorial has described the usage of this debugger.
Sophisticated debugging can be carried out by using the emulation functions that the emulator offers. This provides for effective investigation of hardware and software problems by accurately isolating and identifying the conditions under which such problems arise.

Reference

(Blank Page)

6. Windows/Dialogs

The window of this debugger is shown below.
When the window name is clicked, the reference is displayed.

Window Name	View Menu
RAM Monitor Window	[View]->[CPU]->[RamMonitor]
I/O Timing Setting Window	[View]->[CPU]->[I/O Timing Setting]
Output Port Window *	[View]->[CPU]->[OutputPort]
ASM Watch Window	[View]->[Symbol]->[ASMWatch]
C Watch Window	[View]->[Symbol]->[CWatch]
Coverage Window	[View]->[Code]->[Coverage]
Script Window	[View]->[Script]
S/W Break Point Setting Window	[View]->[Break]->[S/W Break Points]
H/W Break Point Setting Dialog Box	[View]->[Break]->[H/W Break Points]
Trace Point Setting Window *	[View]->[Trace]->[Trace Points]
Trace Window *	[View]->[Trace]->[Trace]
Data Trace Window *	[View]->[Trace]->[Data Trace]
GUI I/O Window	[View]->[Graphic]->[GUI I/O]
MR Window # *	[View]->[RTOS]->[MR]
MR Trace Window # *	[View]->[RTOS]->[MR Trace]
MR Analyze Window # *	[View]->[RTOS]->[MR Analyze]
Task Trace Window # *	[View]->[RTOS]->[Task Trace]
Task Analyze Window # *	[View]->[RTOS]->[Task Analyze]

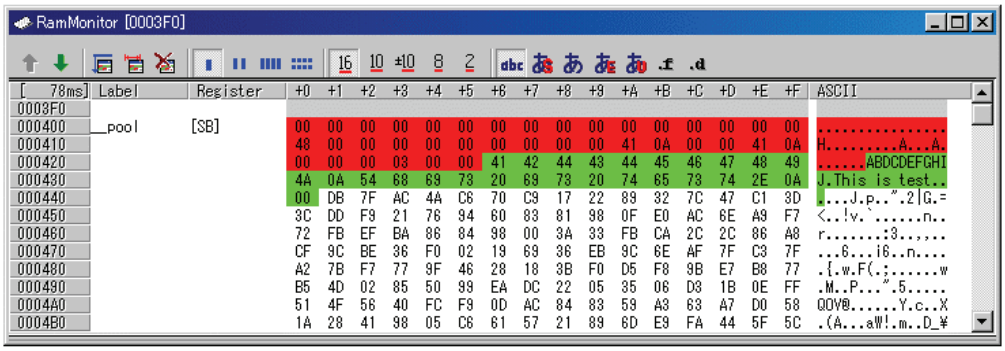
#: The R32C debuggers are not supported
*: The 740 debuggers are not supported.

For the reference of the following windows, refer to the help attached to a High-performance Embedded Workshop main part.

- Differences Window
- Map Window
- Command Line Window
- Workspace Window
- Output Window
- Disassembly Window
- Memory Window
- IO Window
- Status Window
- Register Window
- Image Window
- Waveform Window
- Stack Trace Window

6.1 RAM Monitor Window

The RAM monitor window is a window in which changes of memory contents are displayed while running the target program.
The relevant memory contents are displayed in dump form in the RAM monitor area by using the realtime RAM monitor function. The displayed contents are updated at given intervals (by default, every 100 ms) while running the target program.



- This system provides a 1Kbytes of RAM monitor area, which can be placed at any continuous addresses.
- The RAM monitor area can be changed to any desired address range.
  Refer to "6.1.2 Setting the RAM monitor area" for details on how to change the RAM monitor area.
  The default RAM monitor area is mapped into a 1-Kbyte area beginning with the start address of the internal RAM.
- The display content updating interval can be set for each window individually.
  The actual updating interval at which the display contents are actually updated while running the target program is shown in the title field of the Address display area.
- The background colors of the data display and code display areas are predetermined by access attribute, as shown below.

Access attribute	Background color
Read accessed address	Green
Write accessed address	Red
Non-accessed address	White

The background colors can be changed.

ATTENTION

- The RAM monitor window shows the data that have been accessed through the bus. Therefore, changes are not reflected in the displayed data unless they have been accessed via the target program as in the case where memory is rewritten directly from an external I/O.
- If the data in the RAM monitor area are displayed in lengths other than the byte, it is possible that the data will have different memory access attributes in byte units. If bytes in one data have a different access attribute as in this case, those data are enclosed in parentheses when displayed in the window. In that case, the background color shows the access attribute of the first byte of the data.

001B 00C8 00D2 0000 007C
0000 0000 0000 0000 0000
0000 (007C) FF8C 0000 0000
0000 0000 0000 0050 0000

- The displayed access attributes are initialized by downloading the target program.
- The interval time at which intervals the display is updated may be longer than the specified interval depending on the operating condition (shown below).
  - Host machine performance/load condition
  - Communication interface
  - Window size (memory display range) or the number of windows displayed

6.1.1 Extended Menus

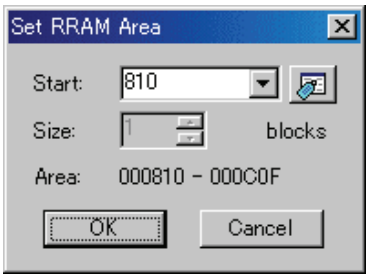
This window has the following popup menus that can be brought up by right-clicking in the window.

Menu		Function
RAM Monitor Area...		Set RAM monitor base address.
Sampling Period...		Set RAM monitor sampling period.
Clear		Clear access attribute.
Up		Moves display position to the immediately preceding RAM monitor area (smaller address)
Down		Moves display position to the immediately following RAM monitor area (larger address)
Address...		Display from specified address.
Scroll Area...		Specify scroll range.
Data Length	1byte	Display in 1Byte unit.
	2bytes	Display in 2Byte unit.
	4bytes	Display in 4Byte unit.
	8bytes	Display in 8Byte unit.
Radix	Hex	Display in Hexadecimal.
	Dec	Display in Decimal.
	Single Dec	Display in Signed Decimal.
	Oct	Display in Octdecimal.
	Bin	Display in Binary.
Code	ASCII	Display as ASCII character.
	SJIS	Display as SJIS character.
	JIS	Display as JIS character.
	UNICODE	Display as UNICODE character.
	EUC	Display as EUC character.
	Float	Display as Floating-point.
	Double	Display as Double Floating-point.
Layout	Label	Switch display or non-display of Label area.
	Register	Switch display or non-display of Register area.
	Code	Switch display or non-display of Code area.
Column...		Set the number of columns displayed on one line.
Split		Split window.
Toolbar display		Display toolbar.
Customize toolbar...		Open toolbar customize dialog box.
Allow Docking		Allow window docking.
Hide		Hide window.

6.1.2 Setting the RAM monitor area

Choose the popup menu [RAM Monitor Area...] in the RAM monitor window.

The Set RRAM Area dialog box shown below will appear.
The start address of the currently set RAM monitor area and the range of the RAM monitor area are displayed in the Start and the Area fields of this dialog box. (No values can be entered in the Size field.)



Use this dialog box to change the position of the RAM monitor area.
Specify the RAM monitor area by its start address. The size cannot be changed (fixed to 1 Kbyte).
The start address can be specified in 0x10 byte units.
If you specify a non-aligned address value, it is rounded off to the nearest address value in 0x10 byte units before being set.

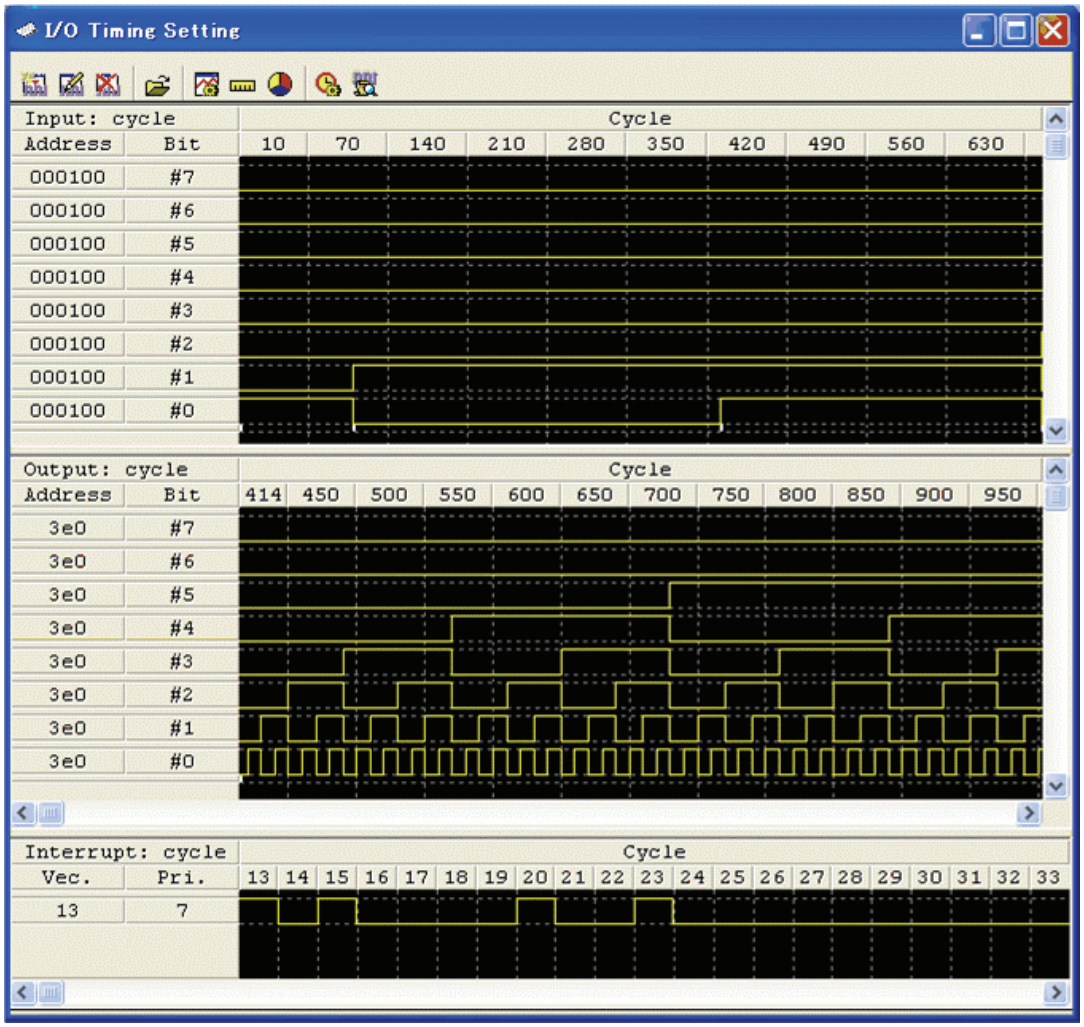
6.1.2.1 Changing the RAM Monitor Area

The start address of the RAM monitor area can be changed.
Specify the start address of the RAM monitor area in the Start field of the Set RRAM Area dialog box. (No values can be entered in the Size field.)

6.2 I/O Timing Setting Window

The I/O Timing Setting Window is used to set and display virtual port input/outputs or virtual interrupts.
Virtual port inputs, virtual interrupt settings, and virtual port output results can be displayed for your reference in numeric or graphic mode.

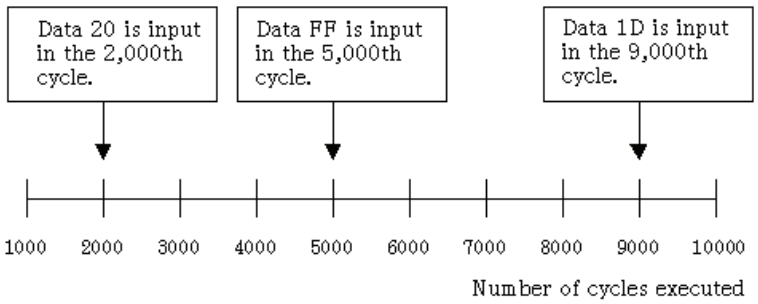
This window is split into three sections, each displaying the setup contents of virtual port inputs, the output results of virtual port outputs, and the setup contents of virtual interrupts.



6.2.1 Virtual Port Input

Virtual Port Input refers to a function that defines changes in the data that is input from external sources to a specified memory address. Use of this function makes it possible to simulate data inputs to the ports defined in the SFR. The defined input data can be referenced by displaying it in chart, numeric (hexadecimal),or graphic mode. There are following three types of virtual port inputs:

1. Cycle synchronized input
   The input data can be written to memory when program execution has reached a specified number of cycles. The data size that can be input is one byte. The diagram below shows an example of a virtual port input that is synchronized to machine cycles.



As shown above, data can be input to memory address 3E0 in any desired cycle as specified by the user.

2. Read access synchronized input
   Data can be input when the program accesses a specified memory location for read.
   The data size that can be input is one byte.
   The diagram below shows an example of a virtual port input that is synchronized to memory accesses for read.

```
#pragma  ADDRESS  port0 = 3e0H
char port0;

read_port()
{
          char key;

          key = port0;   /* Input from port 0 */

             :
             :
}
```

This function aims to assign the value of port 0 to variable key. In such a case, a value can be assigned to variable key by entering it to port 0 when the program accesses port 0 (address 3E0) for read.

To support processing of functions like this, this debugger provides a function that allows you to define the data to be input according to a number of times the specified memory address is read (a virtual input port synchronized to memory accesses for read). By using this function, you can perform an operation where data 0x10 is input to memory address 3E0 when address 3E0 is read first and data 0x20 is input to said memory address when the address is read next.

Number of times the address 3E0 is read	Data input to address 3E0
First	0x10
Second	0x20
Third : :	0x30 : :

3. Interrupt synchronized input
   Data can be input to a specified memory location when a virtual interrupt occurs. The data size that can be input is one byte. The diagram below shows an example of a virtual port input that is synchronized to interrupts.
   Shown in the sample program below is the case where data is read from port 1 (address 3E1) using an interrupt handler routine (in this case, a timer interrupt handler routine).

```
#pragma  ADDRESS  port1 = 3e1H
char port1;

#pragma INTERRUPT        read_port

/* Interrupt handler for polling port 1 */
read_port()
{
          char key;

          key = port1;  /* Input from port 1 */

             :
             :
}
```

This interrupt handler routine aims to assign the value of port 1 to variable key when a virtual interrupt is generated. In such a case, a value can be assigned to variable key by entering it to port 1 when a virtual interrupt (in this case, a timer interrupt) is generated.
It is assumed that timer interrupts are generated using a separately available virtual interrupt function. (For details, refer to the virtual interrupt function described later in this manual.)
To support processing of interrupt handlers like this, this debugger provides a function that allows you to define the data to be input according to a number of times a virtual interrupt is generated (a virtual input port synchronized to virtual interrupts). By using this function, you can perform an operation where data 0xFF is input to memory address 3E1 when the virtual interrupt occurs first and data 0xFE is input to said memory address when the virtual interrupt occurs next time.

Number of times a virtual interrupt is generated	Data input to address 3E1
First	0xFF
Second	0xFE
Third : :	0xFD : :

6.2.2 Virtual Port Output

Virtual Port Output is a function that when data is written to some memory address by the program, allows the written data value to be recorded along with the cycle in which the data was written. The recorded data can be displayed for your reference in chart, numeric, or graphic mode. The number of data entries recorded is the number of entries specified on the Init dialog box's I/O Scrip tab reckoning from when the program started to run.
For example, if data is written to port 0 (address 3E0) by executing a program like the one shown below,

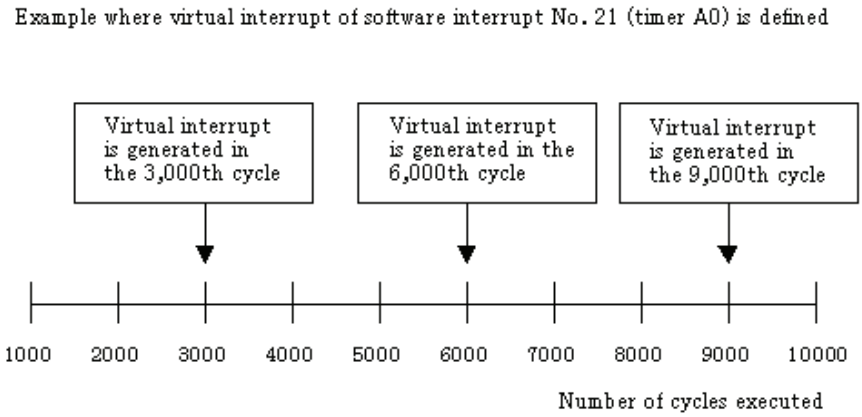
```
#pragma  ADDRESS  port0  = 3e0H
char port0;

out_port(char  data)
{
          port0 = data;   /* Data is output to port 0 */
            :
            :
}
```

the data written to address 3E0 is recorded along with the cycle count in which the data was written.

6.2.3 Virtual Interrupt

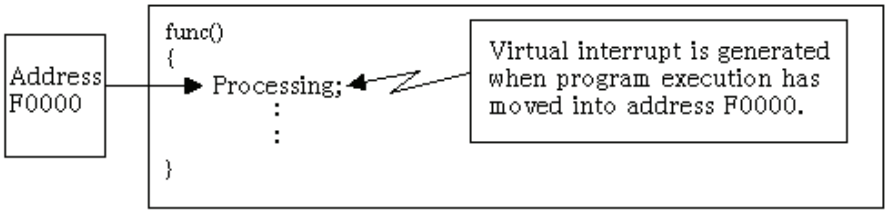
This function defines interrupt generation. Using this function, you can generate timer interrupts or key input interrupts in a simulated manner without having to actually generate them.
There are following three types of virtual interrupts:

1. Cycle synchronized interrupt
   A specified virtual interrupt can be generated when program execution has reached a specified number of cycles. The diagram below shows an example of a virtual interrupt that is synchronized to machine cycles.



As shown above, virtual interrupts (in this case, timer A0 interrupt) can be generated in any desired cycle.

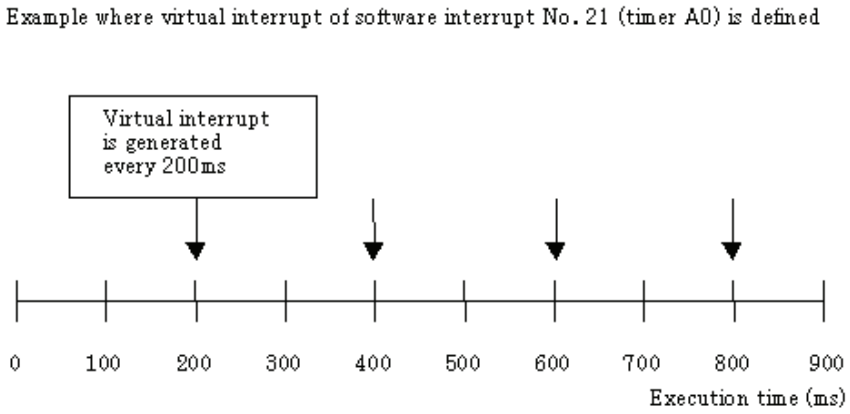
2. Executed address synchronized interrupt
   Virtual interrupts can be generated when the program has executed a specified address. The diagram below shows an example of a virtual interrupt that is synchronized to executed addresses.



As shown above, a specified virtual interrupt can be generated when program execution has moved into address F0000.
By using this function, you can specify that a virtual interrupt be generated when address F0000 is executed first by the program, and that no virtual interrupt be generated when the address is executed next, as shown below.

Number of times the address F0000 is executed	Whether virtual interrupt is generated
First	Virtual interrupt is generated
Second	Virtual interrupt is not generated
Third : :	Virtual interrupt is generated : :

3. Interval-synchronized interrupts
   A virtual interrupt can be generated at specified intervals.
   The following shows an example of a virtual interrupt which is synchronized to a specified interval time.



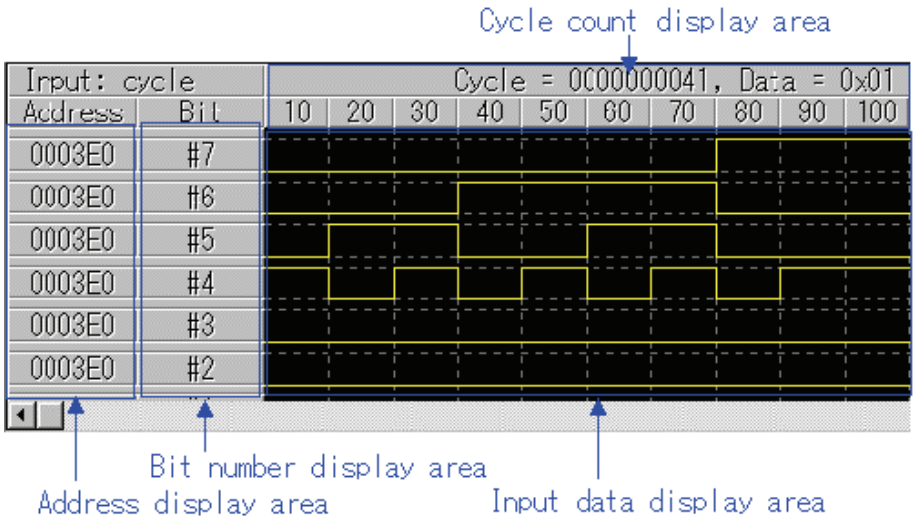
As shown above, virtual interrupts (in this case, timer A0 interrupt) can be generated in interval-synchronized.

6.2.4 Structure of Virtual Port Input Screen

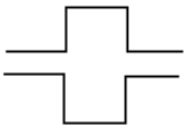
6.2.4.1 Screen structure for cycle-synchronized inputs

If you've set virtual port inputs that are synchronized to machine cycles, they can be displayed in one of the three modes shown below. The display modes can be changed from the Mode menu.

1. Chart mode (displayed in units of bits)
   The virtual port input that has been set is displayed in chart mode in units of bits.



- Address display area displays the memory address to which a virtual port is input.
- Bit number display area displays bit numbers of memory to which a virtual port is input.
- Input data display area displays the virtual port input data that has been set in chart mode in units of bits.



This means that memory bits are in the state of logic 1.

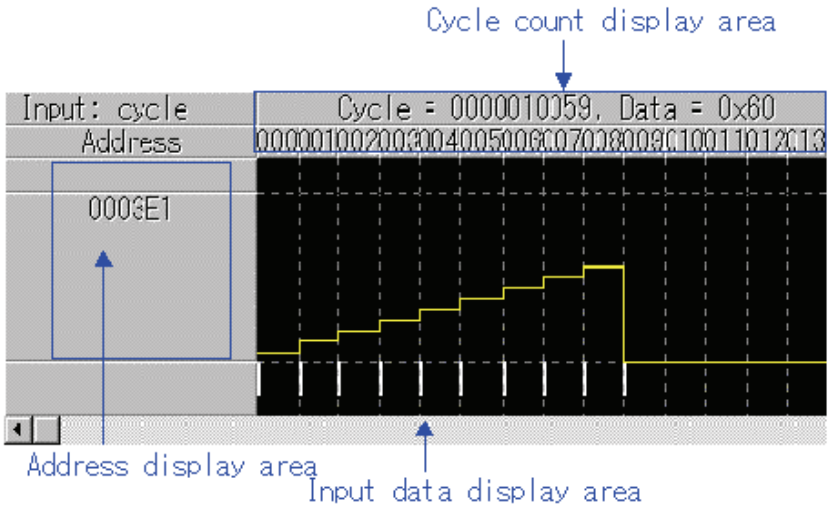
This means that memory bits are in the state of logic 0.

The short white lines appearing at the bottom of the input data display area indicate points at which data are input.

To reference data values, move the mouse cursor into this area and the value and the cycle count of the data at which the cursor is positioned will be displayed in the cycle count display area.

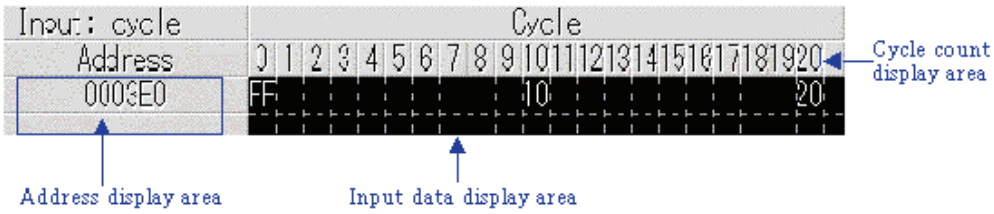
- Cycle count display area displays cycle counts.

2. Graphic mode (displayed in units of bytes)
   The virtual port input that has been set is displayed in graphic mode in units of bytes.



- Address display area displays the memory address to which a virtual port is input.
- Input data display area displays the virtual port input data that has been set in graphic mode.
  The peaks in this graph represent data values derived by equally dividing the height of the data-displaying area by 255 (maximum value of 1-byte data).
  The short white lines appearing at the bottom of the input data display area indicate points at which data are input.
  To reference data values, move the mouse cursor into this area and the value and the cycle count of the data at which the cursor is positioned will be displayed in the cycle count display area.
- Cycle count display area displays cycle counts.

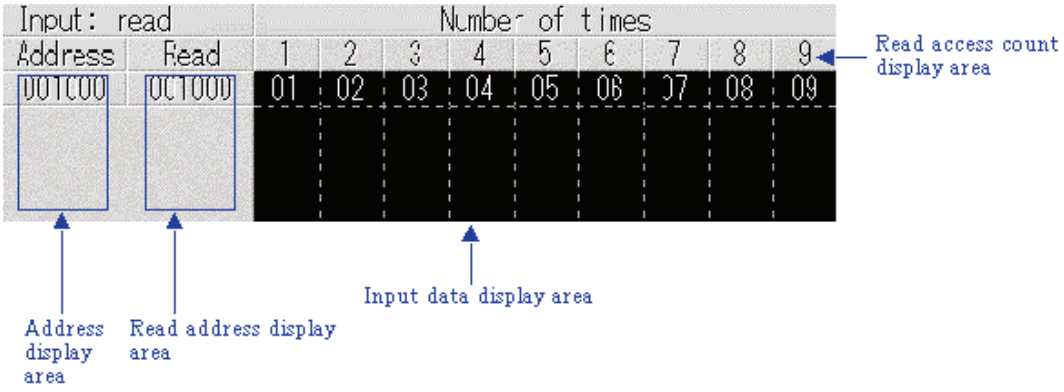
3. Hexadecimal mode
   The virtual port input that has been set is displayed in hexadecimal mode.



- Address display area displays the memory address to which a virtual port is input.
- Input data display area displays the virtual port input data that has been set by hexadecimal numbers.
  To reference data values, move the mouse cursor into this area and the value and the cycle count of the data at which the cursor is positioned will be displayed in the cycle count display area.
- Cycle count display area displays cycle counts.

6.2.4.2 Screen structure for read access-synchronized inputs

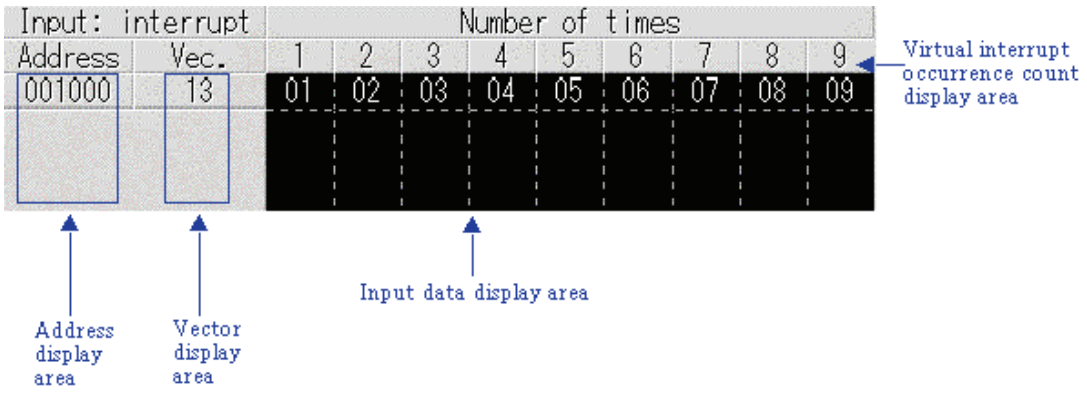
When you've set virtual port inputs that are synchronized to memory accesses for read, a display screen configured as shown below will appear.



- Address display area displays the memory address to which a virtual port is input.
- Read address display area displays the address to be monitored for read access.
- Input data display area displays the virtual port input data that has been set by hexadecimal numbers.
  To reference data values, move the mouse cursor into this area and the value and the read access count of the data at which the cursor is positioned will be displayed in the read access count display area.
- Read access count display area displays read access counts.

6.2.4.3 Screen structure for interrupt-synchronized inputs

When you've set virtual port inputs that are synchronized to virtual interrupts, a display screen configured as shown below will appear.

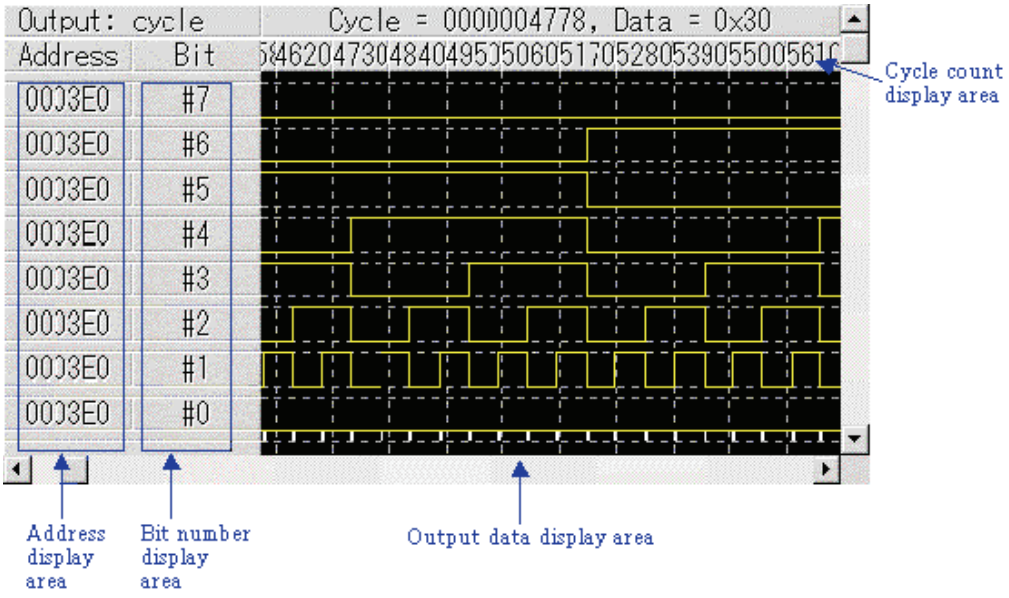


- Address display area displays the memory address to which a virtual port is input.
- Vector number display area displays the virtual interrupt vector number to be monitored.
- Input data display area displays the virtual port input data that has been set by hexadecimal numbers.
  To reference data values, move the mouse cursor into this area and the value and the virtual interrupt occurrence count of the data at which the cursor is positioned will be displayed in the virtual interrupt occurrence count display area. Virtual interrupt occurrence count display area It displays virtual interrupt occurrence counts.
- Virtual interrupt occurrence count display area displays virtual interrupt occurrence counts.

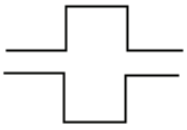
6.2.5 Structure of Virtual Port Output Screen

Virtual port output results can be displayed in one of the three modes shown below. The display modes can be changed from the Mode menu.

1. Chart mode (displayed in units of bits)
   Virtual port output results are displayed in chart mode in units of bits.



- Address display area displays the address to be monitored for virtual port output.
- Bit number display area displays bit numbers of memory being monitored for virtual port output.
- Output data display area displays the data as virtual port output results in chart mode in units of bits.



This means that memory bits are in the state of logic 1.

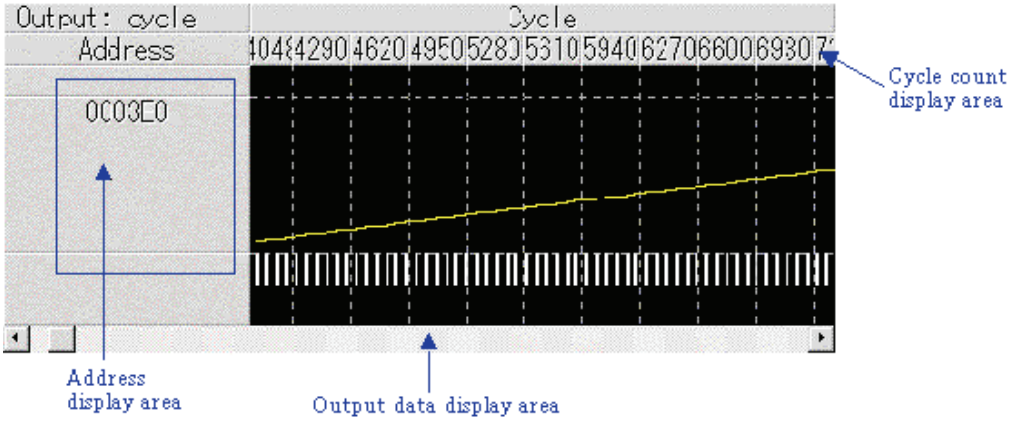
This means that memory bits are in the state of logic 0.

The short white lines appearing at the bottom of the output data display area indicate points at which data are output.

To reference data values, move the mouse cursor into this area and the value and the cycle count of the data at which the cursor is positioned will be displayed in the cycle count display area.

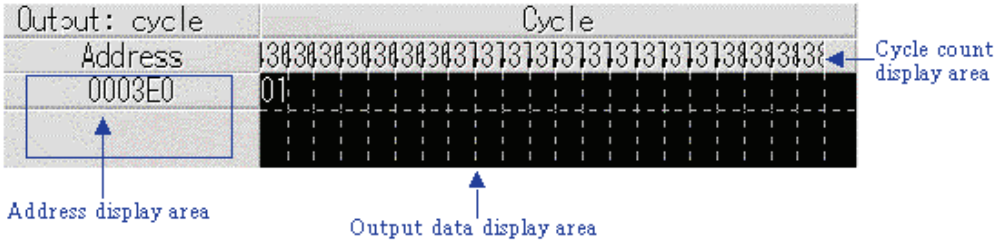
- Cycle count display area displays cycle counts.

2. Graphic mode (displayed in units of bytes)
   Virtual port output results are displayed in graphic mode in units of bytes.



- Address display area displays the address to be monitored for virtual port output.
- Output data display area displays the data as virtual port output results in graphic mode in units of bytes.
  The peaks [ ] in this graph represent data values derived by equally dividing the height of the data-displaying area by 255 (maximum value of 1-byte data).
  The short white lines appearing at the bottom of the output data display area indicate points at which data are output.
  To reference data values, move the mouse cursor into this area and the value and the cycle count of the data at which the cursor is positioned will be displayed in the cycle count display area.
- Cycle count display area displays cycle counts.

3. Hexadecimal mode
   Virtual port output results are displayed in hexadecimal mode.

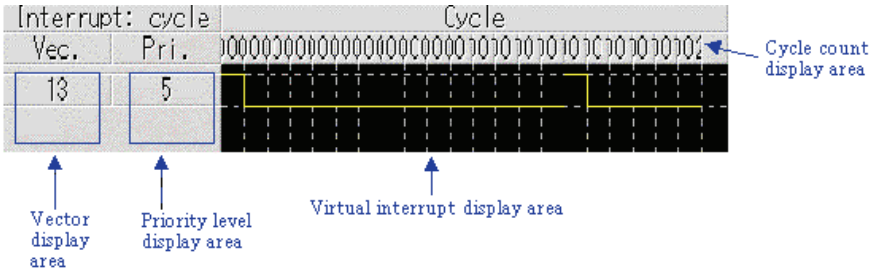


- Address display area displays the address to be monitored for virtual port output.
- Output data display area displays the data as virtual port output results by hexadecimal numbers.
  To reference data values, move the mouse cursor into this area and the value and the cycle count of the data at which the cursor is positioned will be displayed in the cycle count display area.
- Cycle count display area displays cycle counts.

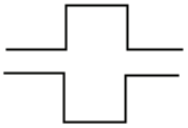
6.2.6 Structure of Virtual Interrupt Screen

6.2.6.1 Screen structure for cycle-synchronized interrupts

When you've set virtual interrupts that are synchronized to machine cycles, a display screen configured as shown below will appear.



- Vector number display area displays the vector number of a virtual interrupt.
- Priority level display area displays the priority level of a virtual interrupt.
- Virtual interrupt display area displays timing at which the virtual interrupt you've set is generated.



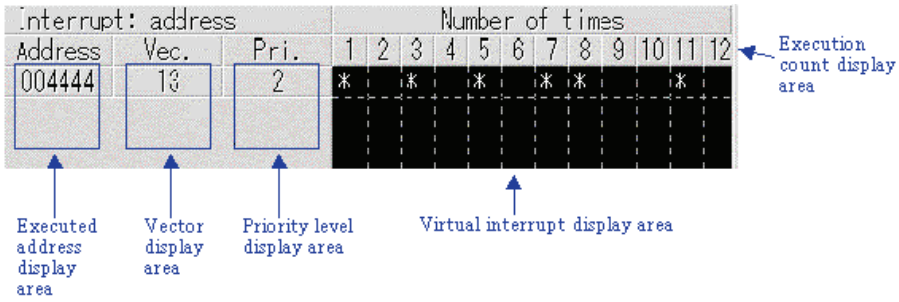
This means that a virtual interrupt is generated.

This means that a virtual interrupt is not generated.

- Cycle count display area displays cycle counts.

6.2.6.2 Screen structure for executed address-synchronized interrupts

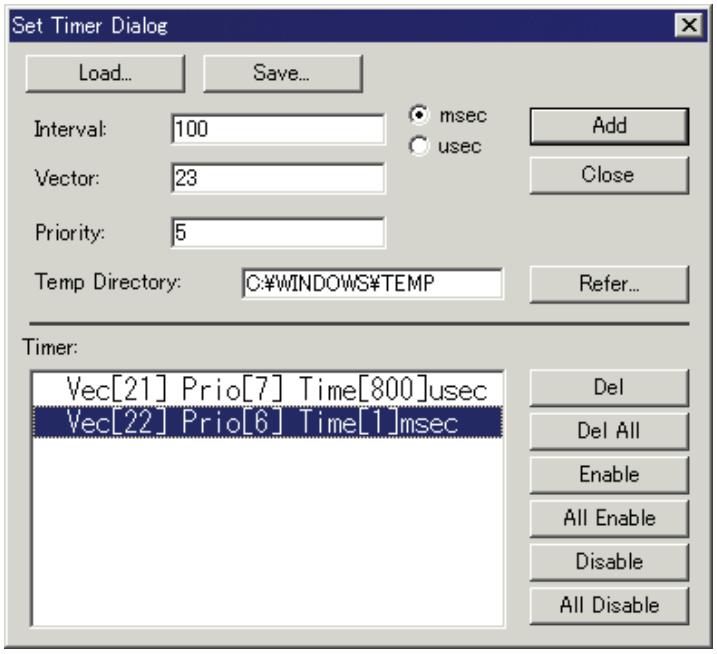
When you've set virtual interrupts that are synchronized to executed addresses, a display screen configured as shown below will appear.



- Executed address display area displays the fetch address (the address where the program is executed) at which time a virtual interrupt is generated.
- Vector number display area displays the vector number of a virtual interrupt.
- Priority level display area displays the priority level of a virtual interrupt.
- Virtual interrupt display area displays timings by an asterisk (*) at which the virtual interrupt you've set is generated.
  When an asterisk (*) is indicated, it means that a virtual interrupt is generated. When an asterisk (*) is not indicated, it means that a virtual interrupt is not generated.
- Execution count display area displays execution counts or a number of times the program has executed a specified address.

6.2.6.3 Screen configuration for interval-synchronized interrupts

To set a virtual interrupt which is synchronized to a specified interval time, click the Timer button and use the Timer dialog box that appears. The Timer dialog box has a display screen configuration similar to the one shown below.



- In the virtual interrupt register area, specify the virtual interrupt you want to set and the interval time at which intervals you want to generate the interrupt.
- The interval time is calculated from the number of execution cycles and the MCU clock and divide-by ratio specified on the MCU tab of the Init dialog box.
- The virtual interrupt display area shows the registered virtual interrupts and the specified interrupt generation intervals.
- The operation buttons for virtual interrupts can be used to delete or disable/enable each virtual interrupt.
- The operation buttons to save/load virtual interrupts can be used to save the virtual interrupt information to a file, as well as load the saved virtual interrupt information from the file.

6.2.7 Extended Menus

This window has the following popup menus that can be brought up by right-clicking in the window.

Menu	Function
Setup...	Sets up data(virtual port input, virtual port output, or virtual interrupt).
Modify...	Changes the setup data.
Delete...	Deletes the setup data.
Load...	Loads the saved setup data or I/O script file.
Mode...	Changes display mode.
Scale...	Changes display scale.
Color...	Changes display color.
Timer...	Sets the interval-synchronized interrupts.
I/O Script Files...	Lists registered I/O script files.
Previous Input data	Find the previous input data.
Next Input data	Find the next input data.
Previous Output data	Find the previous output data.
Next Output data	Find the next output data.
Previous Interrupt data	Find the previous interrupt data.
Next Interrupt data	Find the next interrupt data.
Toolbar display	Display toolbar.
Customize toolbar...	Open toolbar customize dialog box.
Allow Docking	Allow window docking.
Hide	Hide window.

6.2.8 Setting Virtual Port Inputs

The Virtual Port Input function allows you to simulate data inputs and similar other operations performed on the ports defined in the SFR. Data can be input to memory at one of the following timings:

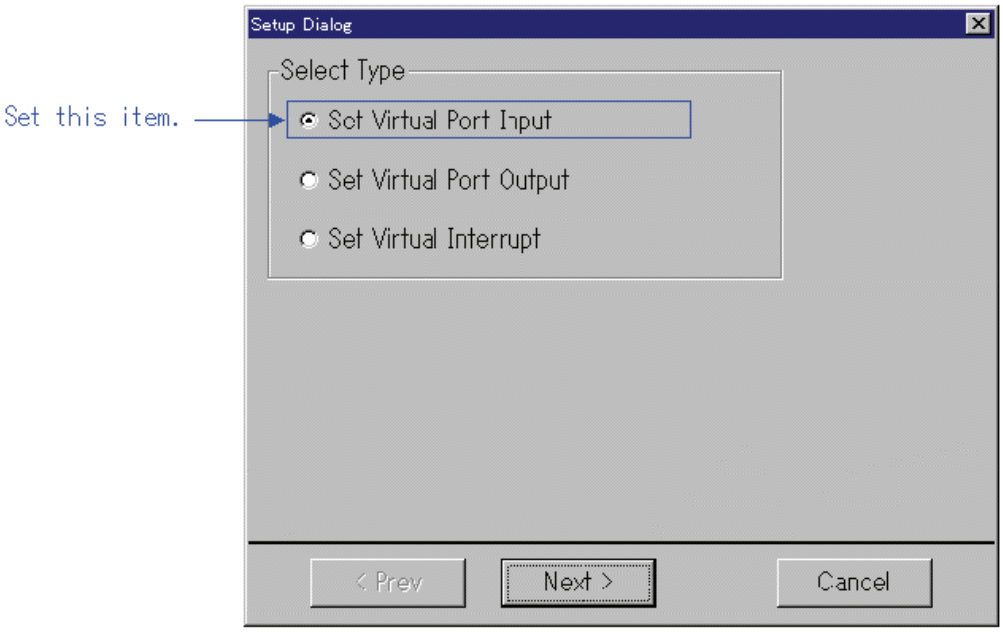
1. If you want data to be input to some memory location with the lapse of time Data can be input when program execution has reached a specified number of cycles. In this case, set cycle-synchronized inputs.
2. If you want data to be input when some memory location is read Data can be input when the program accesses a specified memory location for read. For example, this method can be used in cases where you want a variable (e.g.,global variables located at fixed addresses) to be assigned a value when it is read. In this case, set read access-synchronized inputs.
3. If you want data to be input when some virtual interrupt occurs Data can be input when a specified virtual interrupt is generated. For example, this method can be used in cases where memory for the SFR is referenced in an interrupt handler. In this case, set interrupt-synchronized inputs.

ATTENTION

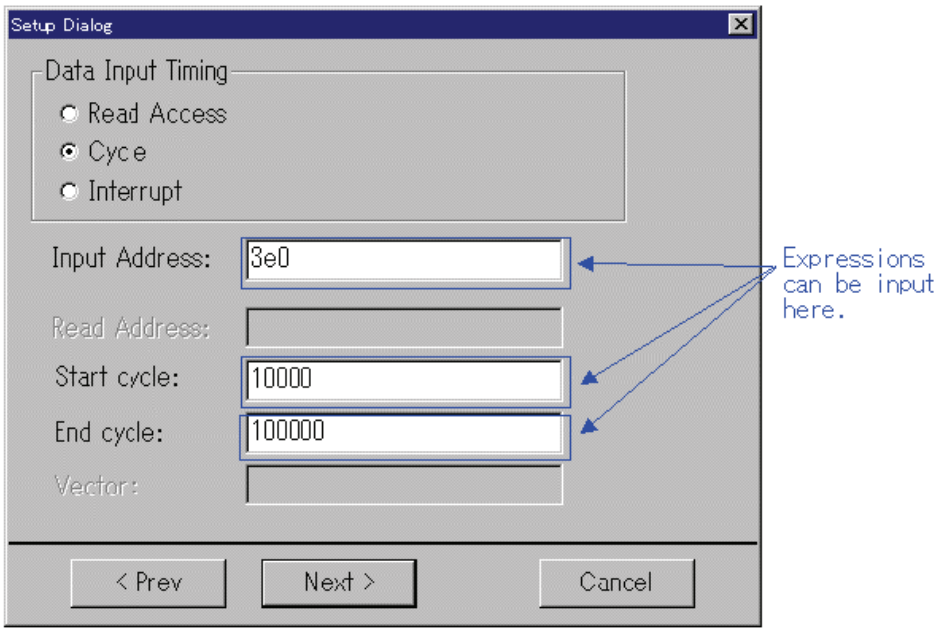
Up to a total of 50 virtual port input, virtual interrupt, and I/O script procedures can be set. However, if you are using the Printf function output function in the Output Port Window, the number of virtual port input, virtual interrupt, and I/O script procedures that can be set is limited to a total of 48.

6.2.8.1 Setting Cycle-synchronized Inputs

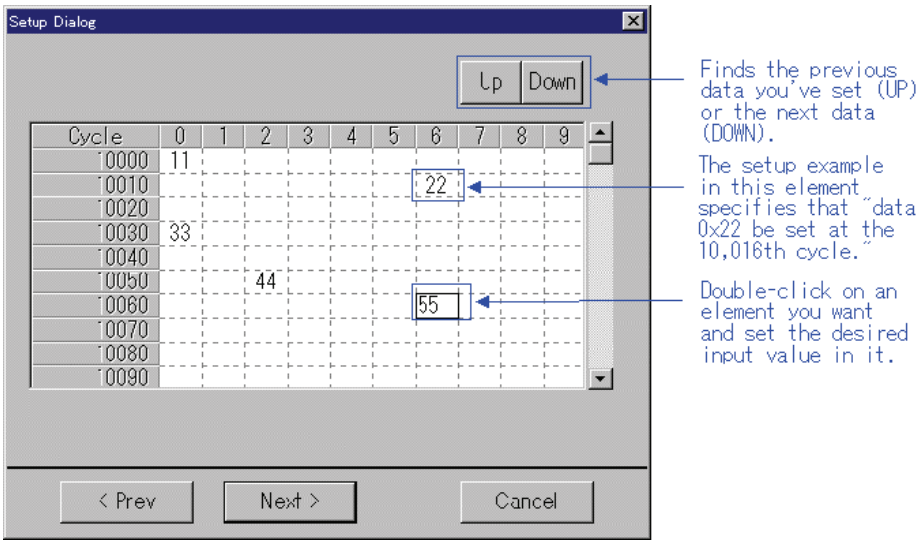
To set cycle-synchronized virtual port inputs, choose the [Setup...] menus in the I/O Timing Setting Window. The dialog box shown below will appear.



Here, choose the item Set Virtual Port Input and press the Next button. (Or press the Cancel button if you want cancel the setup session and close the dialog box.) A dialog box for setting up virtual port input timings will appear.



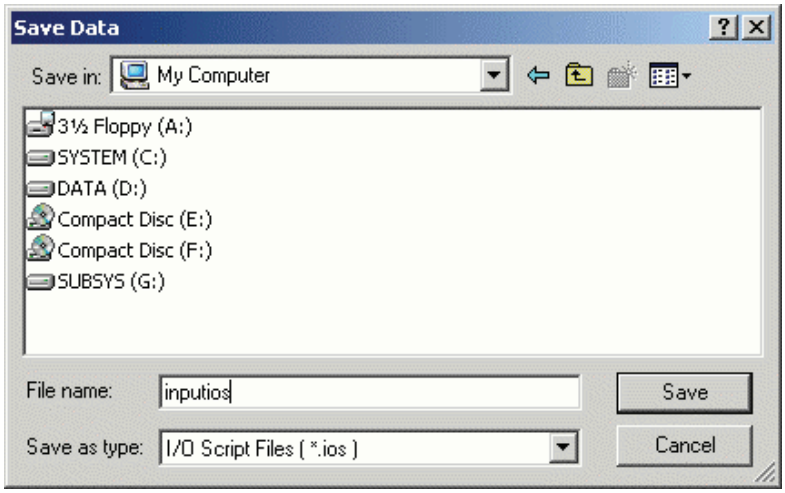
First, choose Cycle in the Data Input Timing column. Next, enter an address for virtual port input in the Input Address column (the address to which you want data to be input) using a hexadecimal number. Then enter the cycles at which you want the virtual port input to be started and ended for Start cycle and End cycle, respectively, using decimal numbers. Then press the Next button. (Or press the Prev button here if you want to return to the previous dialog box.)
A matrix dialog box for setting the virtual port input data will appear.



In this dialog box, set the data you want to be actually input to memory.
Follow the procedure below to set data:

1. Move the mouse cursor to the "cycles" location (called an element) where you want data to be set, then double-click the left mouse button. (Or you can scroll the screen to go to the desired location.)
2. Input data in the selected place using a hexadecimal number. The data size that can be input is one byte (from 0x0 up to 0xFF).
3. Repeat steps 1 and 2 as many times as the number of data you want to set.

When you finished entering all data, press the Next button.
A dialog box for saving the virtual port input data you've set to a file (virtual port input file) will appear.



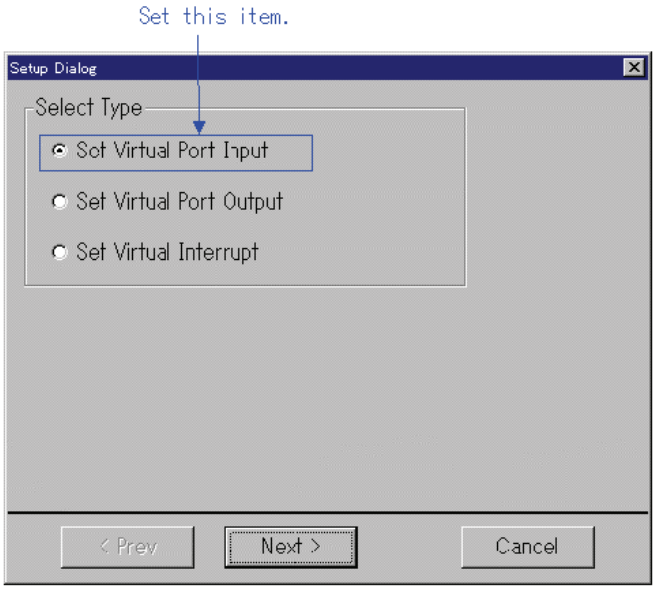
Here, enter the directory and file names in which you want the data you've set to be saved. The saved file can be loaded into the simulator debugger back again by using the [Load...] menus in the I/O Timing Setting window.

When you've input a file name, press the Save button.

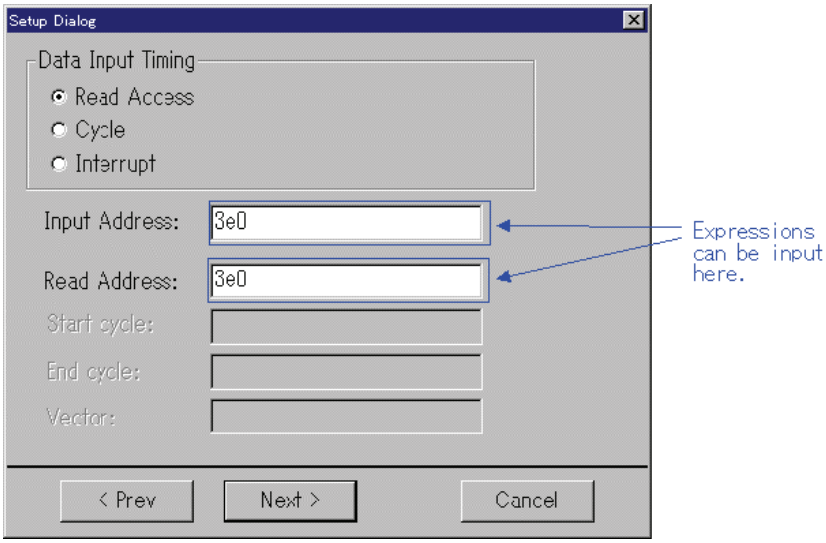
Thus, you've finished setting the cycle-synchronized virtual port inputs.

6.2.8.2 Setting Read Access-synchronized Inputs

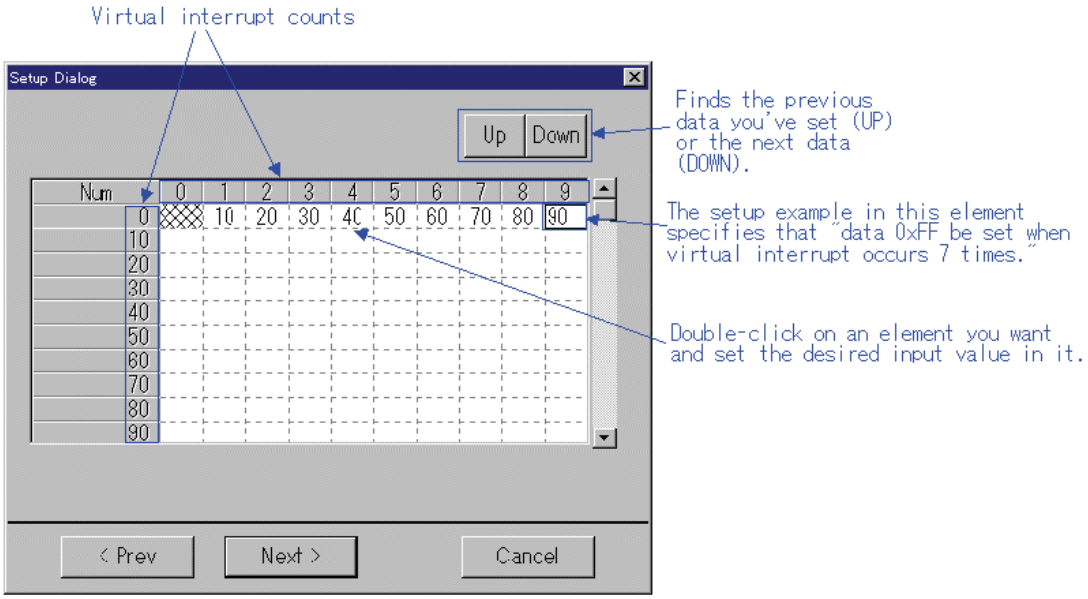
To set read access-synchronized virtual port inputs, choose the [Setup...] menus in the I/O Timing Setting Window. The dialog box shown below will appear.



Here, choose the item Set Virtual Port Input and press the Next button. (Or press the Cancel button if you want cancel the setup session and close the dialog box.) A dialog box for setting up virtual port input timings will appear.



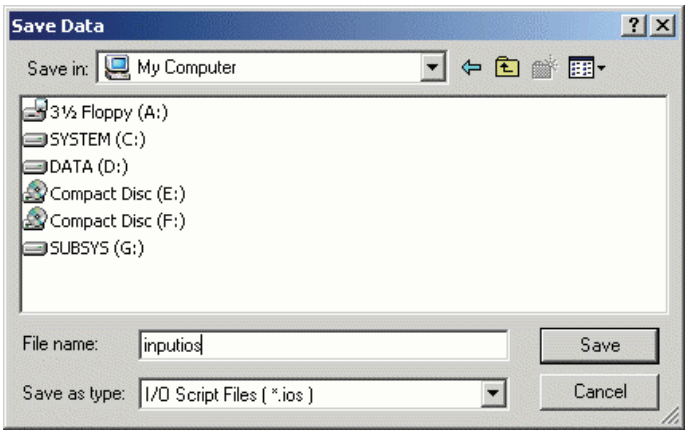
First, choose Read Access in the Data Input Timing column. Next, enter an address for virtual port input in the Input Address column (the address to which you want data to be input) using a hexadecimal number. Then enter the address to be accessed for read (to read data from memory) in the Read Address column. (Virtual port inputs are executed when the memory address you've specified here is accessed for read.) Then press the Next button. (Or press the Prev button here if you want to return to the previous dialog box.)
A matrix dialog box for setting the virtual port input data will appear.



In this dialog box, set the data you want to be actually input to memory.
Follow the procedure below to set data:

1. Move the mouse cursor to the "read access counts" location (called an element) where you want data to be set, then double-click the left mouse button. (Or you can scroll the screen to go to the desired location.)
2. Input data in the selected place using a hexadecimal number. The data size that can be input is one byte (from 0x0 up to 0xFF).
3. Repeat steps 1 and 2 as many times as the number of data you want to set.

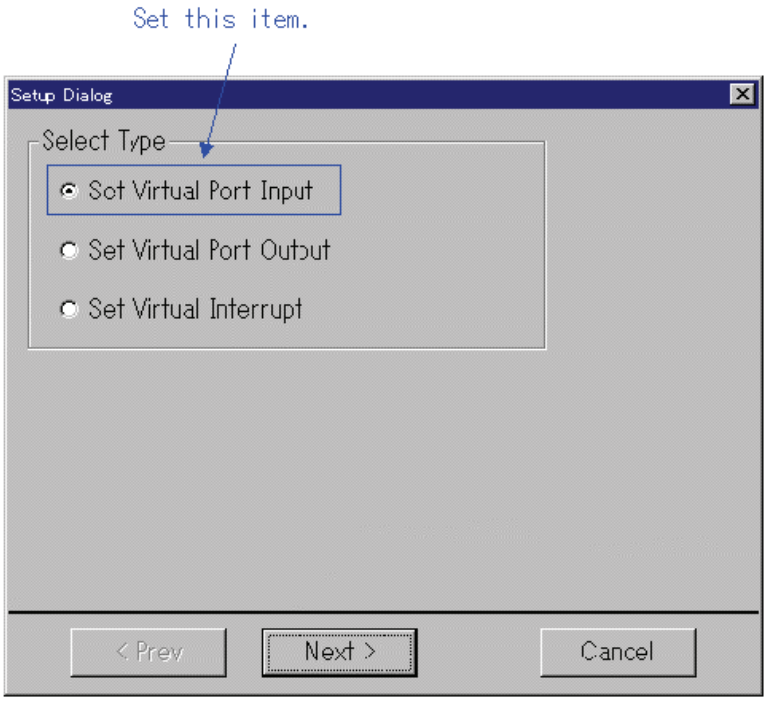
When you finished entering all data, press the Next button.
A dialog box for saving the virtual port input data you've set to a file (virtual port input file) will appear.



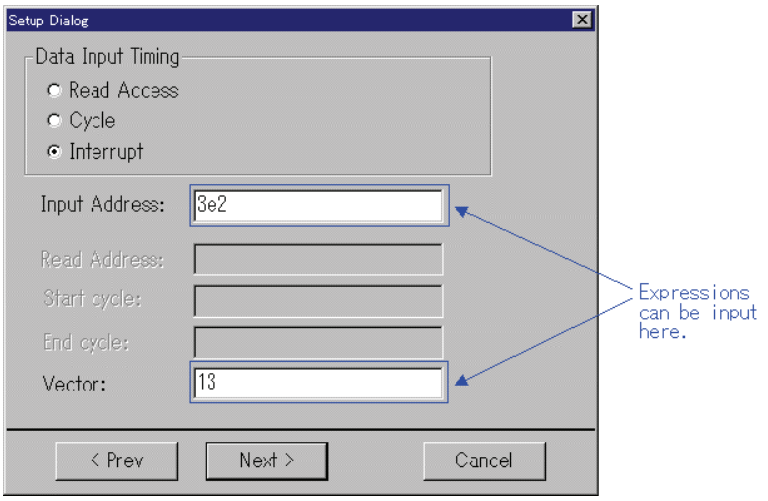
Here, enter the directory and file names in which you want the data you've set to be saved.
The saved file can be loaded into the simulator debugger back again by using the [Load...] menus in the I/O Timing Setting window.
When you've input a file name, press the Save button.
Thus, you've finished setting the read access-synchronized virtual port inputs.

6.2.8.3 Setting Interrupt-synchronized Inputs

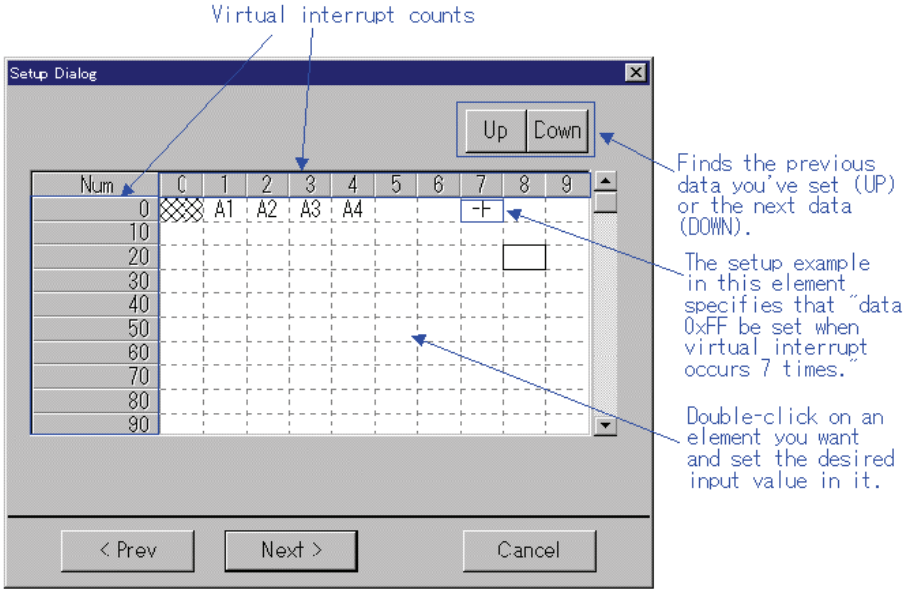
To set interrupt-synchronized virtual port inputs, choose the [Setup...] menus in the I/O Timing Setting Window. The dialog box shown below will appear.



Here, choose the item Set Virtual Port Input and press the Next button. (Or press the Cancel button if you want cancel the setup session and close the dialog box.) A dialog box for setting up virtual port input timings will appear.



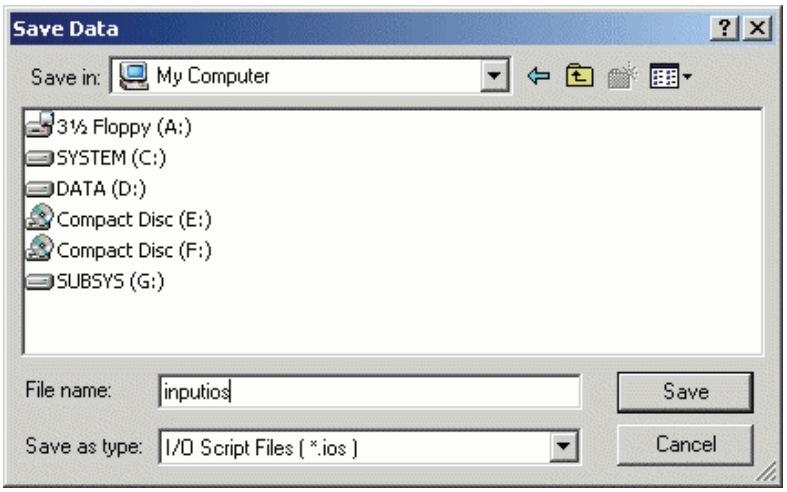
First, choose Interrupt in the Data Input Timing column. Next, enter an address for virtual port input in the Input Address column (the address to which you want data to be input) using a hexadecimal number. Then enter the vector number of a virtual interrupt that signals timing for virtual port input in the Vector column. (For PD32RSIM, the vector addresses are fixed.) Then press the Next button. (Or press the Prev button here if you want to return to the previous dialog box.)
A matrix dialog box for setting the virtual port input data will appear.



In this dialog box, set the data you want to be actually input to memory. Follow the procedure below to set data:

1. Move the mouse cursor to the "virtual interrupt counts" location (called an element) where you want data to be set, then double-click the left mouse button. (Or you can scroll the screen to go to the desired location.)
2. Input data in the selected place using a hexadecimal number. The data size that can be input is one byte (from 0x0 up to 0xFF).
3. Repeat steps 1 and 2 as many times as the number of data you want to set.

When you finished entering all data, press the Next button.
A dialog box for saving the virtual port input data you've set to a file (virtual port input file) will appear.



Here, enter the directory and file names in which you want the data you've set to be saved.
The saved file can be loaded into the simulator debugger back again by using the [Load...] menus in the I/O Timing Setting window.
When you've input a file name, press the Save button.
Thus, you've finished setting the virtual interrupt-synchronized virtual port inputs.

6.2.9 Setting Virtual Port Outputs

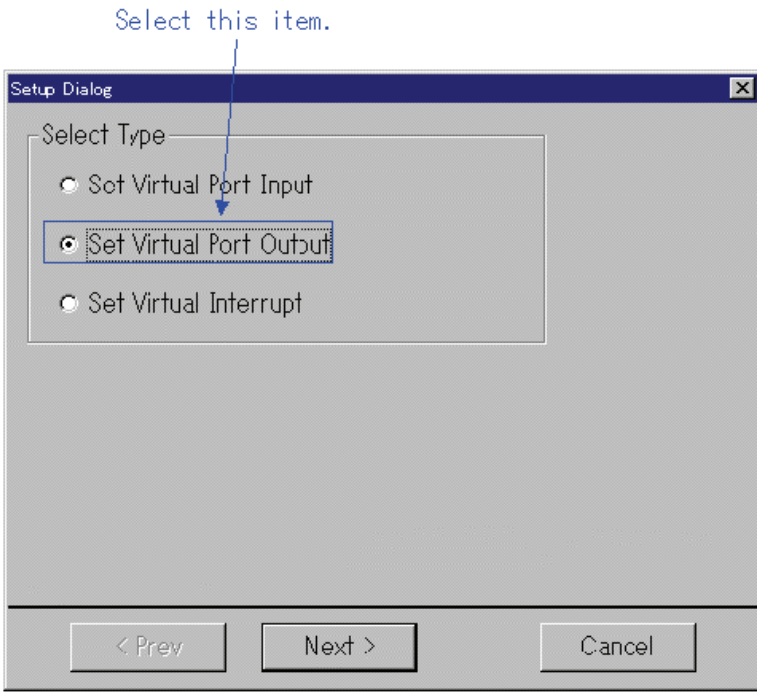
The Virtual Port Output function allows data values written to some memory address by a program to be recorded along with cycles at which data was written. The recorded data can be displayed for verification in graphic or numeric form.

ATTENTION

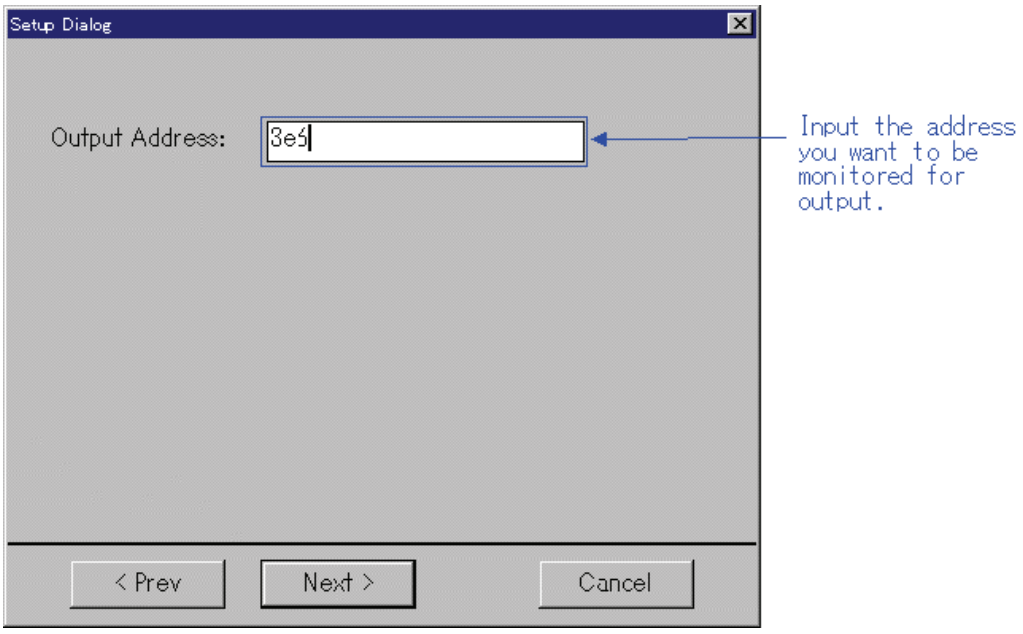
- The number of data entries recorded is the number of entries specified on the Init dialog box's I/O Scrip tab reckoning from when the program started to run. When reexecuted, the previous data is cleared.
- Up to 200 instances of virtual port output can be set. However, if you are using the Output Port Window, the number of virtual port outputs that can be set is limited to 199.

6.2.9.1 Setting Virtual Port Outputs

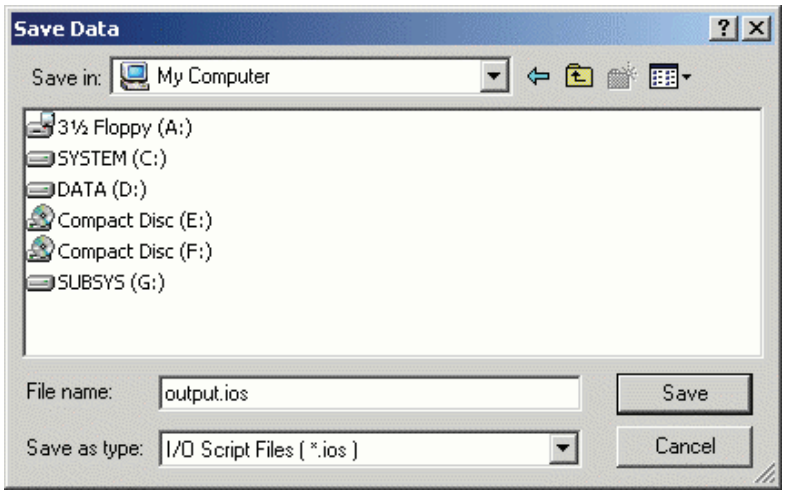
To set virtual port outputs, choose the [Setup...] menus in the I/O Timing Setting Window. The dialog box shown below will appear.



Here, choose the item Set Virtual Port Output and press the Next button. (Or press the Cancel button if you want cancel the setup session and close the dialog box.) A dialog box for setting the address you want to be monitored for virtual port output will appear.



Input the address you want to be monitored for virtual port output in the Output Address column. Then press the Next button.
A dialog box for specifying a file (virtual port output file) to which you want the virtual port output results to be saved (recorded) will appear. (This simulator debugger saves the virtual port output results that have occurred during program execution to this file and references it when the program stops running.)



Here, choose the item Set Virtual Port Output and press the Next button. (Or press the Cancel button if you want cancel the setup session and close the dialog box.) A dialog box for setting the address you want to be monitored for virtual port output will appear.

6.2.10 Setting Virtual Interrupts

The Virtual Interrupt function allows you to generate interrupts in a simulated manner without having to actually generate them. Using this function you can generate timer interrupts or key input interrupts in a simulated manner.
Virtual interrupts can be generated at one of the following timings:

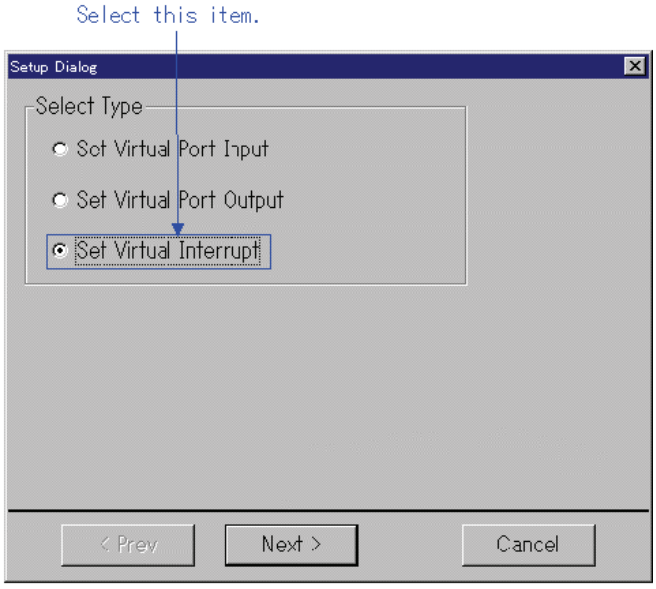
1. If you want virtual interrupts to be generated with the lapse of time
   Virtual interrupts can be generated when program execution has reached a specified number of cycles.
   In this case, set cycle-synchronized interrupts.
2. If you want virtual interrupts to be generated when the program executes a specified address Use this method if you want virtual interrupts to be generated when some specific function is executed.
   n this case, set executed address-synchronized interrupts.
3. When generating a virtual interrupt at fixed intervals
   Use the virtual interrupt function when you want to generate a virtual interrupt at fixed intervals.
   In this case, set the interval-synchronized interrupts.

ATTENTION

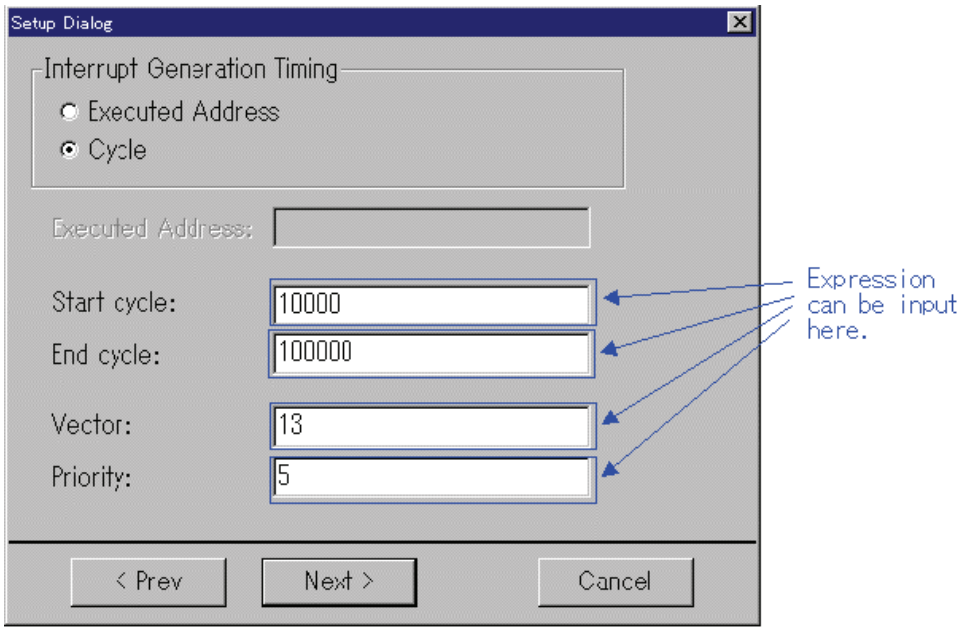
Up to a total of 50 virtual port input, virtual interrupt, and I/O script procedures can be set. However, if you are using the Printf function output function in the Output Port Window, the number of virtual port input, virtual interrupt, and I/O script procedures that can be set is limited to a total of 48.

6.2.10.1 Setting Cycle-synchronized Interrupts

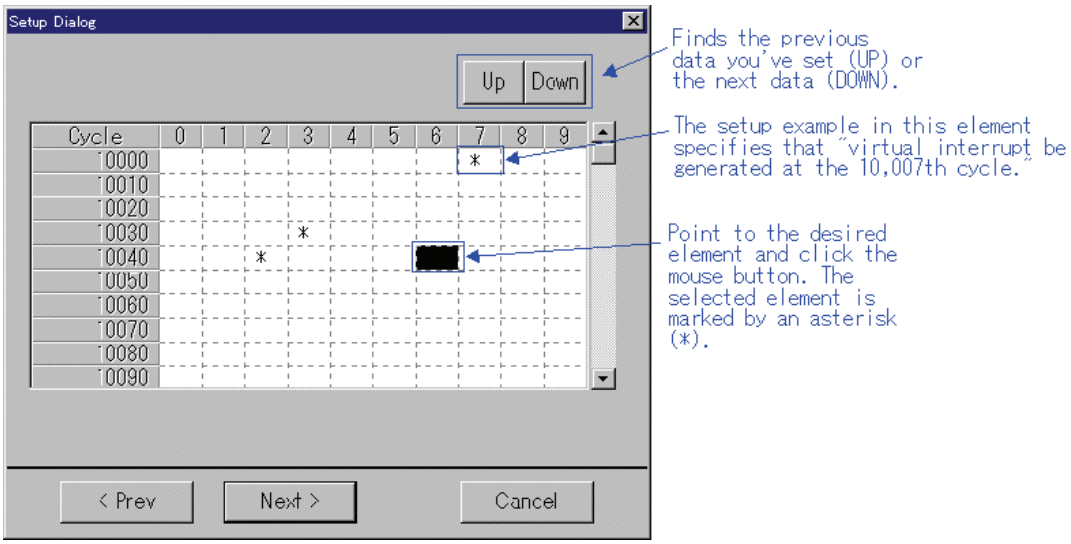
To set cycle-synchronized virtual interrupts, choose the [Setup...] menus in the I/O Timing Setting Window. The dialog box shown below will appear.



Here, choose the item Set Virtual Interrupt and press the Next button. (Or press the Cancel button if you want cancel the setup session and close the dialog box.) A dialog box for setting up virtual interrupt timings will appear.



First, choose Cycle in the Interrupt Generation Timing column. Next, specify the cycles at which you want a virtual interrupt to be started and ended for Start cycle and End cycle, respectively, using decimal numbers. Then specify the vector number and the priority of the virtual interrupt to be generated for Vector and Priority, respectively, using decimal numbers. Then press the Next button. (Or press the Prev button here if you want to return to the previous dialog box.)
A matrix dialog box for setting virtual interrupts will appear.

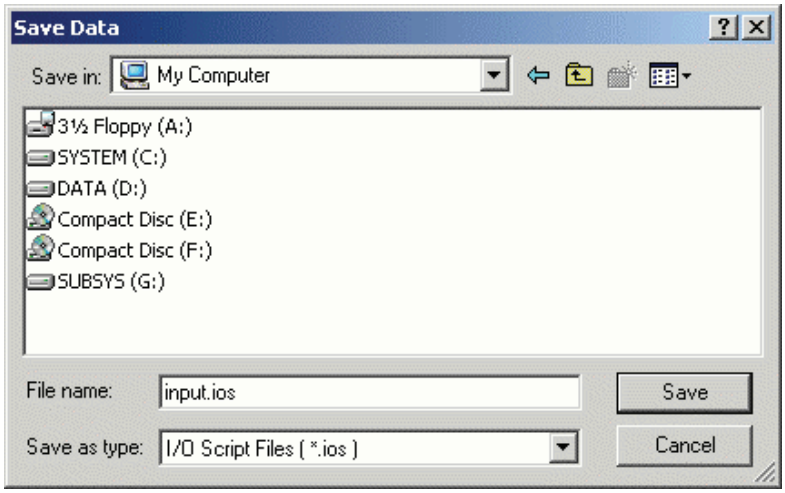


In this dialog box, set the virtual interrupts you want to be actually generated. Follow the procedure below to set virtual interrupts:

1. Move the mouse cursor to the "cycles" location (called an element) where you want a virtual interrupt to be generated, then click the left mouse button. (Or you can scroll the screen to go to the desired location.)
2. The element is marked by an asterisk (*) when you've clicked. Click at the same place again if you want the virtual interrupt you've set to be canceled. In this case, the asterisk goes out.
3. Repeat steps 1 and 2 as many times as the number of virtual interrupts to be generated.

When you finished setting all virtual interrupts, press the Next button.

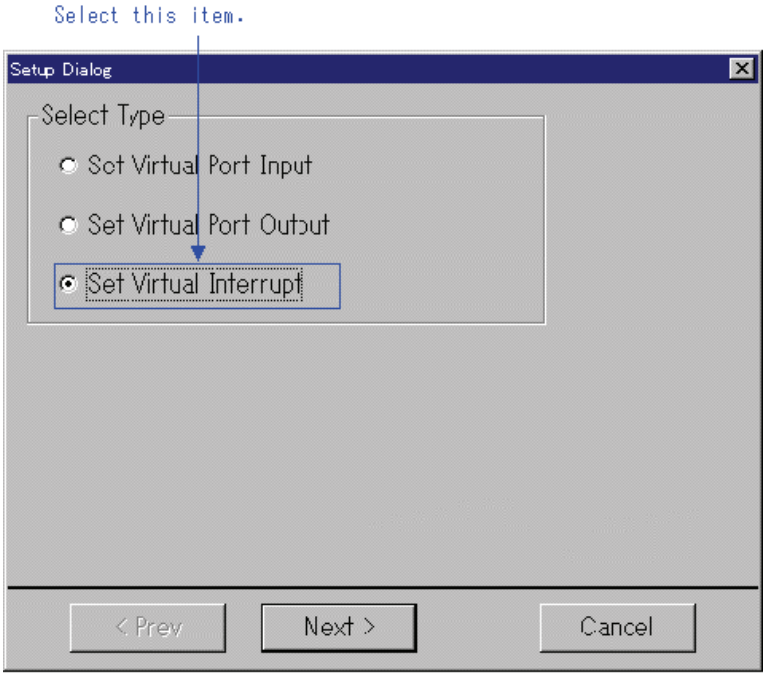
A dialog box for saving the virtual interrupts you've set to a file (virtual interrupt file) will appear.



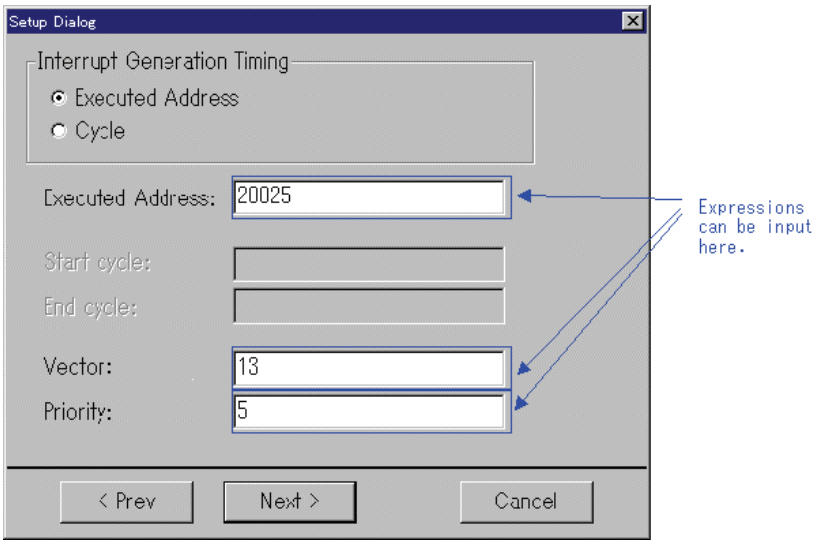
Here, enter the directory and file names in which you want the data you've set to be saved.
The saved file can be loaded into the simulator debugger back again by using the [Load...] menus in the I/O Timing Setting window.
When you've input a file name, press the Save button.
Thus, you've finished setting cycle-synchronized virtual interrupts.

6.2.10.2 Setting Executed Address-synchronized Interrupts

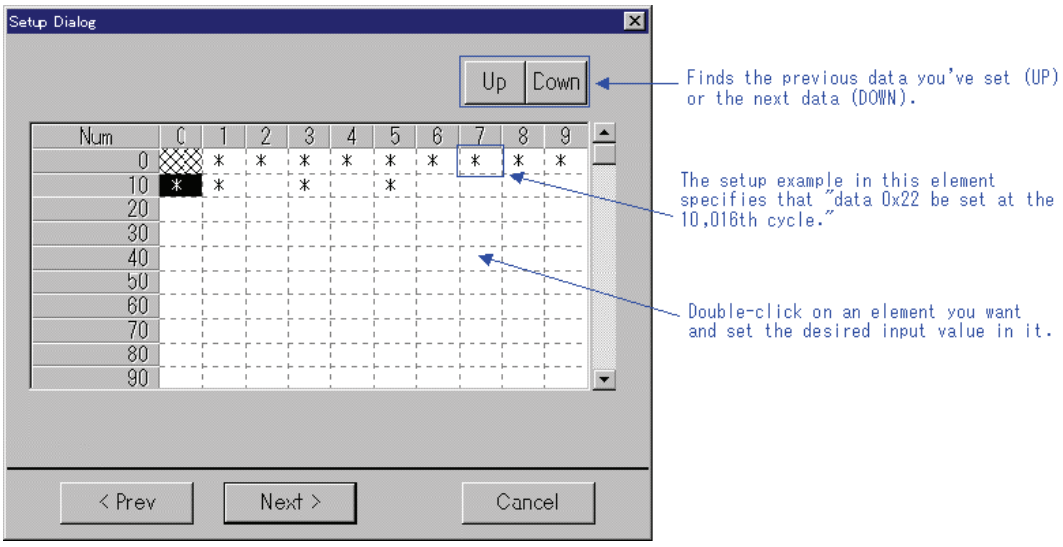
To set executed address-synchronized virtual interrupts, choose the [Setup...] menus in the I/O Timing Setting Window. The dialog box shown below will appear.



Here, choose the item Set Virtual Interrupt and press the Next button. (Or press the Cancel button if you want cancel the setup session and close the dialog box.) A dialog box for setting up virtual interrupt timings will appear.



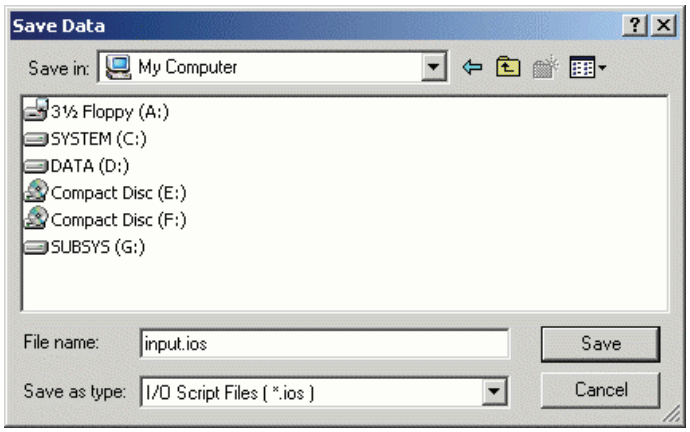
First, choose Executed Address in the Interrupt Generation Timing column. Next, specify the executed address (i.e., the address at which a virtual interrupt is generated when it is executed) for Executed Address. Then specify the vector number and the priority of the virtual interrupt to be generated for Vector and Priority, respectively, using decimal numbers. Then press the Next button. (Or press the Prev button here if you want to return to the previous dialog box.)
A matrix dialog box for setting virtual interrupts will appear.



In this dialog box, set the virtual interrupts you want to be actually generated. Follow the procedure below to set virtual interrupts:

1. Move the mouse cursor to the "cycles" location (called an element) where you want a virtual interrupt to be generated, then click the left mouse button. (Or you can scroll the screen to go to the desired location.)
2. The element is marked by an asterisk (*) when you've clicked. Click at the same place again if you want the virtual interrupt you've set to be canceled. In this case,the asterisk goes out.
3. Repeat steps 1 and 2 as many times as the number of virtual interrupts to be generated.

When you finished setting all virtual interrupts, press the Next button.
A dialog box for saving the virtual interrupts you've set to a file (virtual interrupt file) will appear.



Here, enter the directory and file names in which you want the data you've set to be saved.
The saved file can be loaded into the simulator debugger back again by using the [Load...] menus in the I/O Timing Setting window.
When you've input a file name, press the Save button.
Thus, you've finished setting executed address-synchronized virtual interrupts.

6.2.10.3 Setting interval-synchronized interrupts

To set virtual interrupts which are synchronized to fixed intervals, choose the [Timer...] menus in the I/O Timing Setting window. This opens a dialog box necessary to set virtual interrupts which are synchronized to fixed intervals.
In this dialog box, you can set the following items:

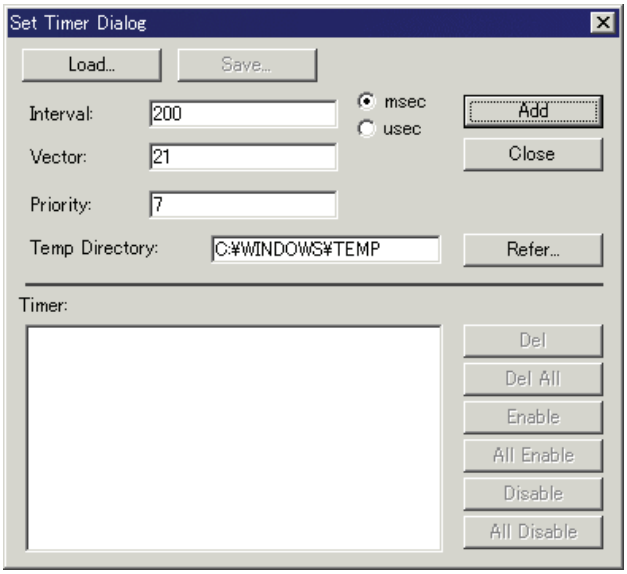
- Setting a virtual interrupt
- Deleting a virtual interrupt
- Temporarily disabling a virtual interrupt
- Reenabling a disabled virtual interrupt
- Saving a virtual interrupt
- Loading a virtual interrupt

The following shows how to specify each item.

6.2.10.3.1. Setting a virtual interrupt

Shown below is an example specification with the M16C/6X simulator debugger.
Example: Generate an interrupt of vector number 21 with priority level (IPL) = 7 every 200 ms

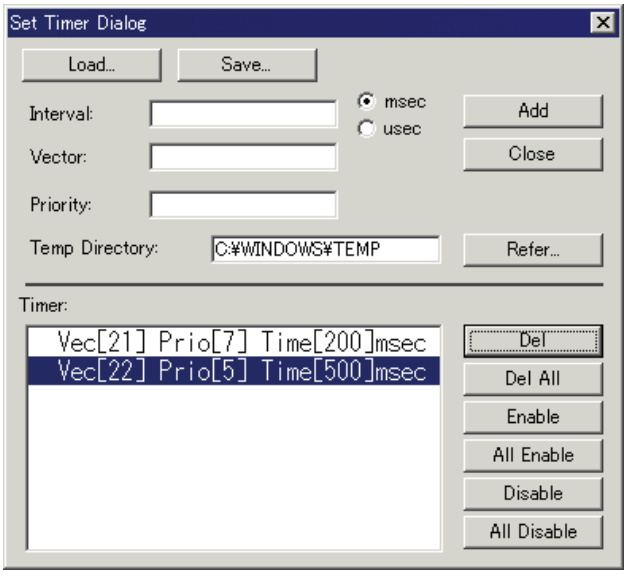
Fill out the dialog box as shown below. For the Temp Directory area, set a writable directory because it is a temporary area used internally by the debugger to set virtual interrupts.



Click the Add button, and the virtual interrupt you've set is added to the list of virtual interrupts in the lower part of the dialog box.
When you've finished setting virtual interrupts, click the Close button

6.2.10.3.2. Deleting a virtual interrupt

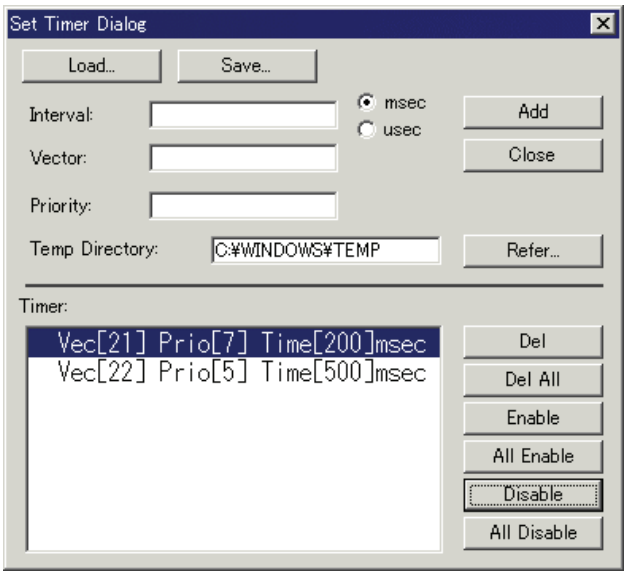
Click to select the virtual interrupt you want to delete from the list of virtual interrupts in the lower part of the dialog box. Then click the Del button



To delete all virtual interrupts, click the DelAll button.
When you've finished deleting virtual interrupts, click the Close button.

6.2.10.3.3. Temporarily disabling a virtual interrupt

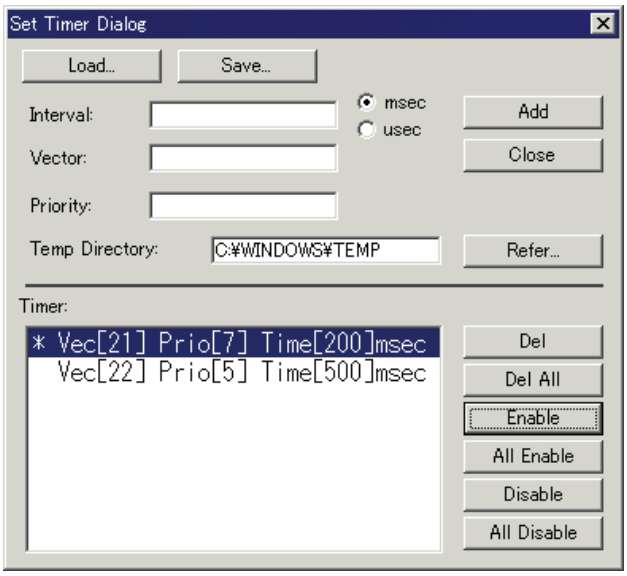
Click to select the virtual interrupt you want to temporarily disable from the list of virtual interrupts in the lower part of the dialog box. Then click the Disable button.
Or you can double-click on a virtual interrupt to temporarily disable it.



The temporarily disabled virtual interrupt is marked with an asterisk (*) to the left of the virtual interrupt list.
To temporarily disable all virtual interrupts, click the All Disable button.
When you've finished setting virtual interrupts, click the Close button.

6.2.10.3.4. Reenabling a disabled virtual interrupt

Click to select the virtual interrupt you want to reenable from the list of virtual interrupts in the lower part of the dialog box. Then click the Enable button. Or you can double-click on a virtual interrupt to reenable it.



The reenabled virtual interrupt has its asterisk (*) shown to the left of the virtual interrupt list disappeared. To reenable all virtual interrupts, click the All Enable button. When you've finished setting virtual interrupts, click the Close button.

6.2.10.3.5. Saving a virtual interrupt

Click the Save button on the dialog box. A file selection dialog box appears. In this dialog box, specify the file name to which to save virtual interrupts. If the extension is omitted, the extension ".stm" is automatically added.

6.2.10.3.6. Loading a virtual interrupt

Click the Load button on the dialog box. A file selection dialog box appears. In this dialog box, specify the file name from which to load virtual interrupts. The virtual interrupts loaded from the file are added to the currently set virtual interrupts.

6.2.11 Regarding Evaluation Timings of Virtual Port Inputs,Virtual Interrupts, and I/O Script Files Set

The virtual port inputs, virtual interrupts, and I/O script files you've set are evaluated at the following timings:

6.2.11.1 Evaluation timings

1. When program is executed (continuously); when come is executed
2. When program is single-stepped
3. When program is overstepped
4. When control is returned

6.2.11.2 Processing when program is reset

The virtual port inputs, virtual interrupts, and I/O script files that you've set are reevaluated. Namely, when a program is reset, the virtual port inputs, virtual interrupts, and I/O script files you've set are set newly again.

6.2.11.3 Processing when I/O Window is closed

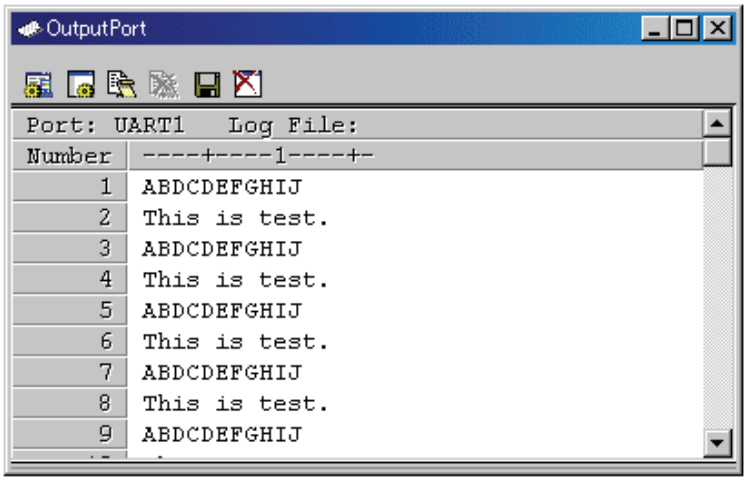
If the I/O Window is closed, the virtual port inputs, virtual interrupts, and I/O script files that you've set are not evaluated. This case is the same as when their settings have been deleted.

6.3 Output Port Window

The Output Port Window is used to display the data to be output to ports on a window or output the data to a file.
The debugger for 740 doesn't support this function.

It also allows you to verify the data that is output to UARTs by the Printf function.

Output Destination	Format
Window	ASCII display
	Hexadecimal display
File	ASCII output
	Hexadecimal output
	Binary output (Not including data output for the Printf function, however)



- For the output port, you can select any port or UART0 or UART1 which is the output destination for the Printf function. For details about the Printf function output destination, see the User's Manual of your Renesas C Compiler NCxx.
- The data which are output to ports can be saved to a specified file (log file) before being presented to the Output Port Window.
- The Output Port Window has a buffer that contains 10,000 bytes of the latest execution result. Even when you forgot to specify a log file, the data which are output to ports can be saved to a file (view file).
- The output data is displayed on a window or output to a file when the target program has stopped.

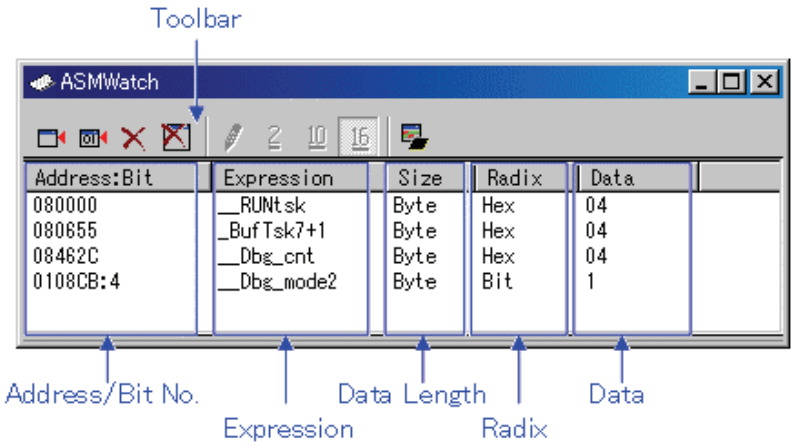
6.3.1 Extended Menus

This window has the following popup menus that can be brought up by right-clicking in the window.

Menu		Function
Set...		Sets output port.
Colm...		Sets column.
Log	On...	Open log file and start recording (start output to file).
	Off	Close log file and end recording (stop output to file).
View	Save...	Save view buffer to file.
	Clear	Clear view buffer.
Toolbar display		Display toolbar.
Customize toolbar...		Open toolbar customize dialog box.
Allow Docking		Allow window docking.
Hide		Hide window.

6.4 ASM Watch Window

The ASM watch window is a window in which you can register specific addresses as watchpoints and inspect memory contents at those addresses.
If a registered address resides within the RAM monitor area, the memory content at that address is updated at given intervals (by default, every 100 ms) during program execution.



- The addresses to be registered are called the "watchpoints." One of the following can be registered:
  - Address (can be specified using a symbol)
  - Address + Bit number
  - Bit symbol
- The registered watchpoints are saved in the debugger when the ASM watch window is closed and are automatically registered when the window is reopened.
- If symbols or bit symbols are specified for the watchpoints, the watchpoint addresses are recalculated when downloading the target program.
- The invalid watchpoints are marked by "-<not active>-" when displayed on the screen.
- The order in which the watchpoints are listed can be changed by a drag-and-drop operation.
- The watchpoint expressions, sizes, radixes and datas can be changed by in-place editing.

ATTENTION

- The RAM monitor obtains the data accessed through the bus. Any change other than the access from the target program will not be reflected.
- If the display data length of the RAM monitor area is not 1 byte, the data's access attribute to the memory may varies in units of 1 byte. In such a case that the access attribute is not unified within a set of data, the data's access attribute cannot be displayed correctly. In this case, the background colors the access attribute color of the first byte of the data.

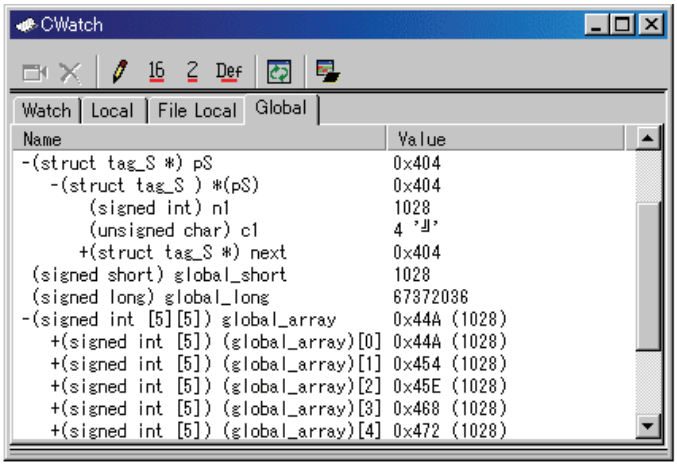
6.4.1 Extended Menus

This window has the following popup menus that can be brought up by right-clicking in the window.

Menu		Function
Add...		Add watchpoint.
Add Bit...		Add bit-lebel watchpoint.
Remove		Remove the selected watchpoint.
Remove All		Remove all watchpoints.
Set...		Set new data to selected watchpoint.
Radix	Bin	Display in Binary.
	Dec	Display in Decimal.
	Hex	Display in Hexadecimal.
Refresh		Refresh memory data.
Layout	Address Area	Switch display or non-display of Address area.
	Size Area	Switch display or non-display of Size area.
RAM Monitor	Enable RAM Monitor	Switch enable or disable RAM moniter function.
	Sampling Period...	Set RAM monitor sampling period.
Toolbar display		Display toolbar.
Customize toolbar...		Open toolbar customize dialog box.
Allow Docking		Allow window docking.
Hide		Hide window.

6.5 C Watch Window

The C Watch Window displays C/C++ expressions and their values (results of calculations).
The C/C++ expressions displayed in the C Watch Window are known as C watchpoints. The displays of the results of calculating the C watchpoints are updated each time a command is executed.
When RAM monitor function is effective and the C watch points are within the RAM monitor area, the displayed values are updated during execution of the target program.



- Variables can be inspected by scope (local, file local or global).
- The display is automatically updated at the same time the PC value changes.
- Variable values can be changed.
- The display radix can be changed for each variable individually.
- Any variable can be registered to the Watch tab, so that it will be displayed at all times:
  - The registered content is saved for each project separately.
  - If two or more of the C watch window are opened at the same time, the registered
- The C watchpoints can be registered to separate destinations by adding Watch tabs.
- Variables can be registered from another window or editor by a drag-and-drop operation.
- The C watchpoints can be sorted by name or by address.
- Values can be inspected in real time during program execution by using the RAM monitor function.

ATTENTION

- You cannot change the values of the C watch points listed below:
  - Bit field variables
  - Register variables
  - C watch point which does not indicate an address(invalid C watch point)
- If a C/C++ language expression cannot be calculated correctly (for example, when a C/C++ symbol has not been defined), it is registered as invalid C watch point.
  It is displayed as "--<not active>--". If that C/C++ language expression can be calculated correctly at the second time, it becomes an effective C watch point.
- The display settings of the Local, File Local and Global tabs are not saved. The contents of the Watch tab and those of newly added tabs are saved.
- The RAM monitor obtains the data accessed through the bus. Any change other than the access from the target program will not be reflected.
- The variables, which are changed in real-time, are global variables and file local variables only.
- If the display data length of the RAM monitor area is not 1 byte, the data's access attribute to the memory may varies in units of 1 byte. In such a case that the access attribute is not unified within a set of data, the data's access attribute cannot be displayed correctly. In this case, the background colors the access attribute color of the first byte of the data.

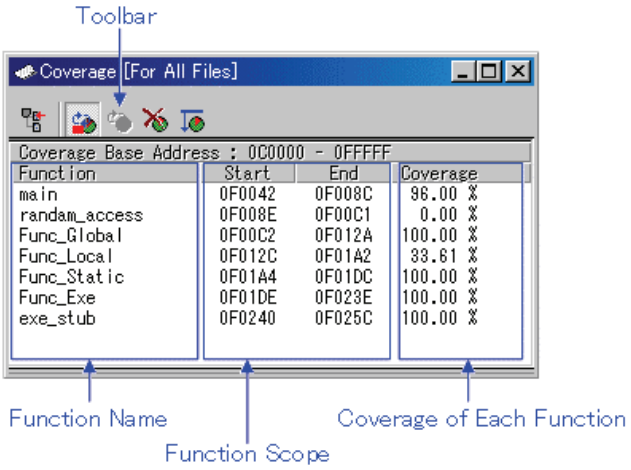
6.5.1 Extended Menus

This window has the following popup menus that can be brought up by right-clicking in the window.

Menu		Function
Add...		Add C watchpoint.
Remove		Remove the selected C watchpoint.
Remove All		Remove all C watchpoints.
Initialize		Reevaluates the selected C watchpoint.
Set New Value...		Set new data to selected C watchpoint.
Radix	Hex	Display in Hexadecimal.
	Bin	Display in Binary.
	Default	Display in Default Radix.
	Toggle(All Variables)	Change radix (toggle).
Refresh		Refresh memory data.
Hide type name		Hide type names from variables.
Show char* as string		Selects whether to display char* type as a string.
Sort	Sort by Name	Sort variables by its name.
	Sort by Address	Sort variables by its address.
RAM Monitor	Enable RAM Monitor	Switch enable or disable RAM moniter function.
	Sampling Period...	Set RAM monitor sampling period.
Add New Tab...		Add new tab.
Remove Tab		Remove the selected tab.
Toolbar display		Display toolbar.
Customize toolbar...		Open toolbar customize dialog box.
Allow Docking		Allow window docking.
Hide		Hide window.

6.6 Coverage Window

The Coverage window allows you to reference the coverage measurement result of the functions of the target program downloaded. The coverage which can be measured is C0 coverage.
Two types of windows are provided: the Coverage window in which you can check the start address/end address of the functions and coverage measurement results; and the Editor window in which you can check execution/non-execution by source line.



- All of the memory space is the target for coverage measurement.
- By double-clicking any function line, the corresponding function appears in the Editor(Source) window.
- During coverage measurement, "-%" appears in the coverage display area.
- You can change the display ratio between the function name display area and the function range display area, using the mouse.

6.6.1 Extended Menus

This window has the following popup menus that can be brought up by right-clicking in the window.

Menu		Function
Select source file...		Select a source file for checking the coverage.
Auto Refresh		Refresh coverage measurement result automatically.
Refresh		Refresh coverage measurement result.
Clear		Clear coverage measurement result .
Base...		Change coverage base address.
File	Save...	Save coverage measurement result to file.
	Load...	Load coverage measurement result from file.
Layout	Address	Switch display or non-display of Address area.
Toolbar display		Display toolbar.
Customize toolbar...		Open toolbar customize dialog box.
Allow Docking		Allow window docking.
Hide		Hide window.

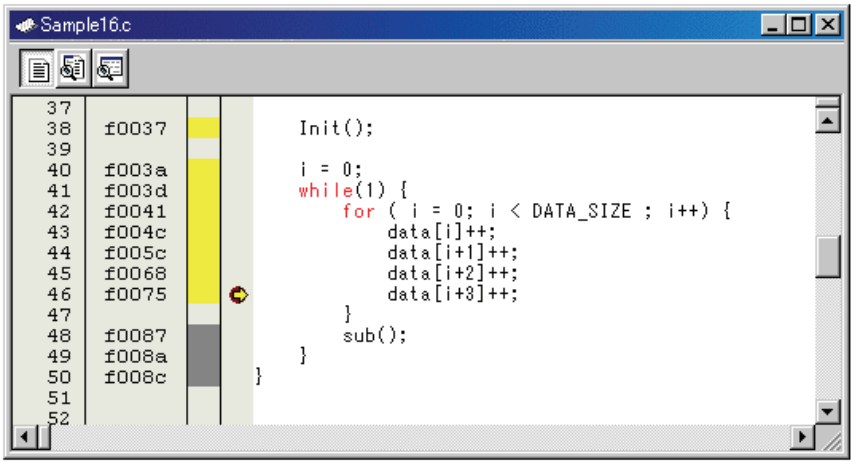
*:The simulator debugger doesn't support, because the entire memory area is coverage area.

6.6.2 Refer to the Source Line/the Executed Address

It is possible to refer in the Editor(Source) Window or Memory Window.

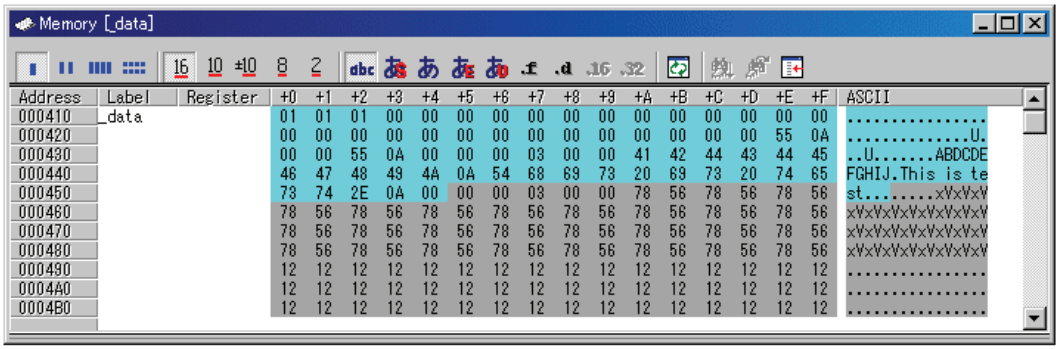
6.6.2.1 Refer in the Editor(Source) Window

In the Editor(Source) window, a display of Coverage Measurement is set to "Disable" by default.
To enable the display, check the [Coverage] check box in the dialog box opened by choosing the main menu - [Edit] -> [Define Column Format]. The column for a coverage measurement display is displayed on all Editor (Source) windows.
And select popup menu - [Columns] -> [Coverage] in the Editor (Source) window, A column can be set up for each Editor (Source) windows.



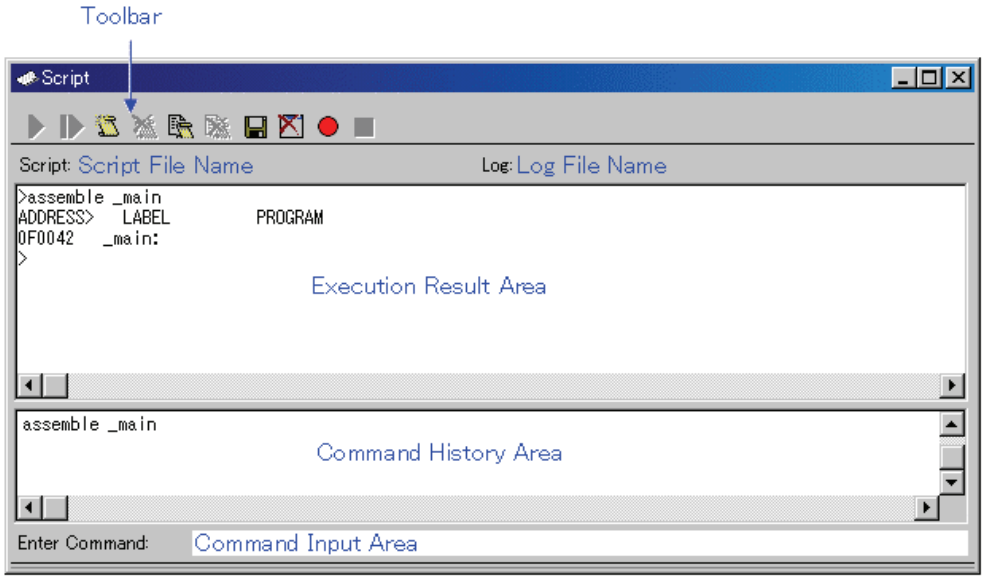
6.6.2.2 Refer in the Memory Window

In the Memory window, a display of Coverage Measurement is set to "Disable" by default.
To enable the display, select popup menu - [Coverage] -> [On/Off] in the Memory window.



6.7 Script Window

The Script Window displays the execution of text -format script commands and the results of that execution.
Script commands can be executed using a script file or interactively. You can also write script commands in the script file so that they are automatically executed. The results of script command execution can also be stored in a previously specified log file.



- The Script Window has a view buffer that stores the results of executing the last 1000 lines. The results of execution can therefore be stored in a file (view file) without specifying a log file.
- When a script file is opened, the command history area changes to become the script file display area and displays the contents of the script file. When script files are nested, the contents of the last opened script file are displayed. The script file display area shows the line currently being executed in inverse vide.
- When a script file is open, you can invoke script commands from the command input area provided the script file is not being executed.
- The Script Window can record the history of the executed commands to a file. This function is not the same as the log function. This function records not the result but only the executed commands, so the saved files can be used as the script files.

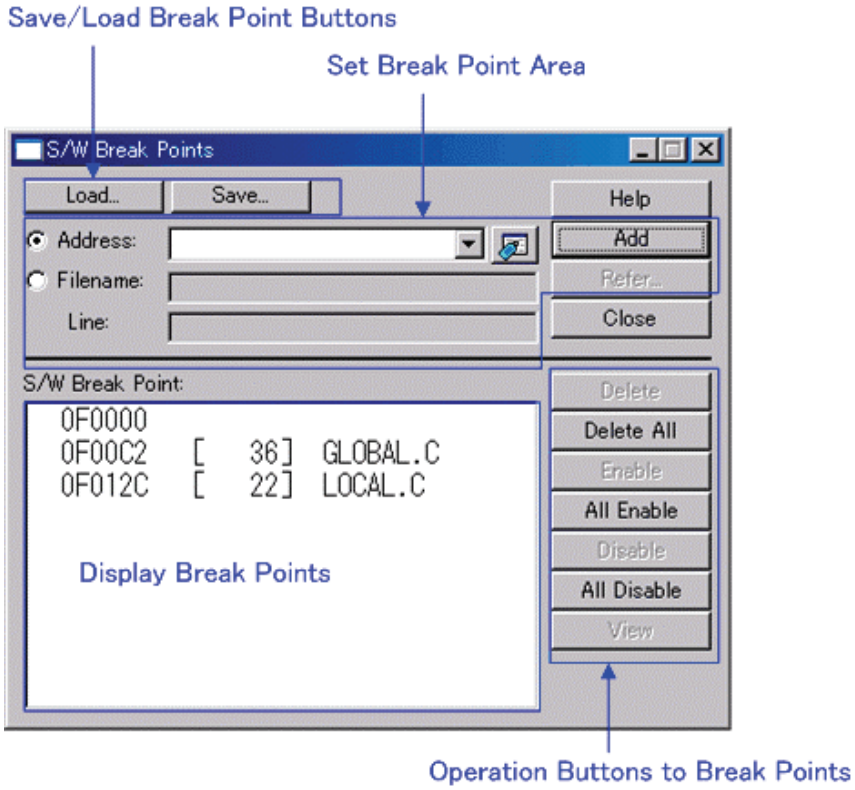
6.7.1 Extended Menus

This window has the following popup menus that can be brought up by right-clicking in the window.

Menu		Function
Script	Open...	Open script file.
	Run	Run script file.
	Step	One step execution of script file.
	Close	Close script file.
View	Save...	Save view buffer to file.
	Clear	Clear view buffer.
Log	On...	Open log file and start recording (start output to file).
	Off	Close log file and end recording (stop output to file).
Record	On...	Record the executed commands to a file.
	Off	Stop recording the executed commands.
Copy		Copy the selection and put it on the Clipboard.
Paste		Insert Clipboard contents.
Cut		Cut the selection and put it on the Clipboard.
Delete		Erase the selection.
Undo		Undo the last action.
Toolbar display		Display toolbar.
Customize toolbar...		Open toolbar customize dialog box.
Allow Docking		Allow window docking.
Hide		Hide window.

6.8 S/W Break Point Setting Window

The S/W Break Point Setting window allows you to set software break points.
Software breaks stop the execution of instructions immediately before the specified break point.



- If you have set multiple software breakpoints, program execution stops when any one software break address is encountered (OR conditions).
- You can continue to set software breakpoints until you click the "Close" button to close the S/W Break Point Setting Window.
- You can clear, enable or disable software breakpoints selected by clicking in the software breakpoint display area. You can also enable and disable software breakpoints by double-clicking on them.
- Click on the "Save" button to save the software break points in the file. To reload software break point settings from the saved file, click the "Load" button. If you load software break points from a file, they are added to any existing break points.

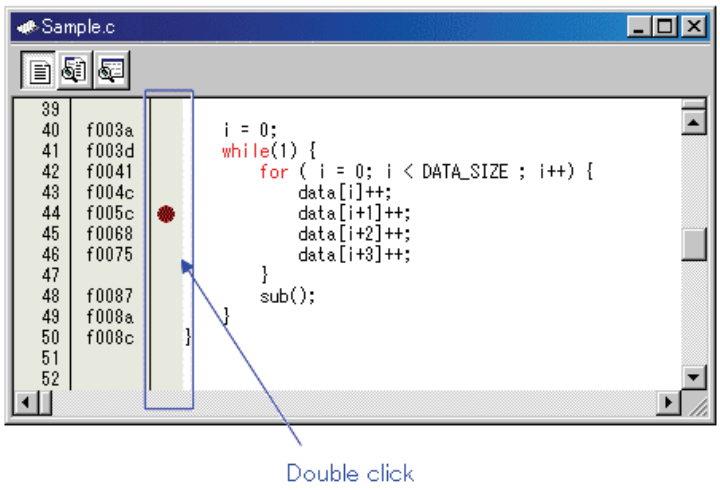
6.8.1 Command Button

The buttons on this window has the following meanings.

Button	Function
Load...	Load setting information from a file in which it was saved.
Save...	Save the contents set in the window to a file.
Help	Display the help of this window.
Add	Add the break point.
Refer...	Open file selection dialog box.
Close	Close the window.
Delete	Remove the selected break point.
Delete All	Remove all break points.
Enable	Enable the selected break points.
All Enable	Enable all break points.
Disable	Disable the selected break point.
All Disable	Disable all break points.
View	Shows the selected breakpoint positions in the Editor(Source) window.

6.8.2 Setting and Deleting a Break Points from Editor(Source) Window

The area which can be set in the software breakpoint is different according to the product. Please refer to "12.1.2 Area where software breakpoint can be set" for details.
You can set break points in the Editor(Source) Window. To do so, double-click the break point setting area ("S/W breakpoints" column) for the line in which you want to set the break. (A red marker is displayed on the line to which the break point was set.)

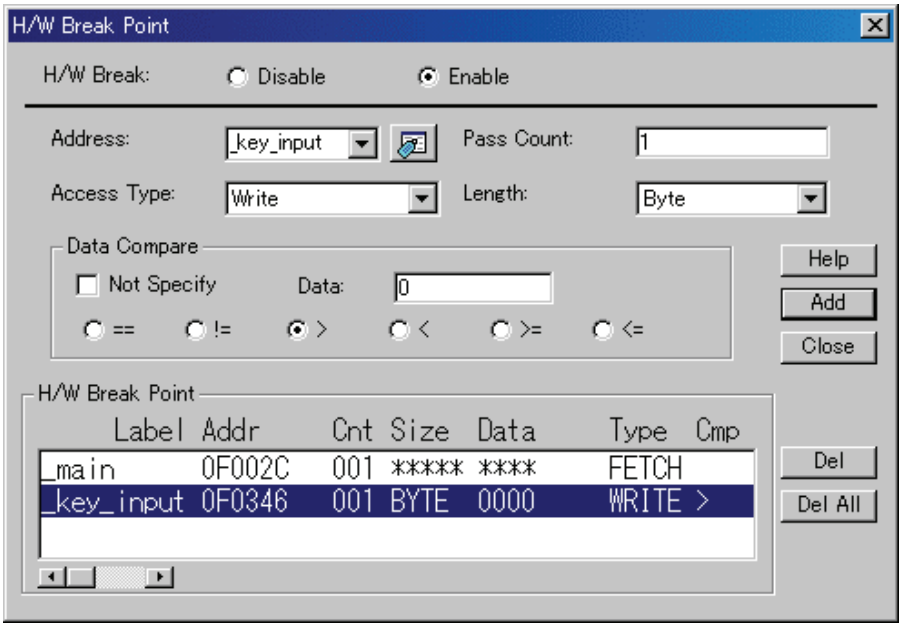


You can delete the break point by double-clicking again in the break point setting area ("S/W breakpoints" column).

In the Editor(Source) window, a display of "S/W breakpoints" column is set to "Enable" by default. To erase this column, deselect the [S/W breakpoints] check box in the dialog box opened by choosing the main menu - [Edit] -> [Define Column Format]. The "S/W breakpoints" column is erased from all Editor (Source) windows. And select popup menu - [Columns] -> [S/W breakpoints] in the Editor (Source) window, A column can be set up for each Editor (Source) windows.

6.9 H/W Break Point Setting Dialog Box

The H/W Break Point Setting dialog box allows you to set hardware break points.

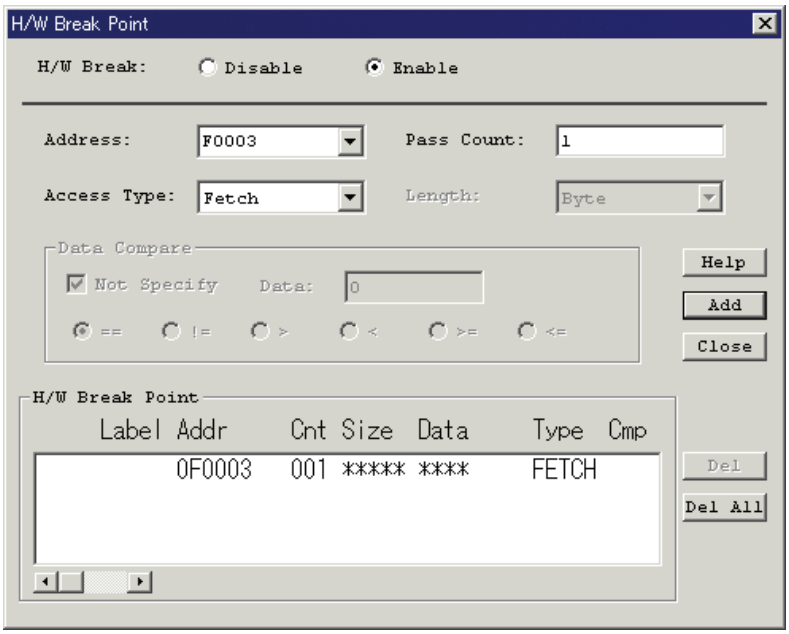


- The hardware break point of 64 points can be set up. You can set one address breakpoints with pass counts.
- As address break point access types, you can specify writing data to the address break point (Write), reading data from the address break point (Read), reading or writing data (R/W), and fetching instructions (Fetch).
- You can also specify that execution breaks if the data read from or written to the address break point has a specific value. Moreover, you can specify valid and invalid bits for the specific value.
- If you have set multiple hardware breakpoints, program execution stops when any one hardware break address is encountered (OR conditions).

6.9.1 Specify the Events

6.9.1.1 Instruction Fetch

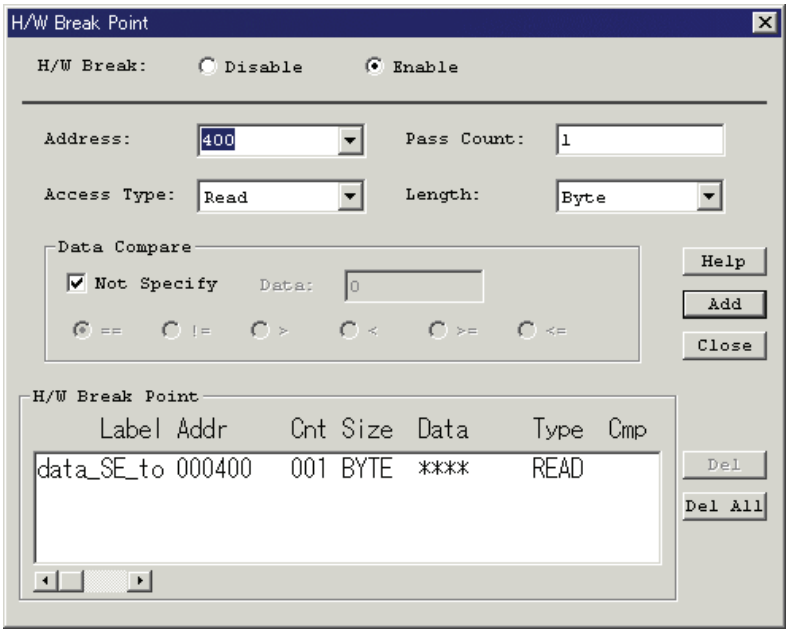
Set as below.
Example) Executing a instruction at address F0003h



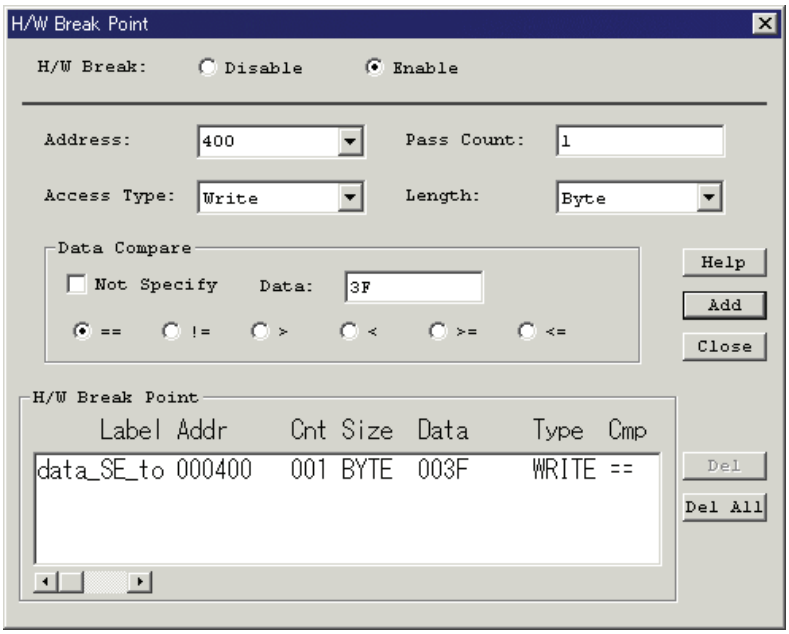
When the Add button is clicked, the breakpoint is added to the breakpoint list under the dialog.
Please click the Close button after completing the hardware breakpoint setting.

6.9.1.2 Memory Access

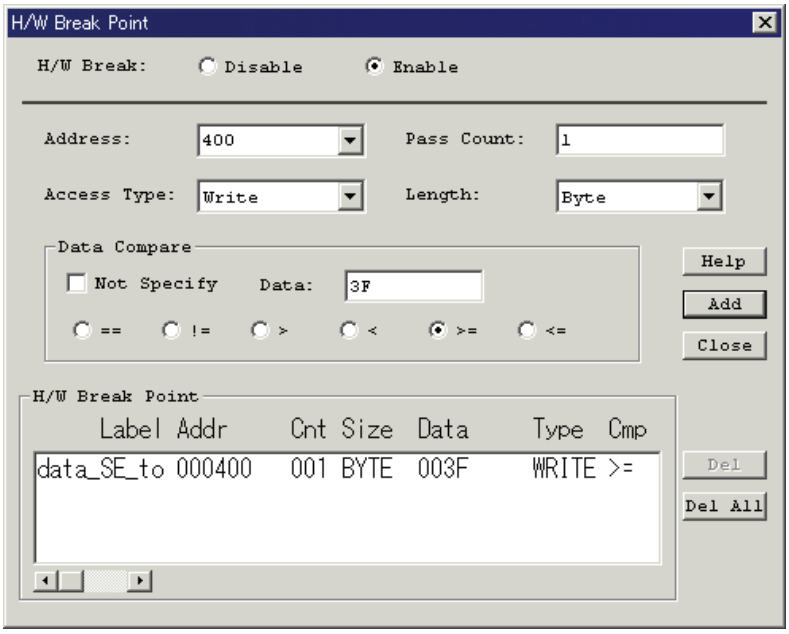
Set as below.
Example) Reading to even address 400h



Example) Writing byte length data 3Fh to even address 400h

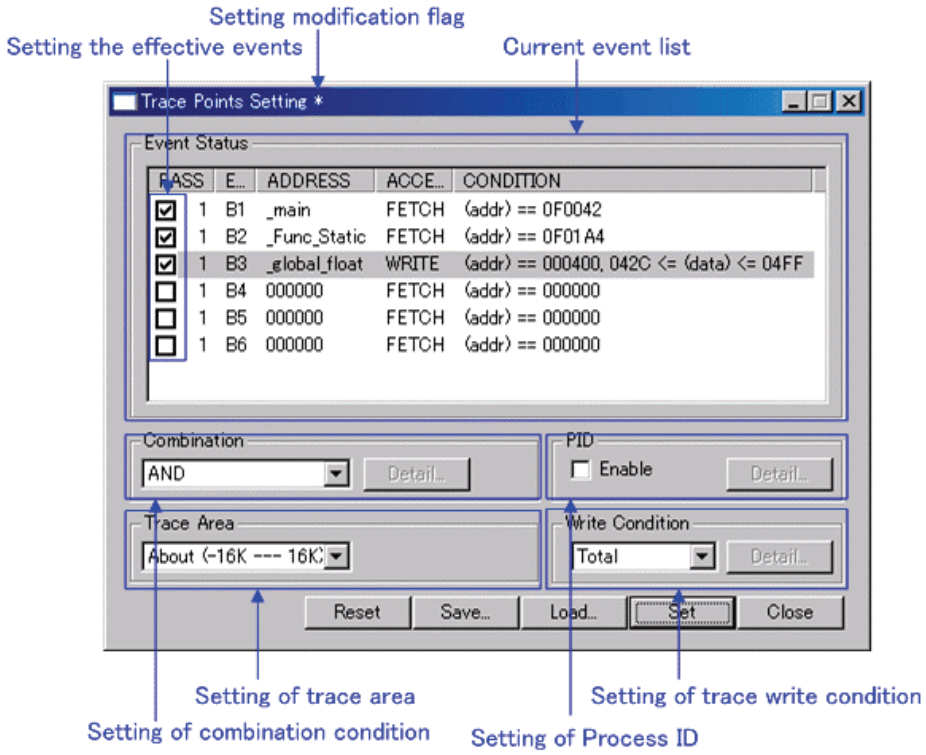


Example) Write data equal to or greater than 3Fh to address 400h



6.10 Trace Point Setting Window

The Trace Point Setting window is used to set trace points.
The debugger for 740 doesn't support this function.

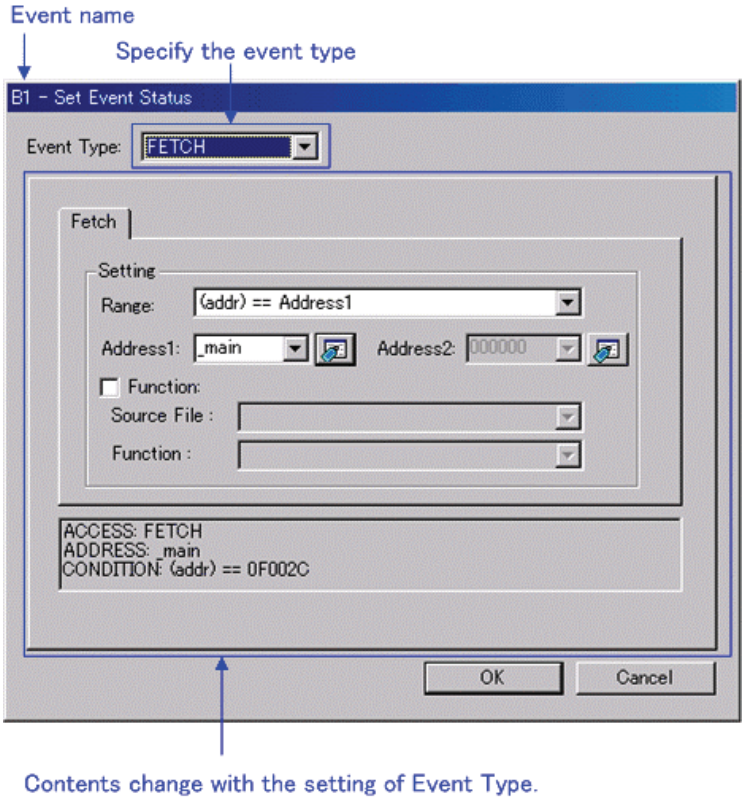


- The events listed below can be specified as trace events. If the contents of events are altered, they are marked by an asterisk (*) on the title bar. The asterisks (*) are not displayed after setting up the simulator.
  Fetch, Memory Access, Bit Access, Interrupt

- Events at up to six points can be used.
- These events can be combined in one of the following ways:
  - Trace when all of the valid events are established (AND condition)
  - Trace when all of the valid events are established at the same time (simultaneous AND condition)
  - Trace when one of the valid events is established (OR condition)
  - Trace upon entering a break state during state transition (State Transition condition)

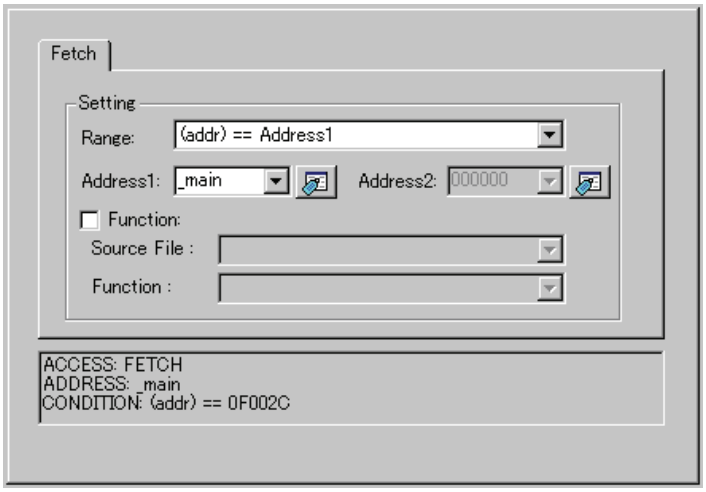
6.10.1 Specify the Trace Event

To set events, double-click to select the event you want to set from the event setting area of the Trace Point Setting Window. This opens the dialog box shown below.

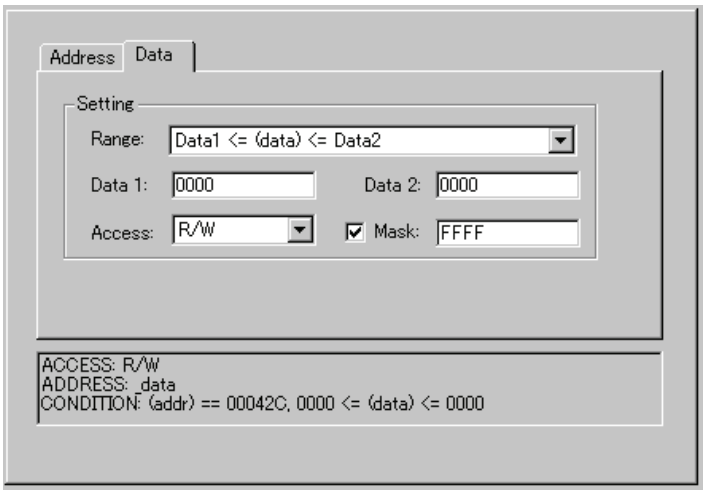


Following events can be set by specifying Event Type in this dialog box.

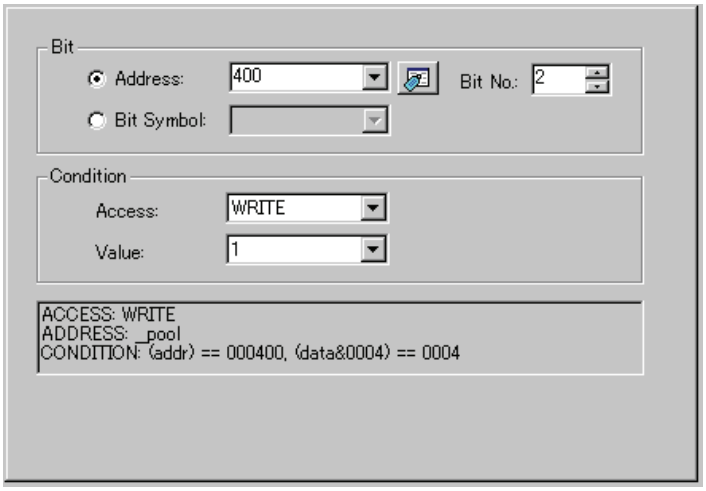
- When FETCH is selected
  Traces for the instruction fetch.



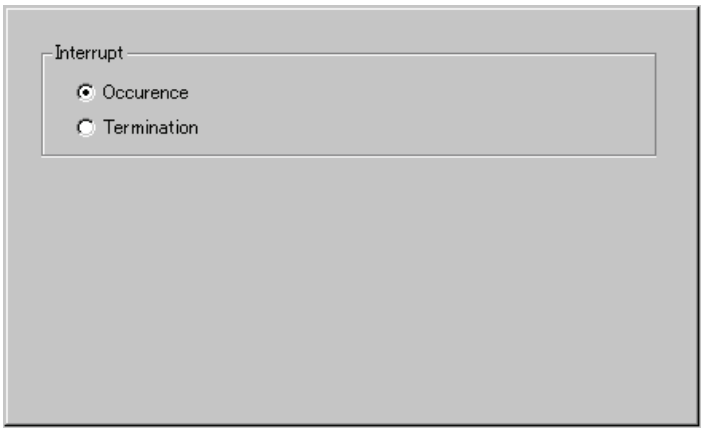
- When DATA ACCESS is selected
  Traces for the memory access.



- When BIT SYMBOL is selected
  Traces for the bit access.



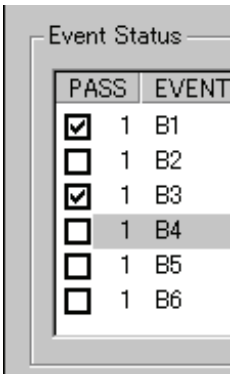
- When INTERRUPT is selected
  Traces for the interrupt occurrence or termination.



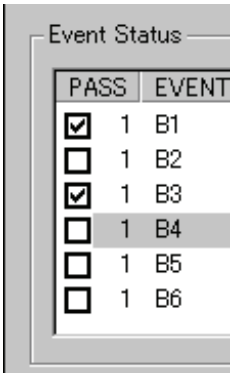
6.10.2 Specify the Combinatorial Condition

To specify a combinatorial condition, specify the desired condition from the combinatorial condition specification area.

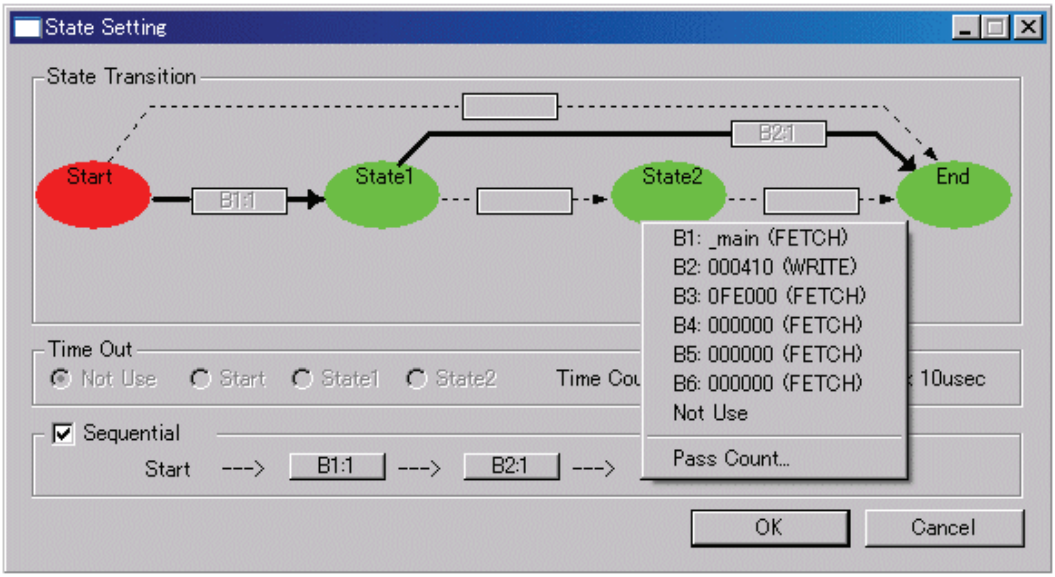
- When AND or OR is selected
  In the event specification area, the event used and a pass count for that event can be specified. To alter the pass count, while the event to alter is being selected, click the pass count value of that event.



- When AND (Same Time) is selected
  In the event specification area, the event used can be specified. No pass counts can be specified.

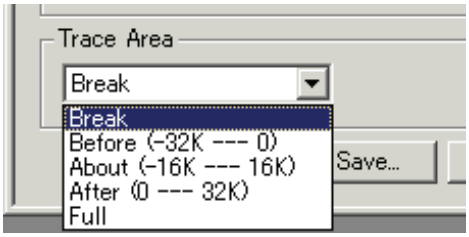


- When State Transition is selected
  Click the Details... button, and the dialog box shown below appears. Sequential specification can be used. If the content of any event is altered, it is marked with an asterisk (*) on the title bar. Once conditions are set in the simulator, asterisks are not displayed.



6.10.3 Specify the Trace Range

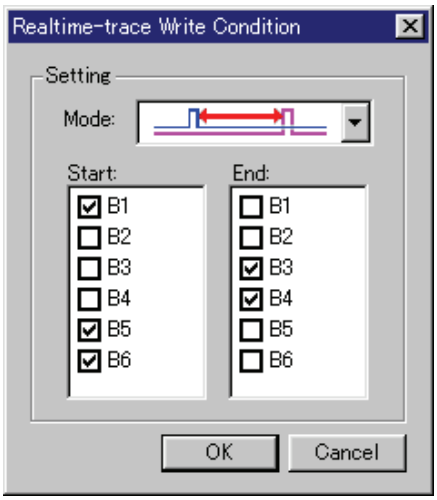
For the simulator debugger, as many cycles as specified on the Init dialog box's Trace tab can be recorded. (Descriptions below are written assuming 32K cycles.)



Break	Stores the 32K cycles (-32K to 0 cycles) to the point at which the target program stops.
Before	Stores the 32K cycles (-32K to 1 cycles) to the point at which the trace point is passed.
About	Stores the 32K cycles (-16K to 16K cycles) either side of the trace point.
After	Stores the 32K cycles (0 to 32K cycles) of trace data after the trace point.
Full	Stores the 32K cycles (-32K to 0 cycles) of trace data after the trace starts.

6.10.4 Specify the Trace Write Condition

Conditions for cycles to be written to trace memory can be specified.



Total	Writes all cycles.
Pick up	Writes only the cycles where specified condition holds true.
Exclude	Writes only the cycles where specified condition does not hold true.

Also, following three write modes are supported.

	Only cycles where specified event is established
	Cycles from where specified event is established to where specified event is not established
	Cycles from where start event is established to where end event is established

6.10.5 Command Button

The buttons on this window has the following meanings.

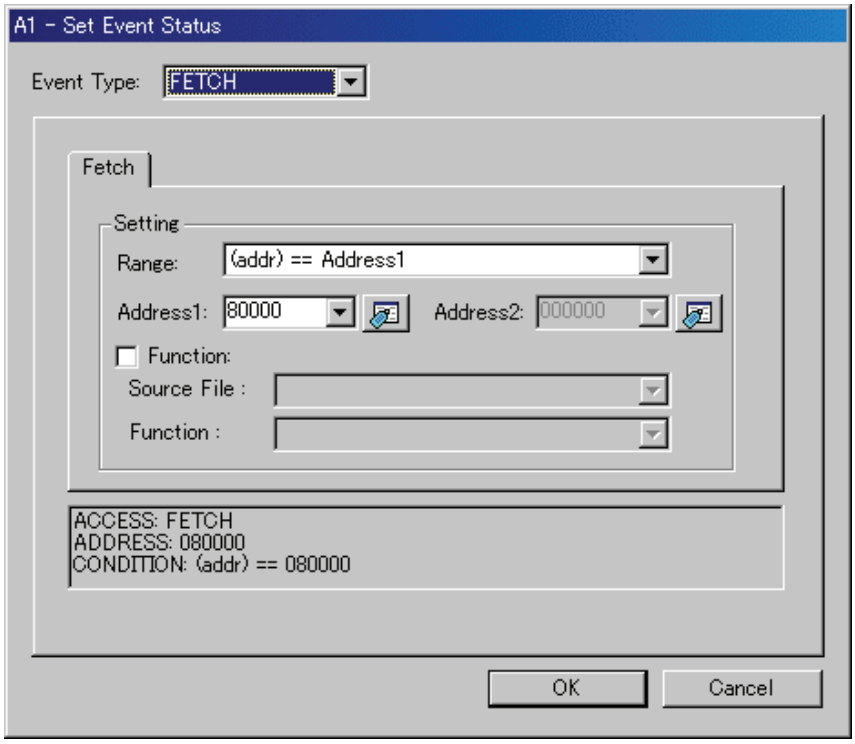
Button	Function
Reset	Discards the contents being displayed in the window and loads contents from the simulator in which they were set.
Save...	Saves the contents set in the window to a file.
Load...	Loads event information from a file in which it was saved.
Set	Sends the contents set in the window to the simulator.
Close	Closes the window.

6.10.6 Specify the Events (Instruction Fetch)

To specify an instruction fetch event, change the event select dialog box's Event Type to "FETCH". The event is established when instruction is fetched from the specified address or any address in the specified address range.

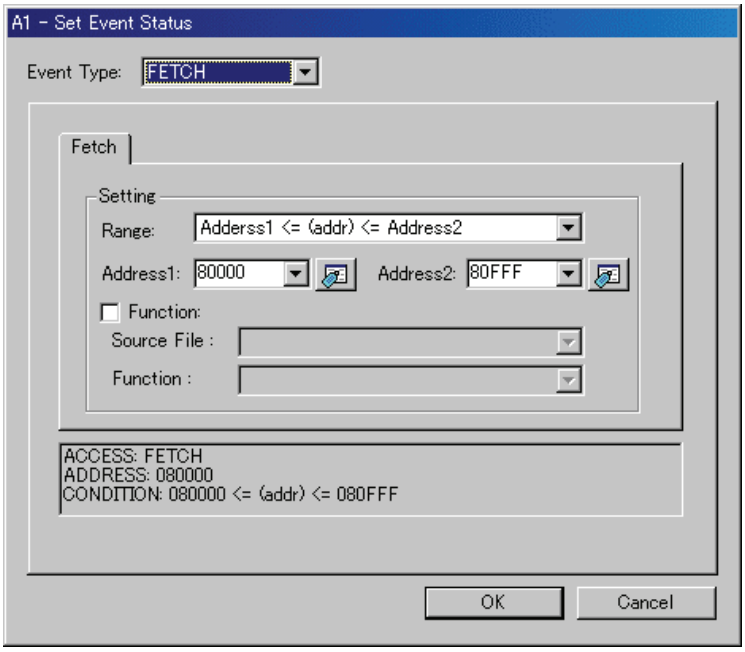
6.10.6.1.1. Instruction Fetch of Specified Address

Set as below.
Example) Instruction fetch at address 80000h



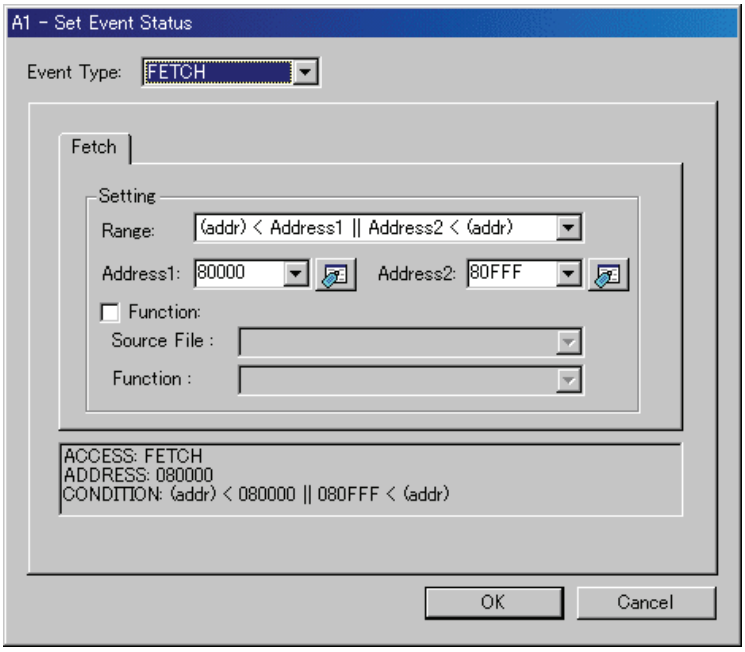
6.10.6.2 Instruction Fetch of Specified Address Area(In)

Set as below.
Example) Instruction fetch at address 80000h to 80FFFh



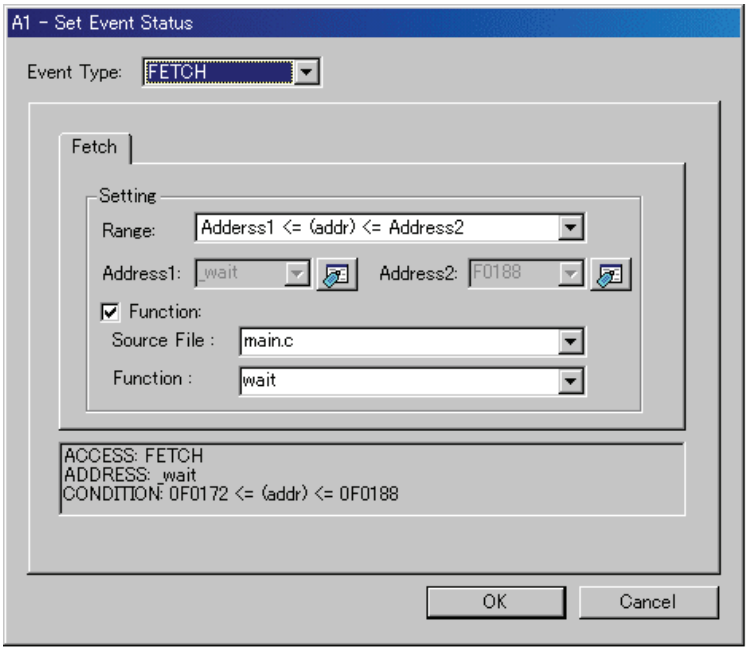
6.10.6.3 Instruction Fetch of Specified Address Area(Out)

Set as below.
Example) Instruction fetch at any address other than the range 80000h to 80FFFh

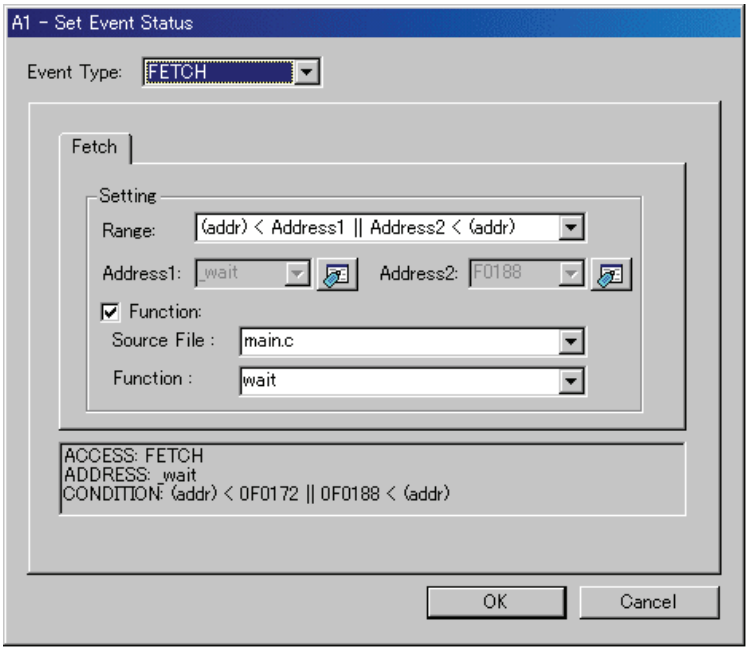


6.10.6.4 Entering/exiting to specified function

Set as below.
Example) Entering a break to function name "wait"



Example) Exiting from function name "wait"



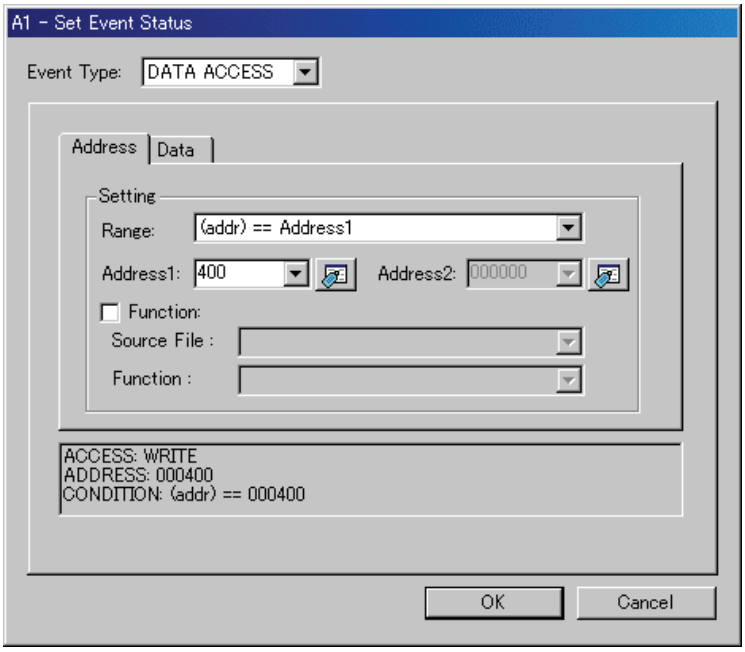
6.10.7 Specify the Events (Memory Access)

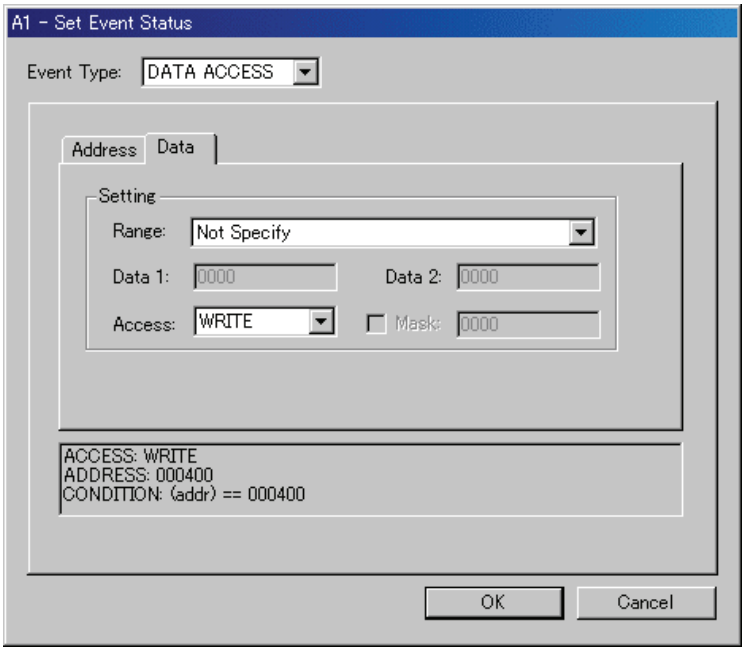
To specify a memory access event, change the event select dialog box's Event Type to "DATA ACCESS". The event is established when memory is accessed at the specified address or under conditions set for the specified address range.

6.10.7.1 Memory Access(The debugger for R32C)

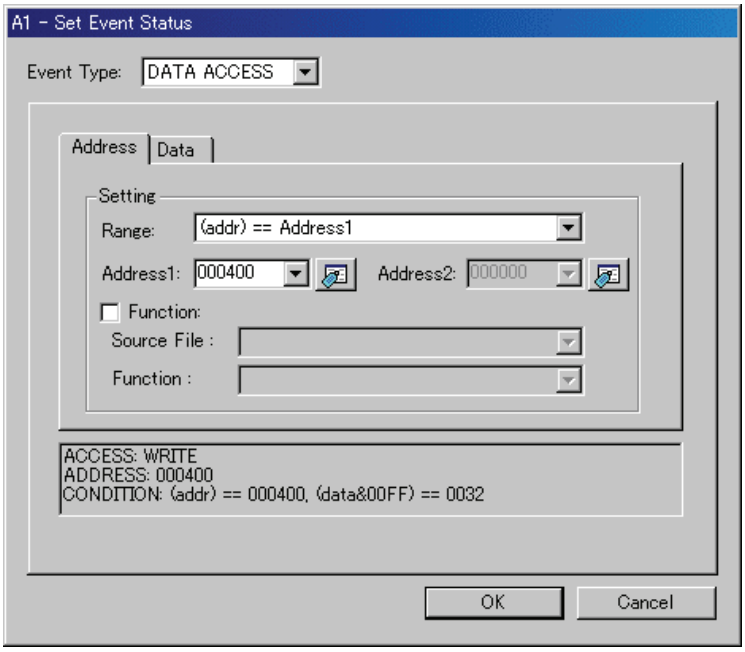
6.10.7.1.1. Writing/Reading a Specified Address

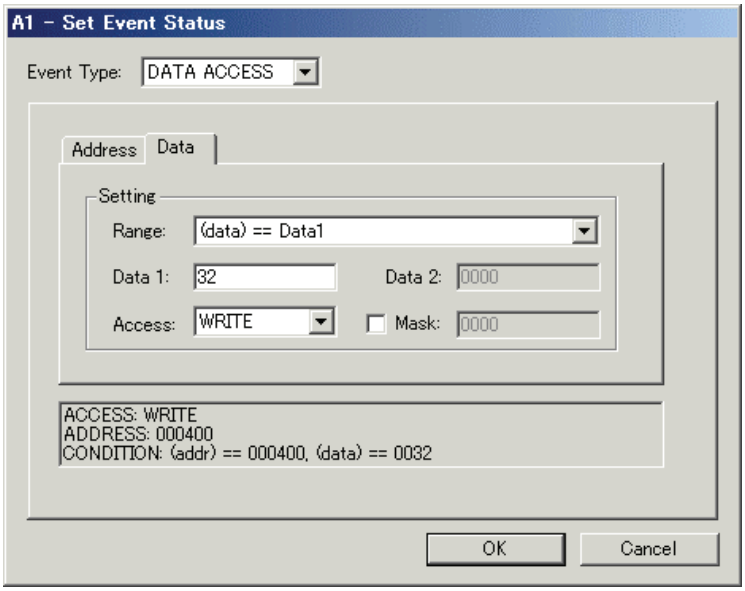
Set as below.
Example) Writing to even address 400h



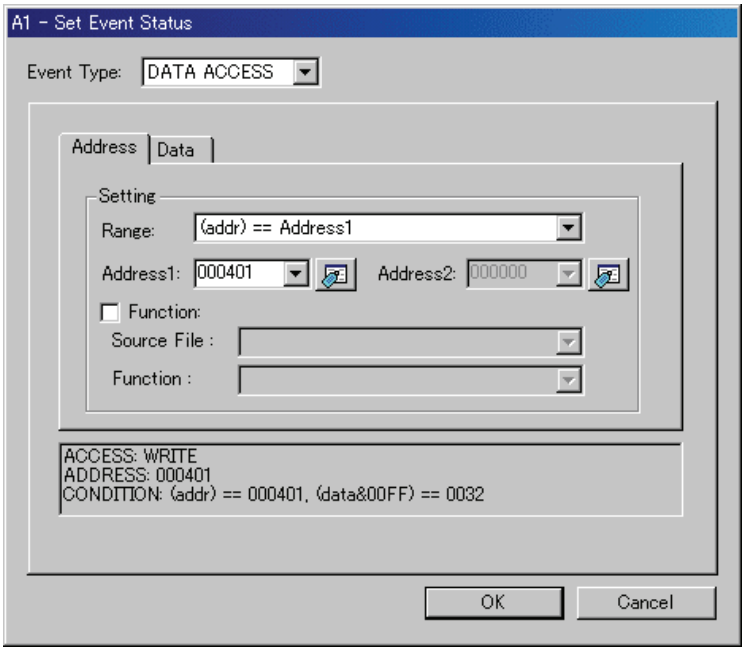


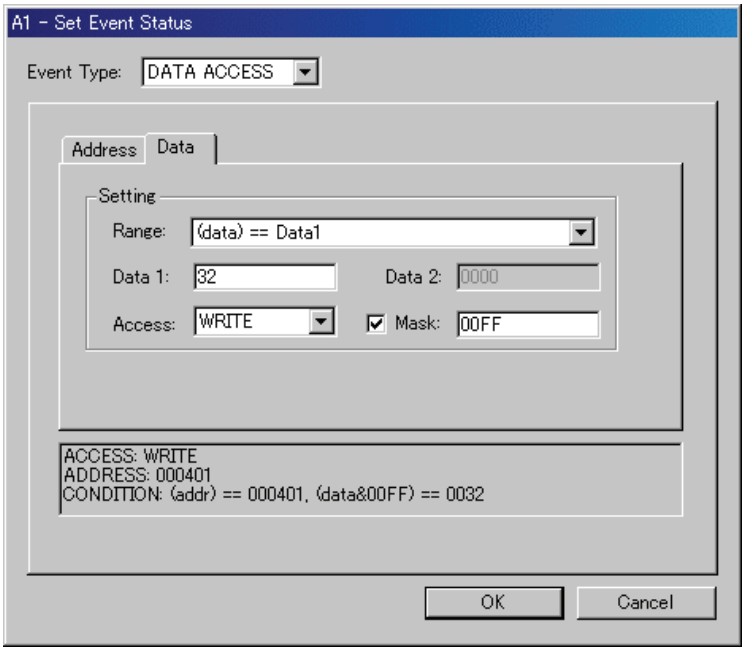
Example) Writing byte length data 32h to even address 400h



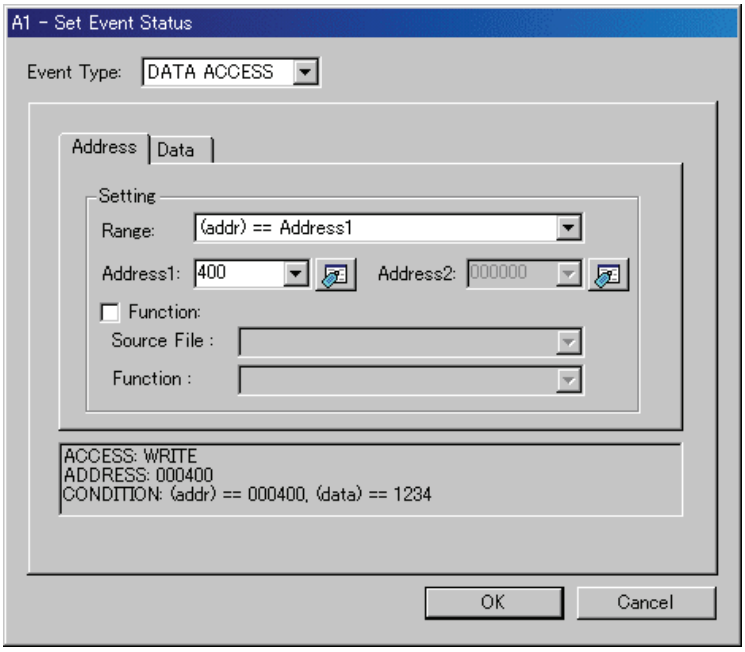


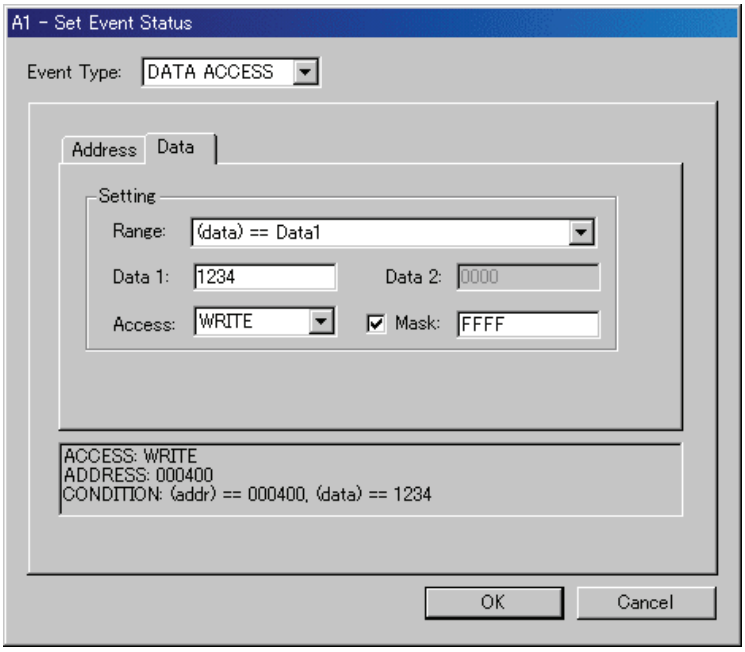
Example) Writing byte length data 32h to odd address 401h



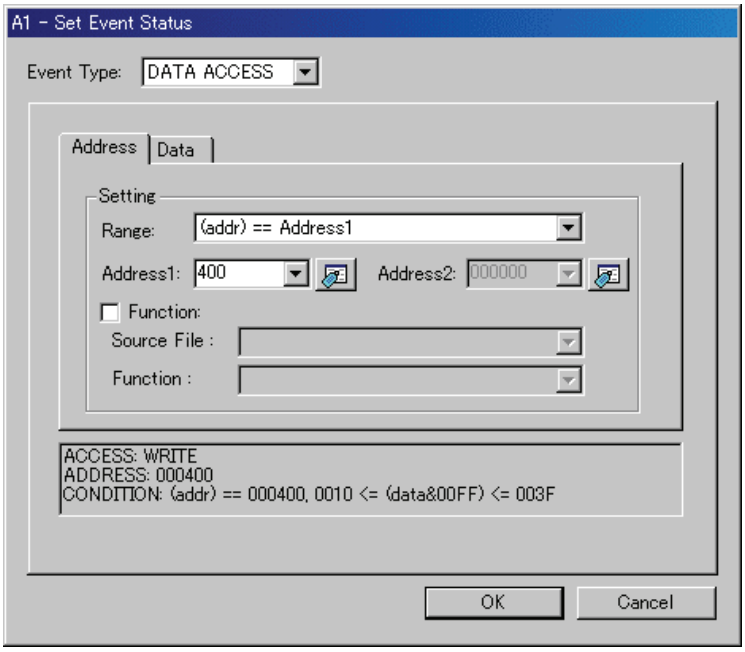


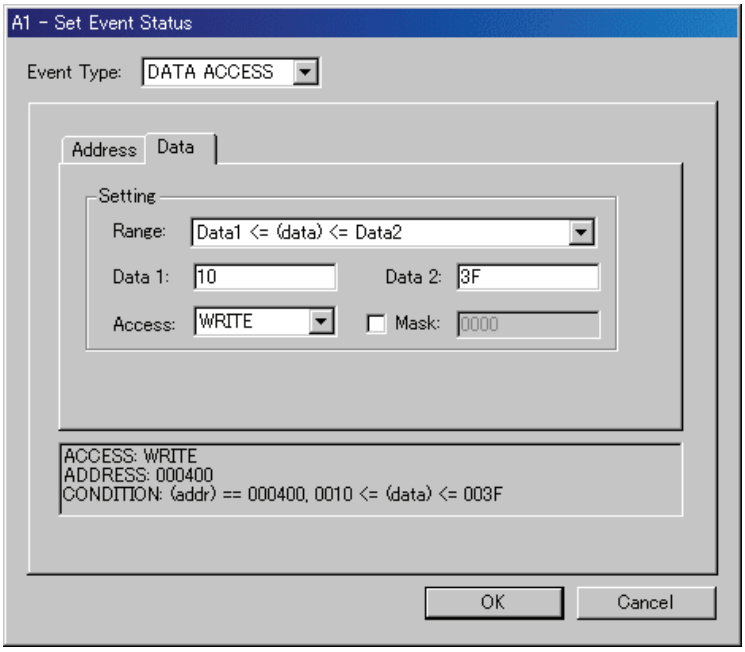
Example) Writing word length data 1234h to even address 400h





Example) Writing data 10h - 3Fh to even address 400h

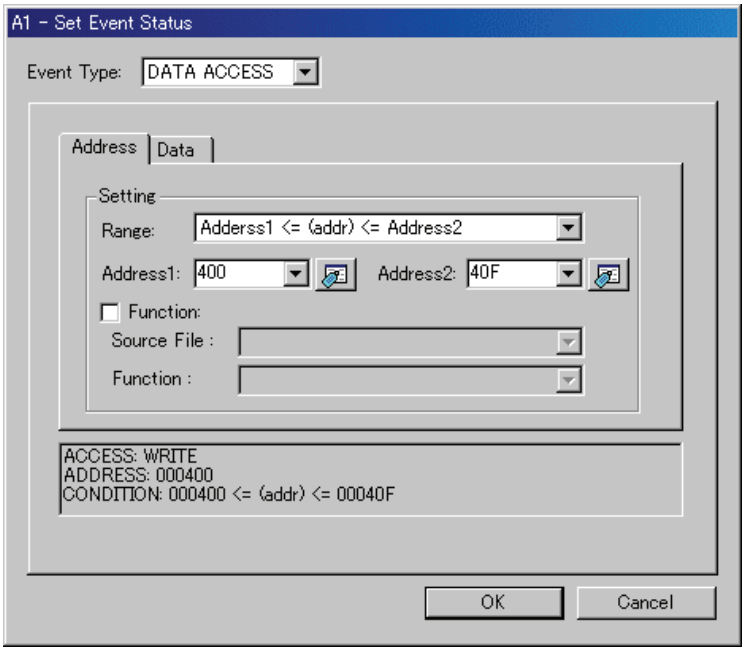


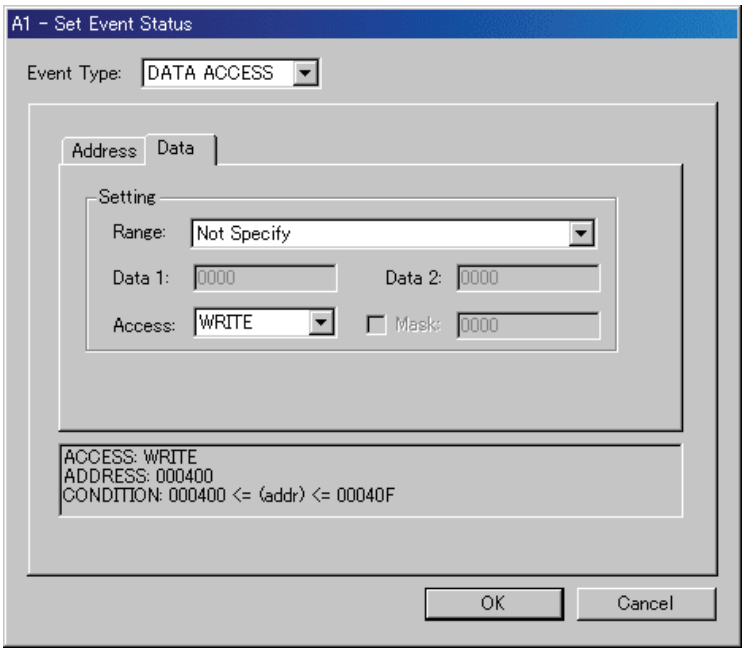


6.10.7.1.2. Reading/writing data to the specified address range

Set as below.

Example) Writing data to addresses ranging from 400h to 40Fh

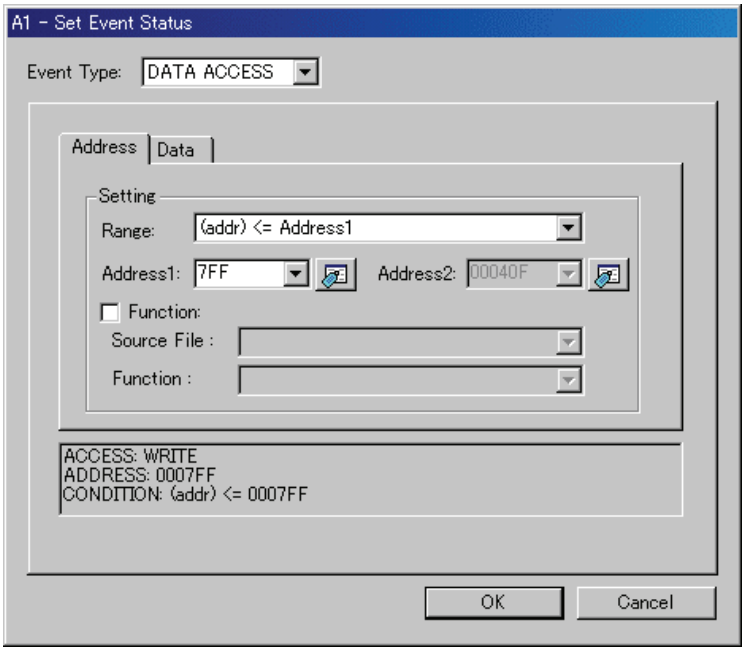


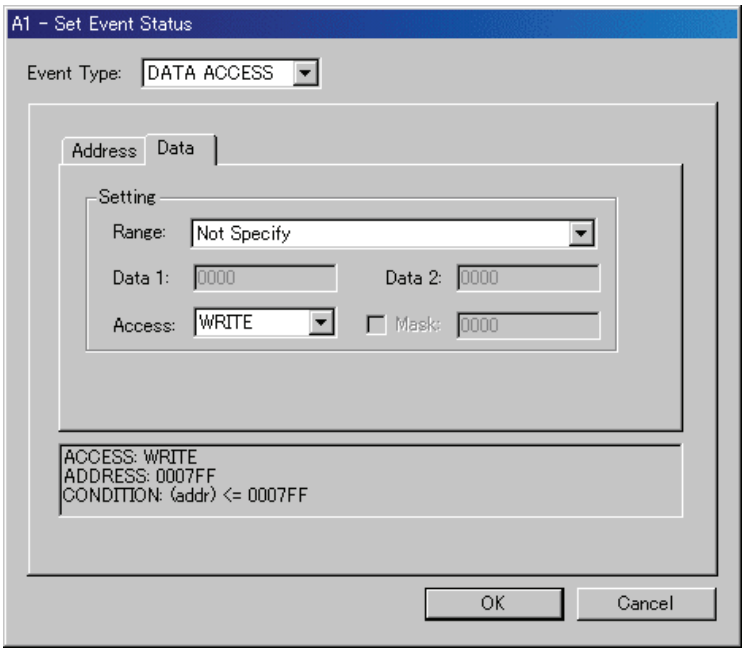


6.10.7.1.3. Reading/writing data to addresses outside the specified range

Set as below.

Example) Writing data to addresses below 7FFh

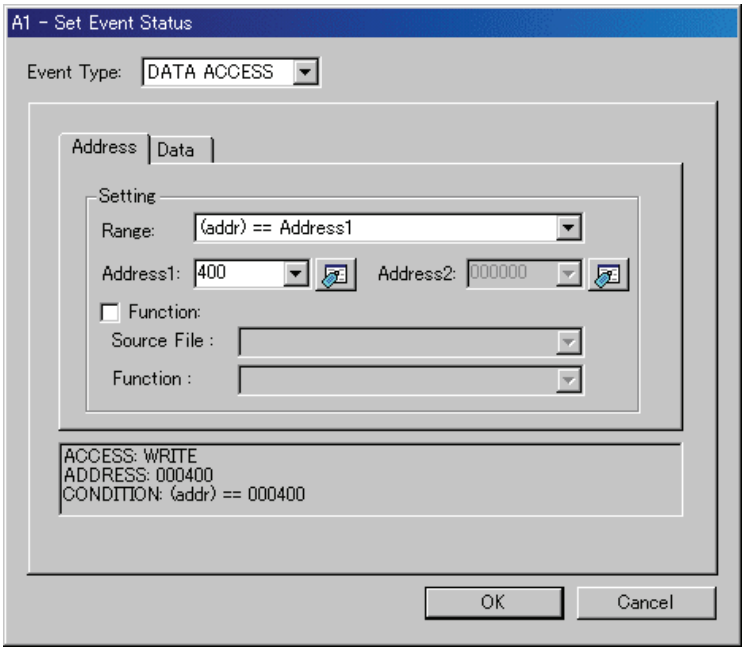


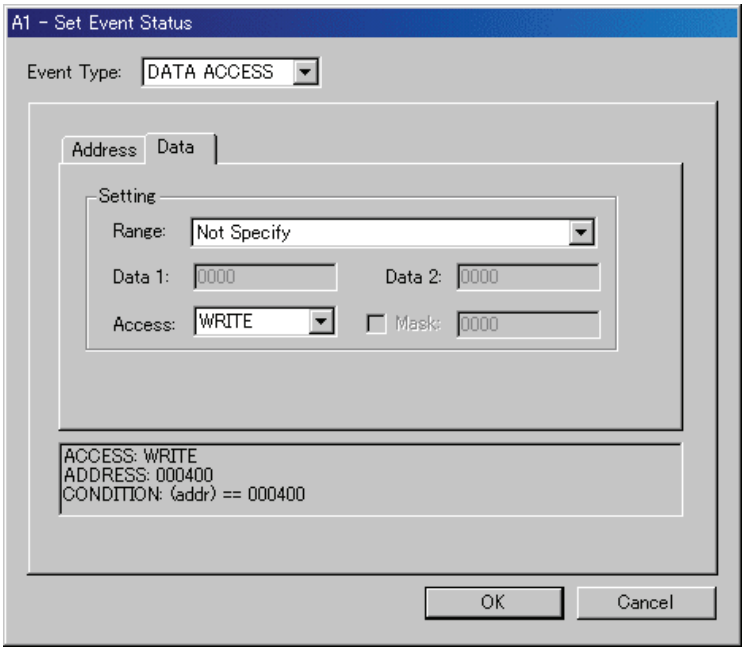


6.10.7.2 Memory Access(The debugger for M32C)

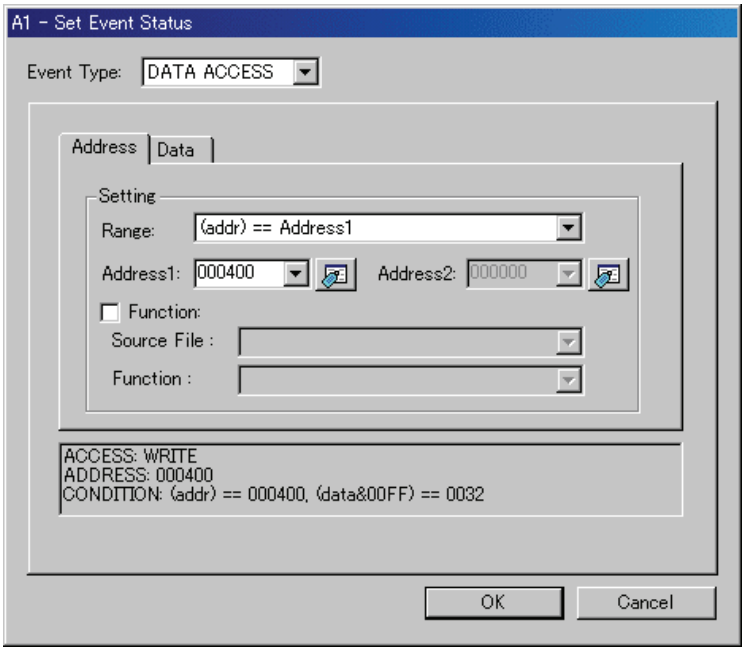
6.10.7.2.1. Writing/Reading a Specified Address

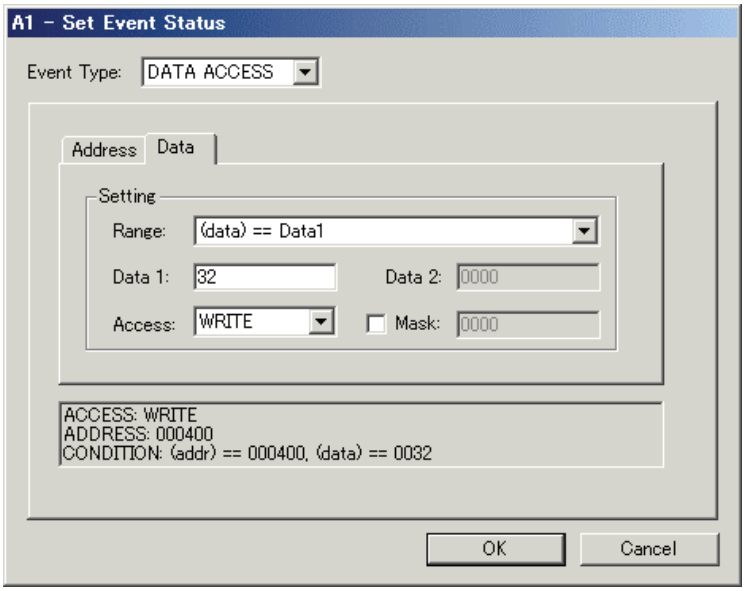
Set as below.
Example) Writing to even address 400h



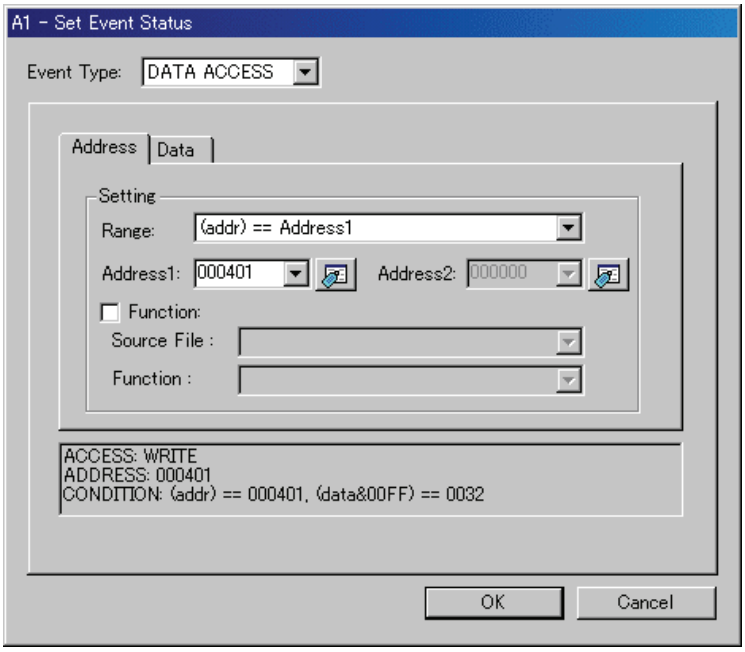


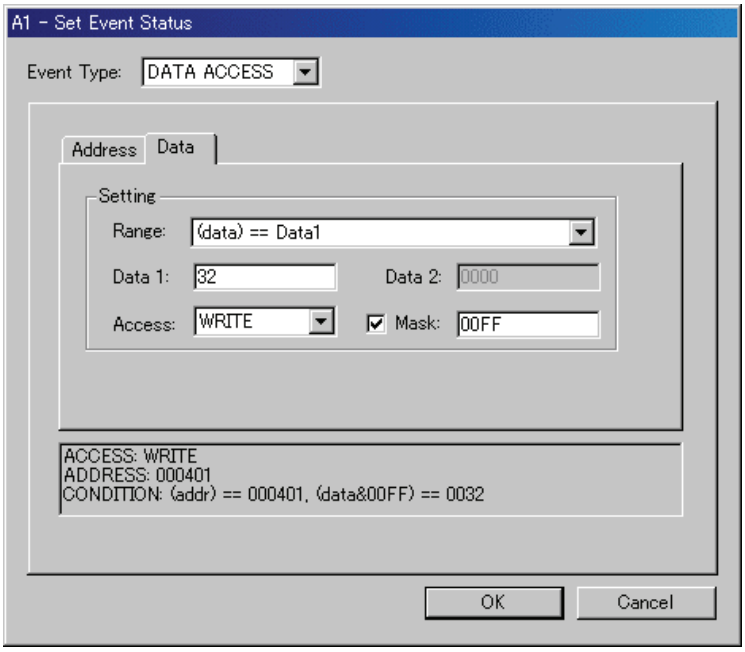
Example) Writing byte length data 32h to even address 400h



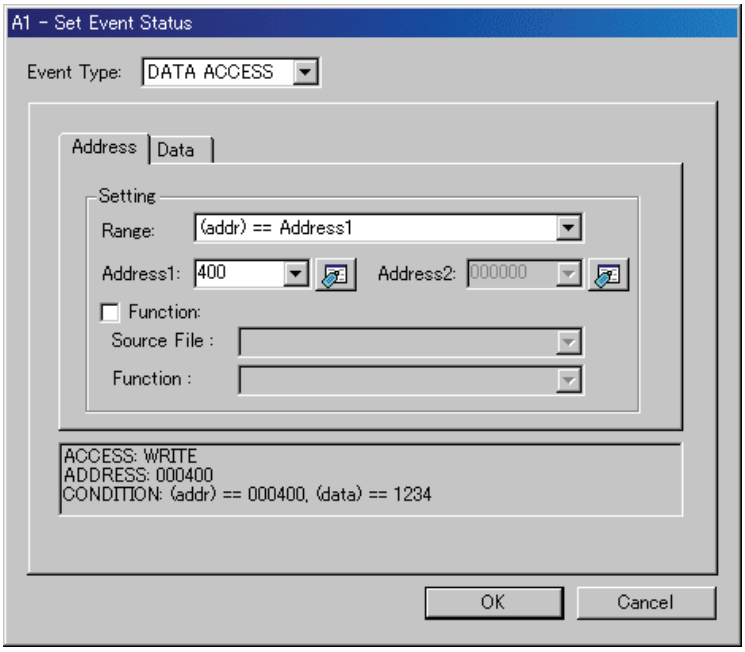


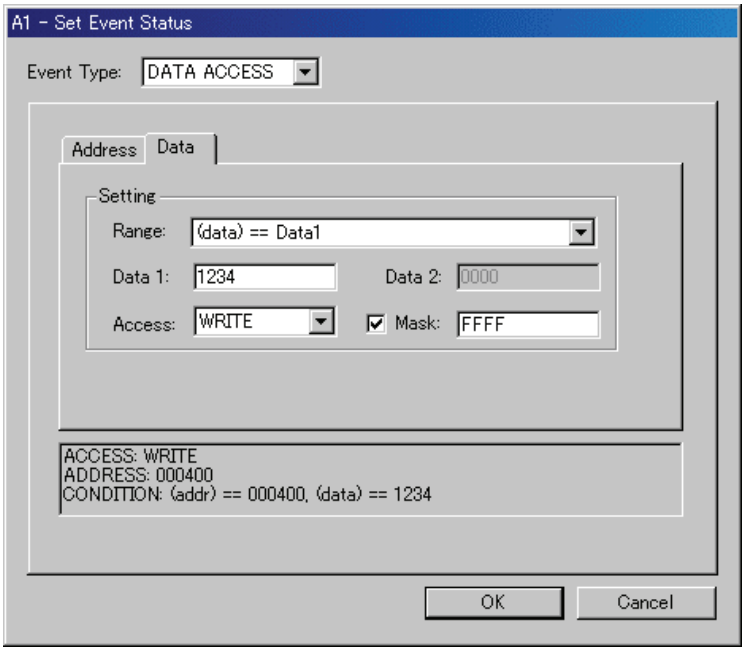
Example) Writing byte length data 32h to odd address 401h



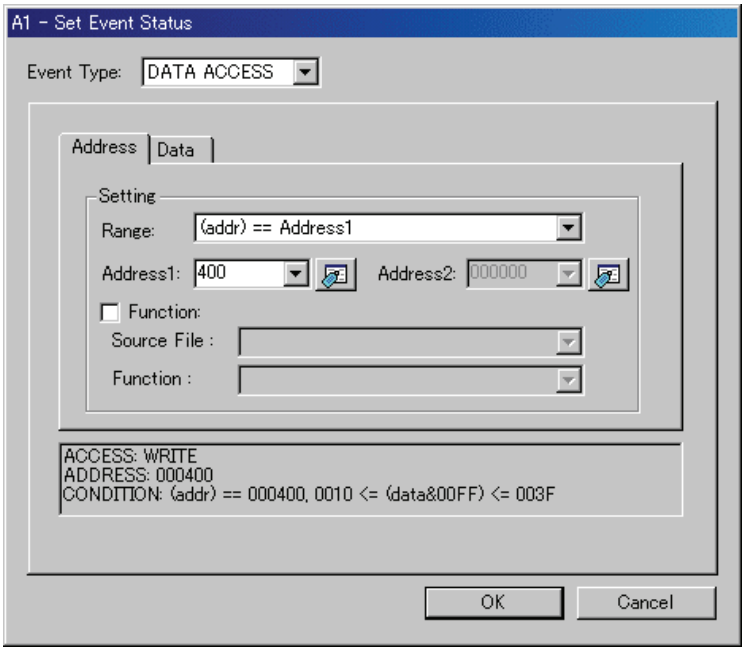


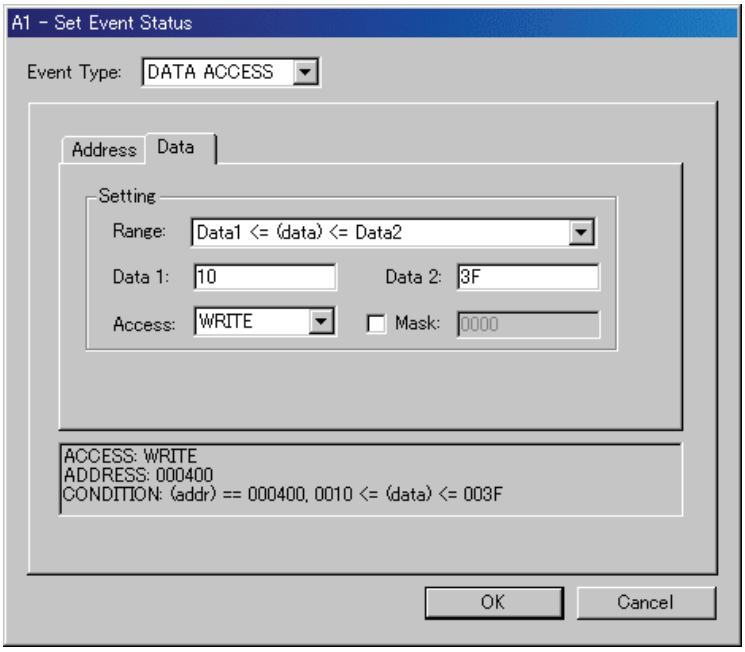
Example) Writing word length data 1234h to even address 400h





Example) Writing data 10h - 3Fh to even address 400h

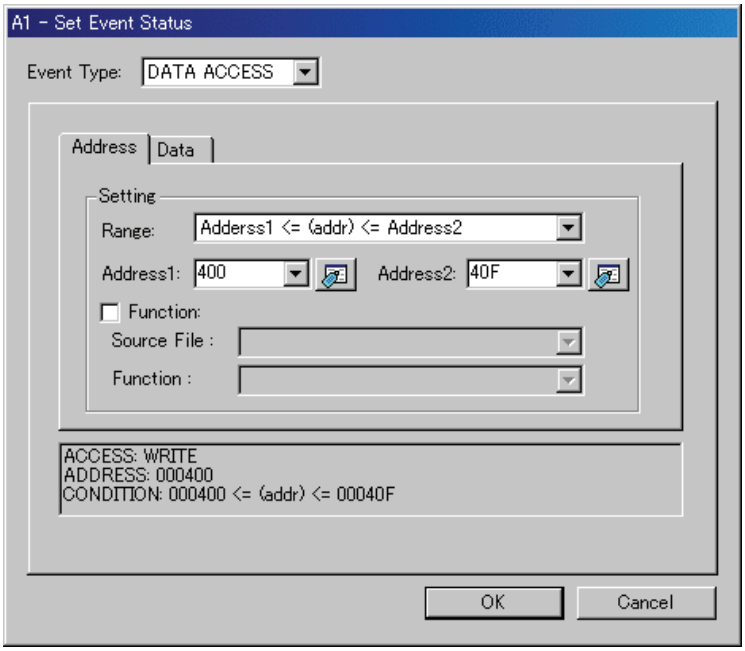


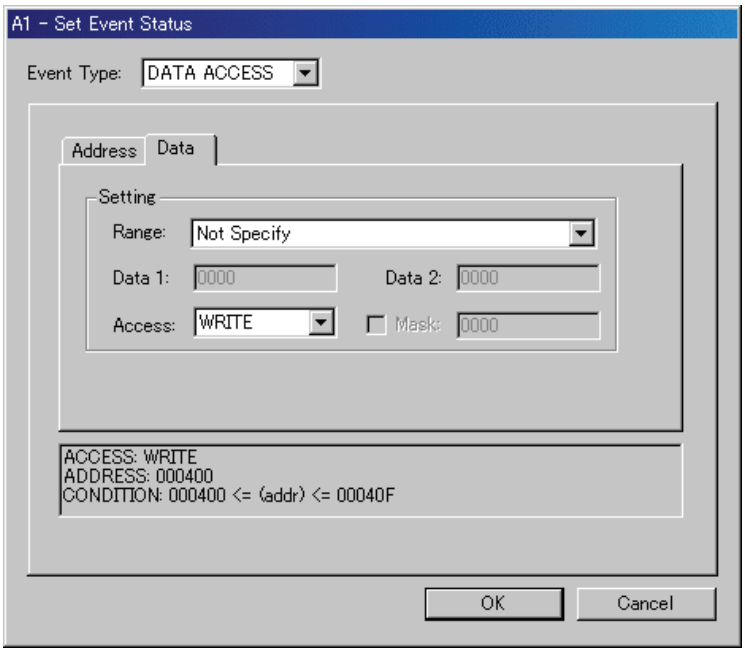


6.10.7.2.2. Reading/writing data to the specified address range

Set as below.

Example) Writing data to addresses ranging from 400h to 40Fh

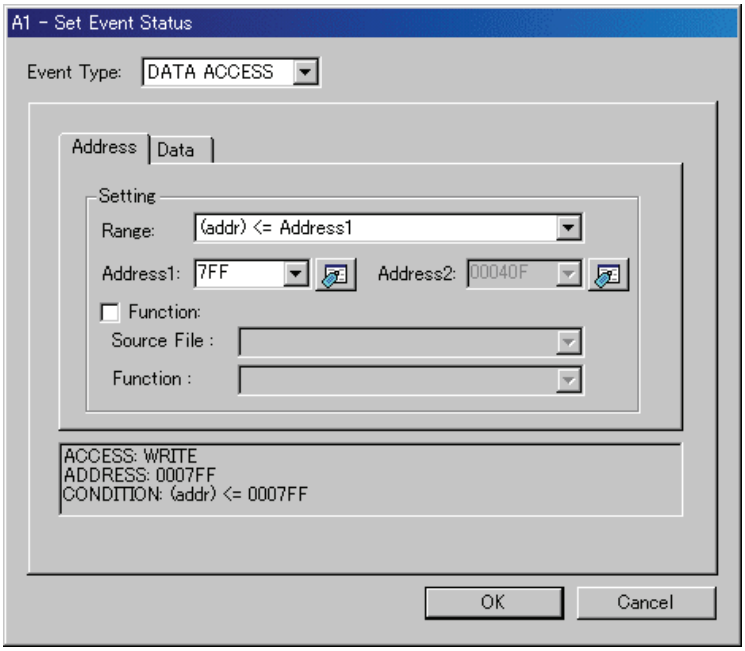


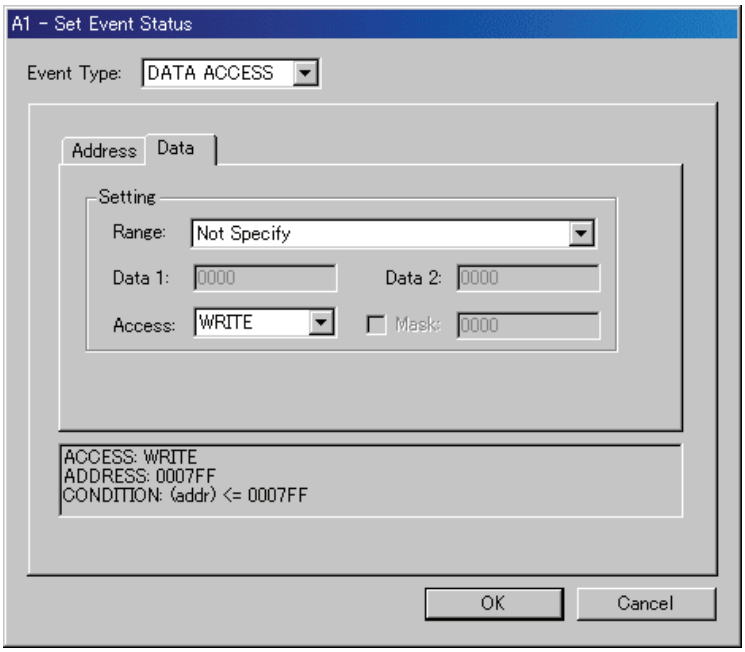


6.10.7.2.3. Reading/writing data to addresses outside the specified range

Set as below.

Example) Writing data to addresses below 7FFh

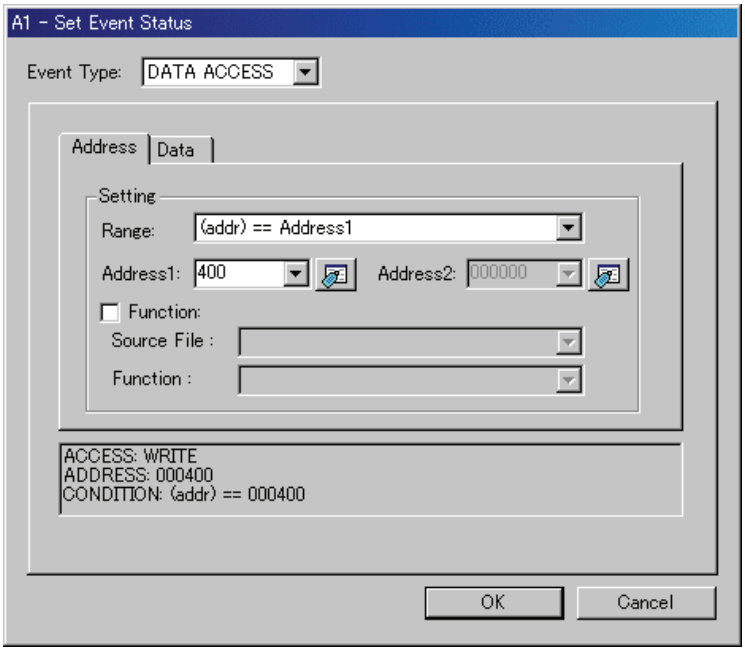


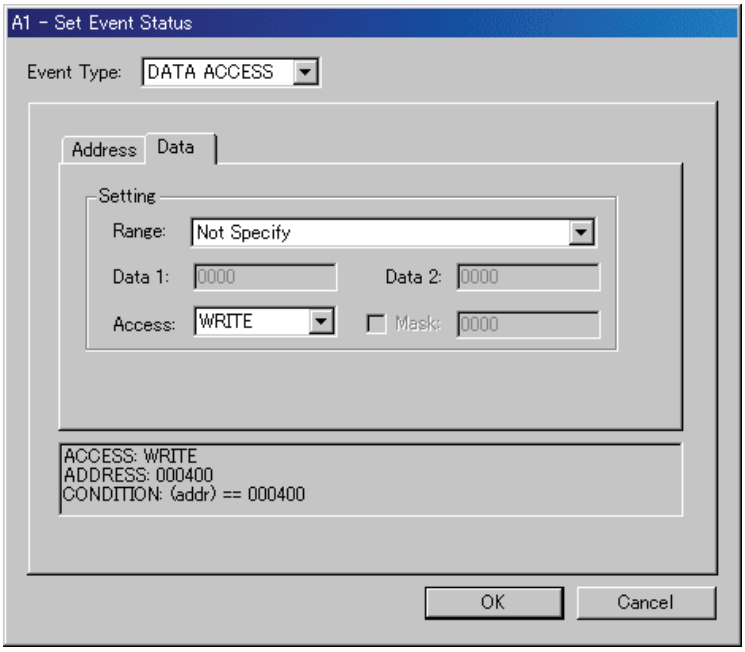


6.10.7.3 Memory Access(The debugger for M16C/R8C)

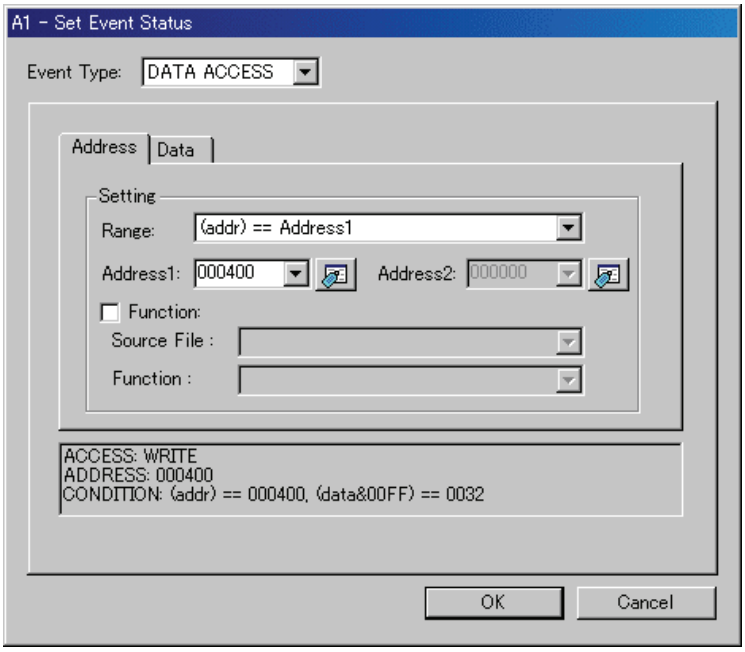
6.10.7.3.1. Writing/Reading a Specified Address

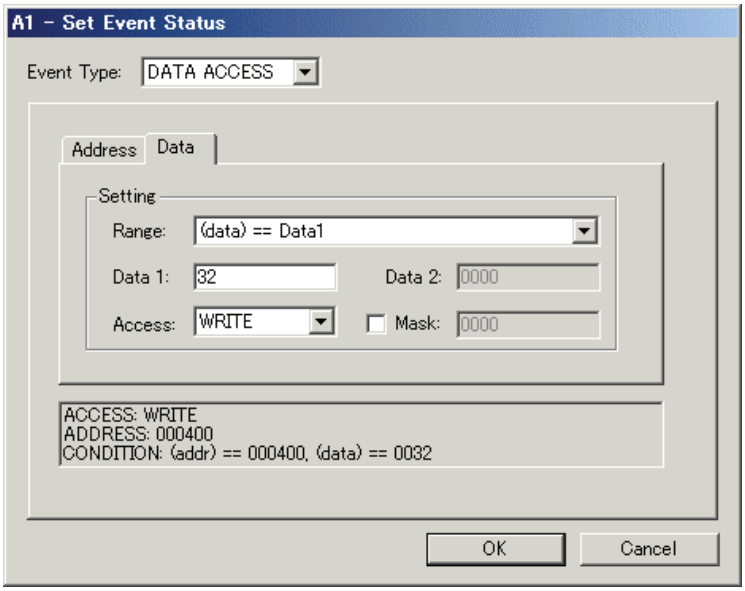
Set as below.
Example) Writing to even address 400h



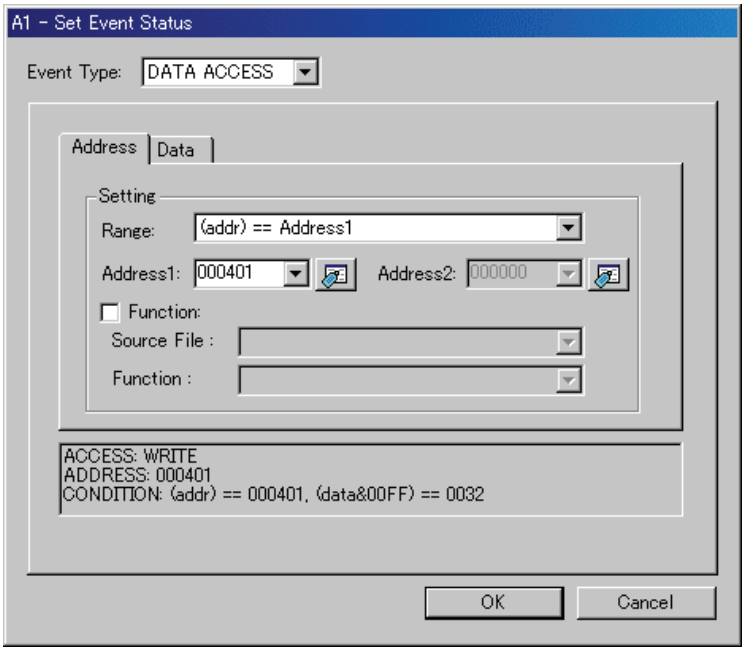


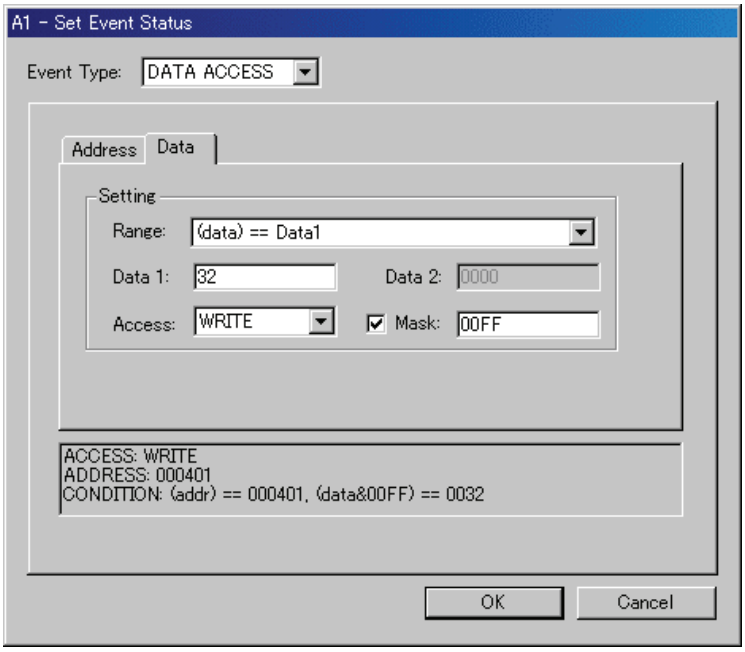
Example) Writing byte length data 32h to even address 400h



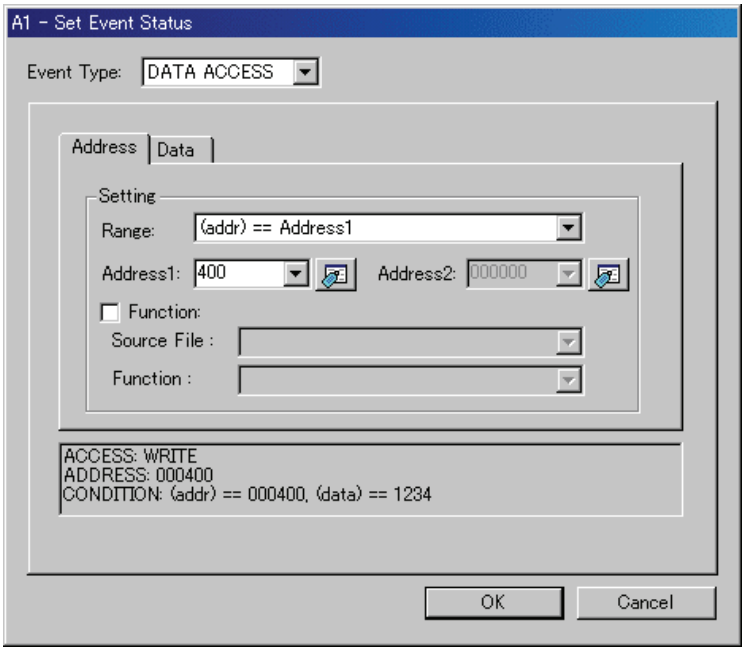


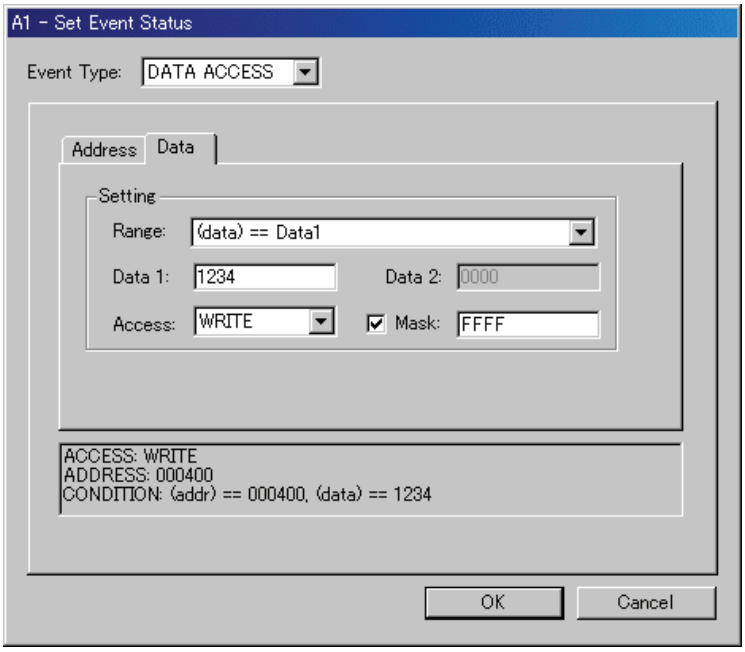
Example) Writing byte length data 32h to odd address 401h



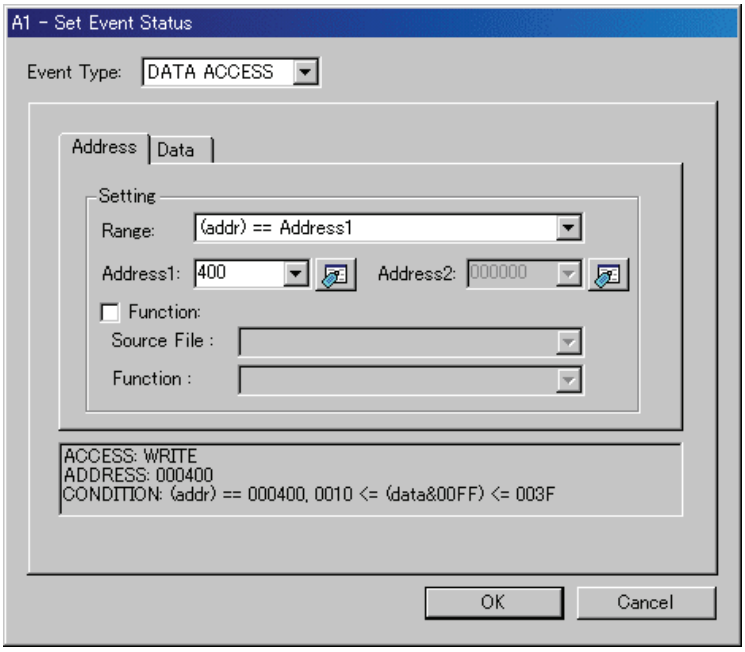


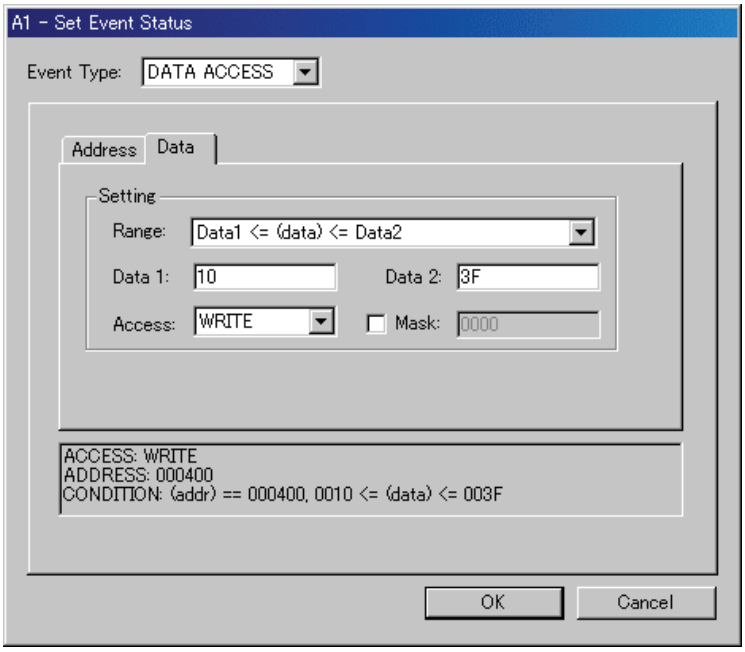
Example) Writing word length data 1234h to even address 400h





Example) Writing data 10h - 3Fh to even address 400h

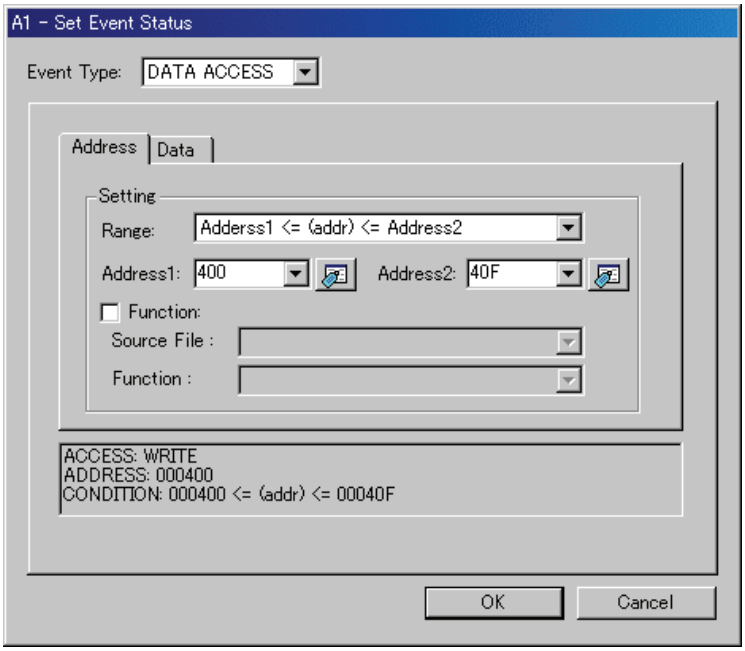


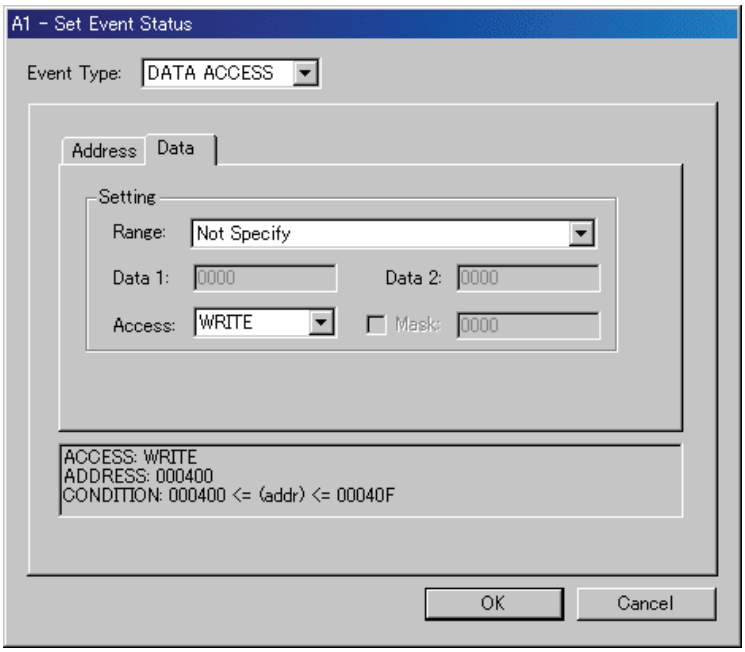


6.10.7.3.2. Reading/writing data to the specified address range

Set as below.

Example) Writing data to addresses ranging from 400h to 40Fh

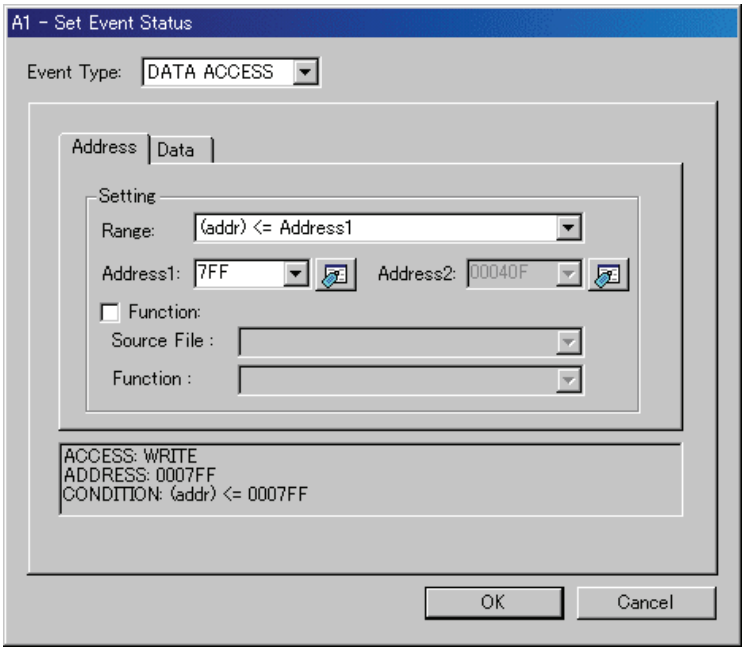


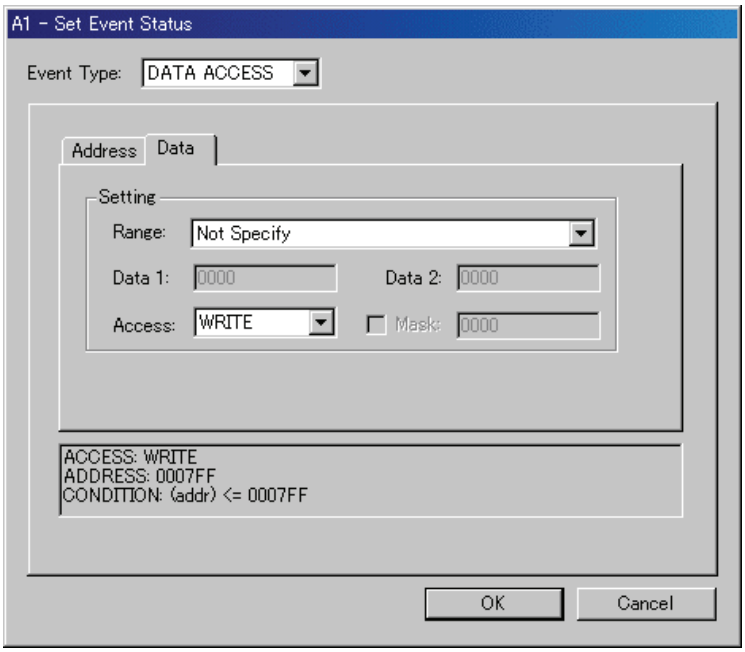


6.10.7.3.3. Reading/writing data to addresses outside the specified range

Set as below.

Example) Writing data to addresses below 7FFh



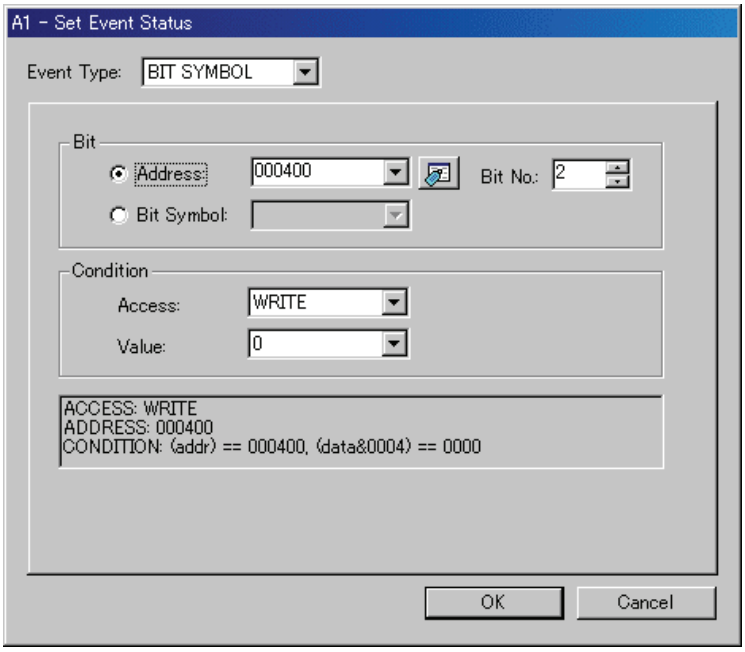


6.10.8 Specify the Events (Bit Access)

To specify a bit access event, change the event select dialog box's Event Type to "BIT SYMBOL". The event is established when the specified bit at the specified address or specified bit symbol is accessed under specified conditions.

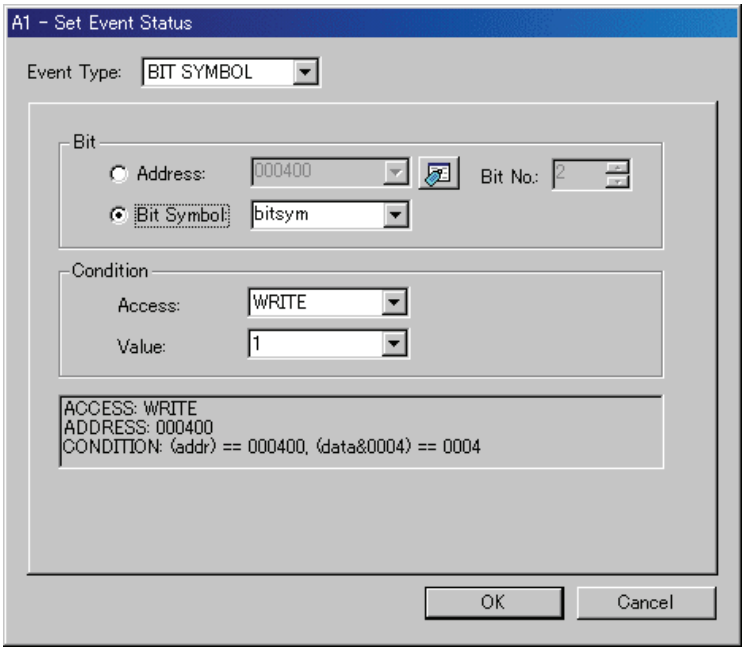
6.10.8.1 Writing/Reading a Specified Bit

Set as below.
Example) Writing "0" to bit 2 at address 400h



6.10.8.2 Writing/Reading a Specified Bit Symbol

Set as below.
Example) Writing "1" to bit symbol "bitsym"

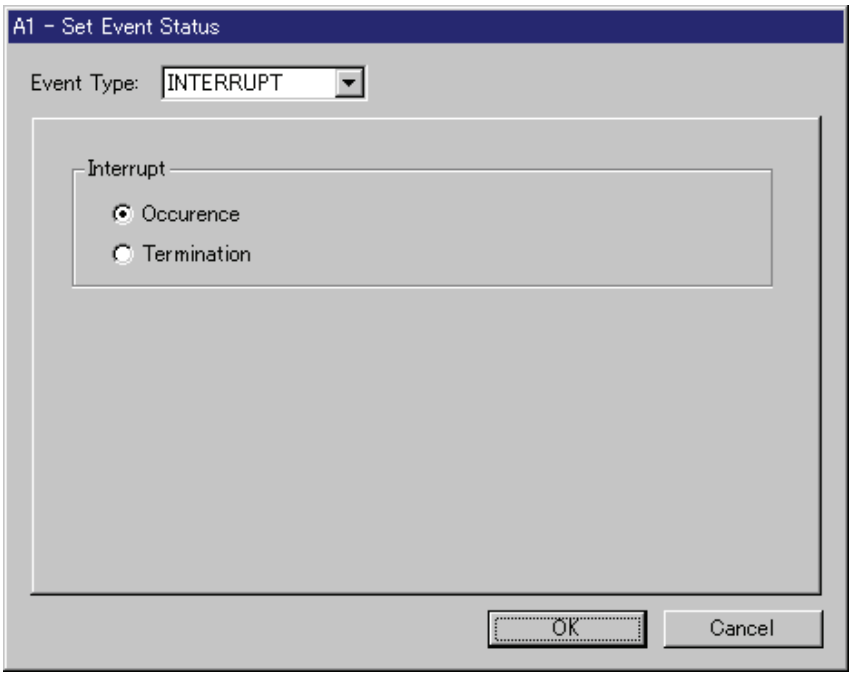


6.10.9 Specify the Events (Interrupt)

To specify an interrupt event, change Event Type in the event select dialog box to "INTERRUPT". When an interrupt is generated or finished, the event is established.

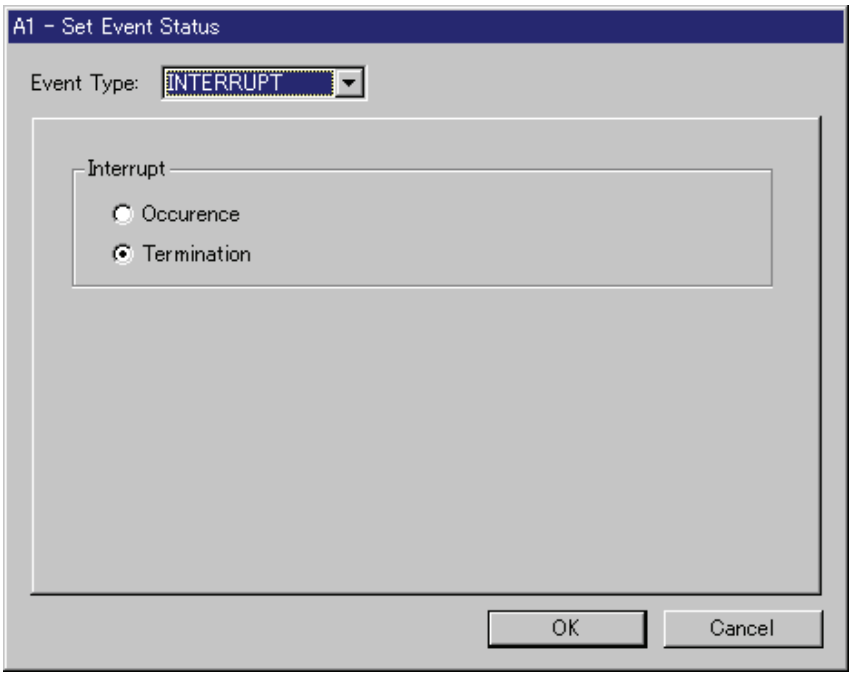
6.10.9.1 Interrupt Occurrence

Set as below.



6.10.9.2 Interrupt Termination

Set as below.



6.10.10 Specify the Event Combination Condition

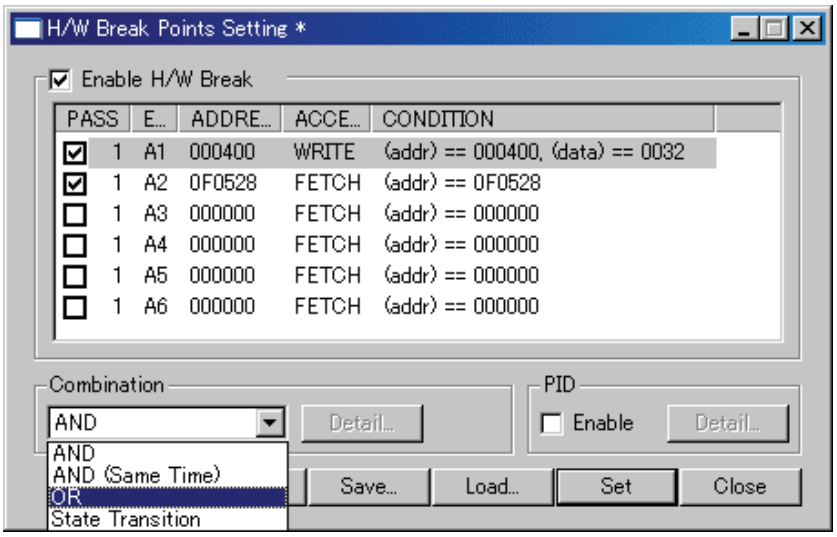
Use the Combination group of the event setting windows to specify the combinatorial conditions of events.
The combination of two or more events can be used.
One of the following combinatorial conditions can be selected.

AND	All of the specified events are established
AND(Same Time)	The specified events are established at the same time
OR	One of the specified events is established
STATE TRANSITION	Established upon entering a break state in the state transition diagram

Pass counts (number of times passed) can be specified for each event (1-255). If the specified combinatorial condition is AND (Same Time), no pass counts can be set (fixed to 1).

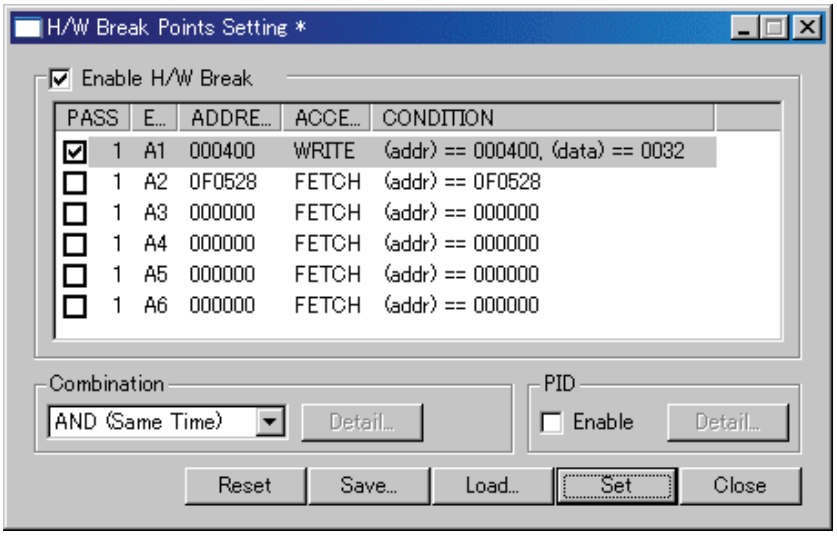
6.10.10.1 Select AND, OR

Change the Combination group to "AND" to specify AND for the combinatorial condition, or "OR" to specify OR for the combinatorial condition. Next, check (turn on) an event in the event specification area that you want to use, and specify a pass count for that event. To alter the pass count, while the event to alter is being selected, click the pass count value of that event.



6.10.10.2 Select AND(Same Time)

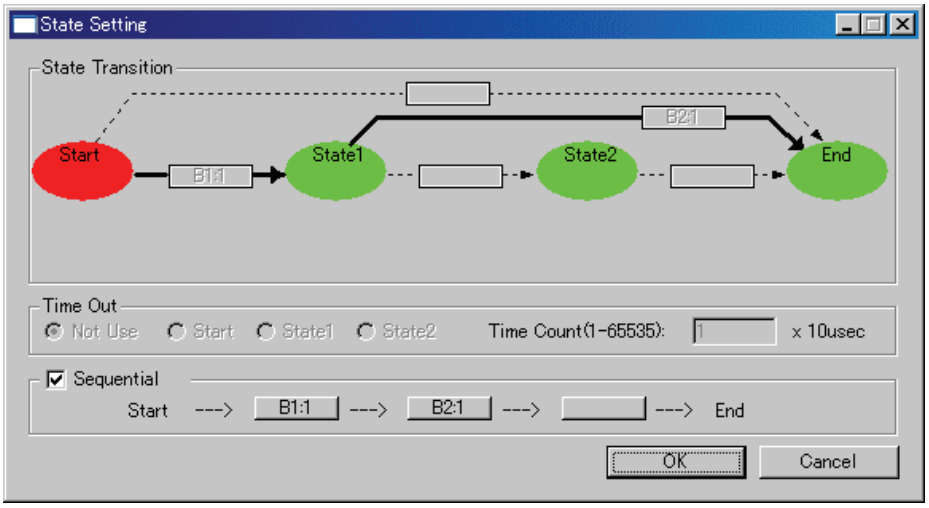
Change the Combination group to "AND (Same Time)". Next, check (turn on) an event in the event specification area that you want to use. No pass counts can be specified (fixed to 1).



6.10.10.3 Select State Transition

Change the Combination group to "State Transition". The Detail... button included in the Combination group becomes useful, so click that button. This opens the State Setting window. In this window, State Transition can be specified using sequentially.
Use the buttons included in the Sequential group. Pass counts can be specified from the popup menu that appears when selecting an event. The contents set here are reflected in the state transition diagram.

Example: Events B1, and B2 that occur successively in that order are established



6.10.11 Specify the write condition

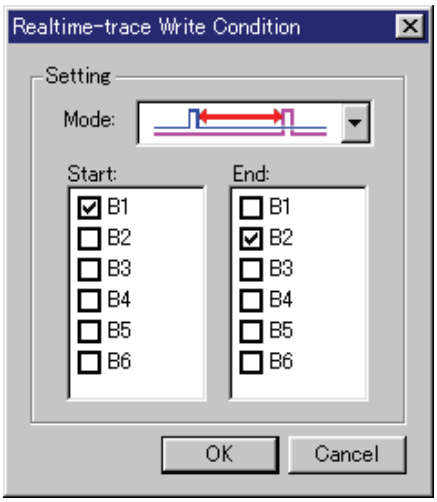
Trace data write conditions can be specified.
You can specify the following write conditions:

1. Write conditions unlimited (default)
2. Cycles from the start event established to the end event established
3. Only cycles where the start event is established
4. Cycles from the start event established to the start event unestablished
5. Other than cycles from the start event established to the end event established
6. Other than cycles where the start event is established
7. Other than cycles from the start event established to the start event unestablished

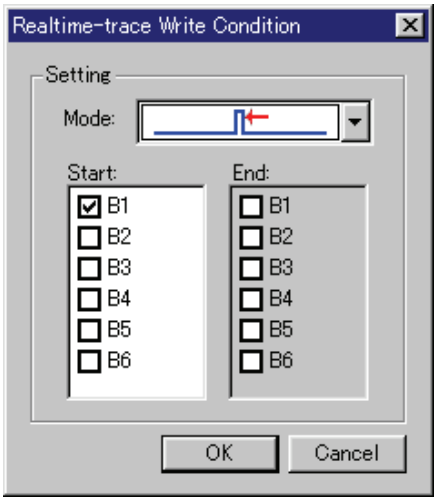
To specify condition 1, choose "Total" from the list box of the window's "Write Condition" item.

To specify conditions 2 to 4, choose "Pick Up" and click the "Detail..." button to open the "Realtime-trace Write Condition" dialog box.

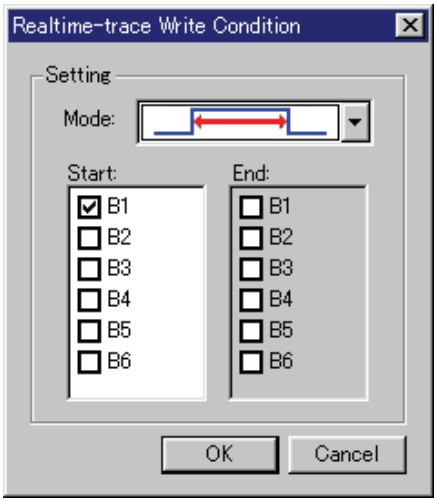
- For condition 2, choose the Mode shown below and set the Start and End events.



- For condition 3, choose the Mode shown below and set the Start event.

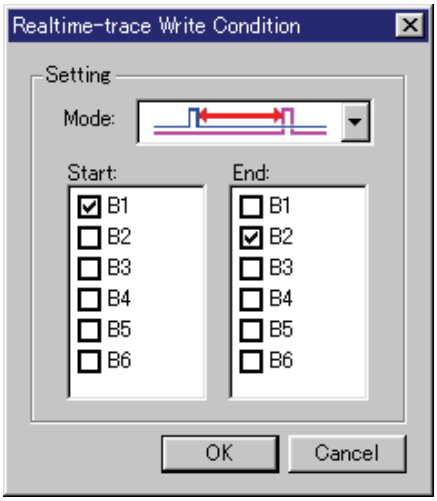


- For condition 4, choose the Mode shown below and set the Start event.

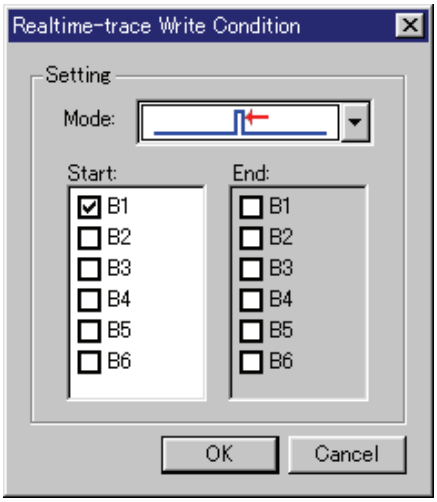


Similarly, when specifying conditions 5 to 7, choose "Exclude" and click the "Detail..." button to open the Realtime-trace Write Condition dialog box.

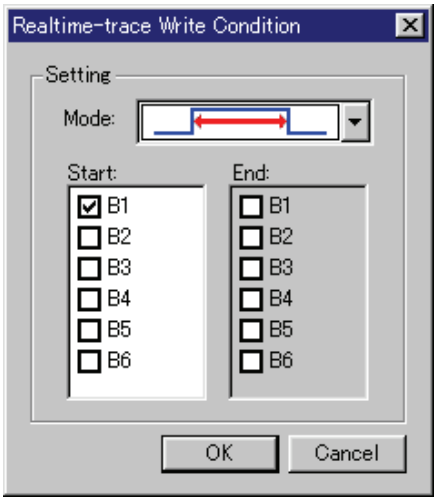
- For condition 5, choose the Mode shown below and set the Start and End events.



- For condition 6, choose the Mode shown below and set the Start event.



- For condition 7, choose the Mode shown below and set the Start event.



6.11 Trace Window

The Trace Window is used to display the results of real-time trace measurement.
The debugger for 740 doesn't support this function.
The measurement result can be displayed in the following display modes.

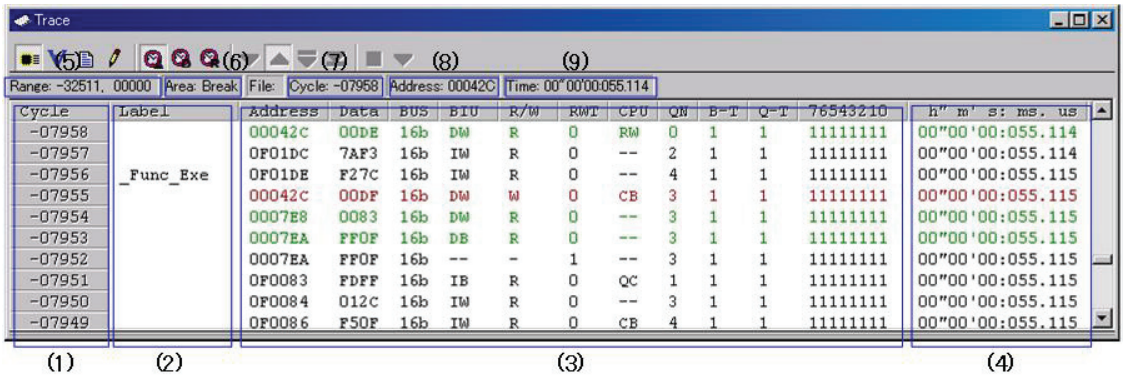
- Bus mode
  This mode allows you to inspect cycle-by-cycle bus information. The display content depends on the MCU and simulator system used. In addition to bus information, this mode allows disassemble, source line or data access information to be displayed in combination.
- Disassemble mode
  This mode allows you to inspect the executed instructions. In addition to disassemble information, this mode allows source line or data access information to be displayed in combination.
- Data access mode
  This mode allows you to inspect the data read/write cycles. In addition to data access information, this mode allows source line information to be displayed in combination.
- Source mode
  This mode allows you to inspect the program execution path in the source program.

The measurement result is displayed when a trace measurement has finished. When a trace measurement restarts, the window display is cleared.

The range of a trace measurement can be altered in the Trace Point Setting Window. For details about this window, refer to "6.10 Trace Point Setting Window." With default settings, the trace information immediately before the program has stopped is recorded.

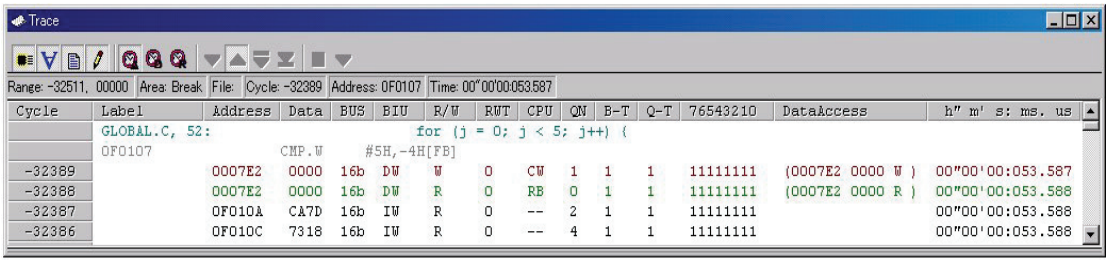
6.11.1 Configuration of Bus Mode

When bus mode is selected, trace information is displayed in bus mode. Bus mode is configured as shown below.
The display content in bus mode differs depending on the MCU or simulator system used.



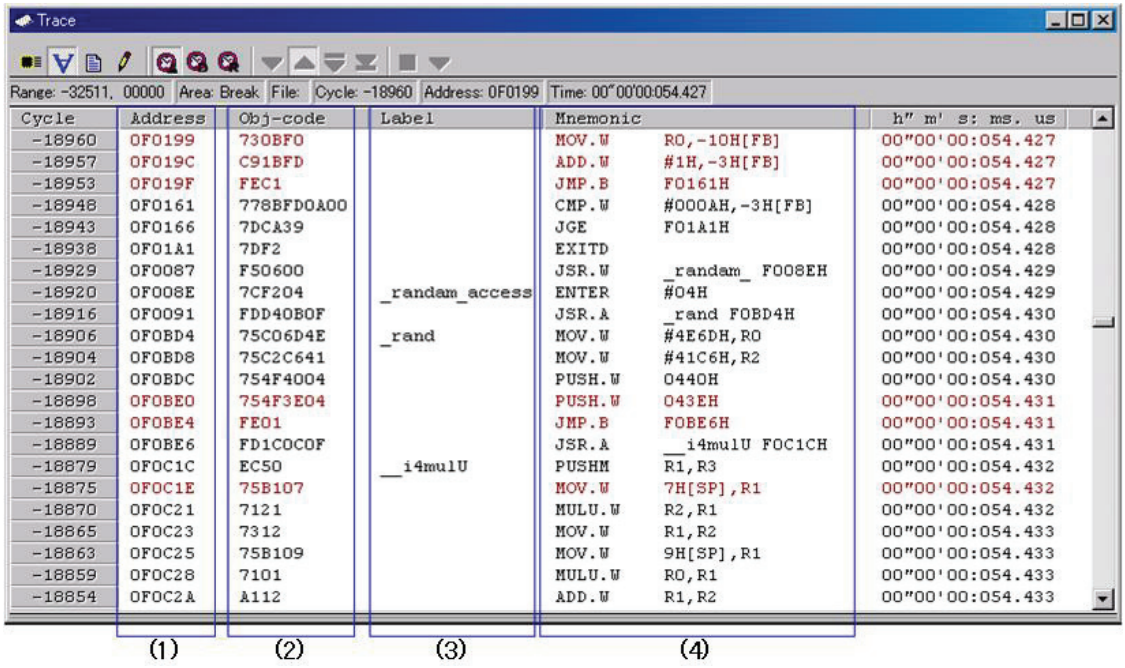
1. Cycle display area:
   Shows trace cycles. Double-click here to bring up a dialog box to change the displayed cycle.
2. Label display area:
   Shows labels corresponding to address bus information. Double-click here to bring up a dialog box to search for addresses.
3. Bus information display area:
   The content displayed here differs depending on the MCU or simulator system used.
   - "6.11.6 Display of bus information on the Simulator Debugger"
4. Time information display area:
   Shows time information of trace measurement result. One of the following three modes can be selected from the menu.
   - Absolute Time:Shows an elapsed time from the time the program started running up to now in terms of absolute time (default).
   - Differences:Shows a differential time from the immediately preceding cycle.
   - Relative Time:Shows a relative time from the selected cycle. Note, however, that this mode changes to the absolute time display mode when the trace measurement result is updated.
5. Acquired range of trace measurement result:
   Shows the currently acquired range of trace measurement result.
6. Trace measurement range:
   Shows the currently set range of trace measurement.
7. First line cycle:
   Shows the cycle of the first line displayed.
8. First line address:
   Shows the address of the first line displayed.
9. First line time:
   First line time: Shows the time information of the first line displayed.
10. Window splitting box:
    Double-clicking this box splits the window into parts.

In addition to bus information, the window can display disassemble, source line or data access information in combination. In this case, the display will be similar to the one shown below.



6.11.2 Configuration of Disassemble Mode

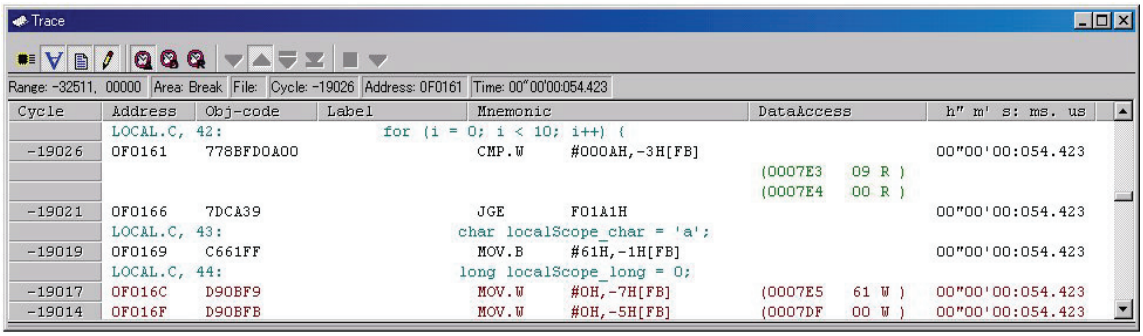
When disassemble mode is selected while bus mode is unselected, trace information is displayed in disassemble mode. Disassemble mode is configured as shown below.



1. Address display area:
   Shows addresses corresponding to instructions. Double-click here to bring up a dialog box to search for addresses.
2. Object code display area:
   Shows the object codes of instructions.
3. Label display area:
   Shows labels corresponding to instruction addresses. Double-click here to bring up a dialog box to search for addresses.
4. Mnemonic display area:
   Shows the mnemonics of instructions.

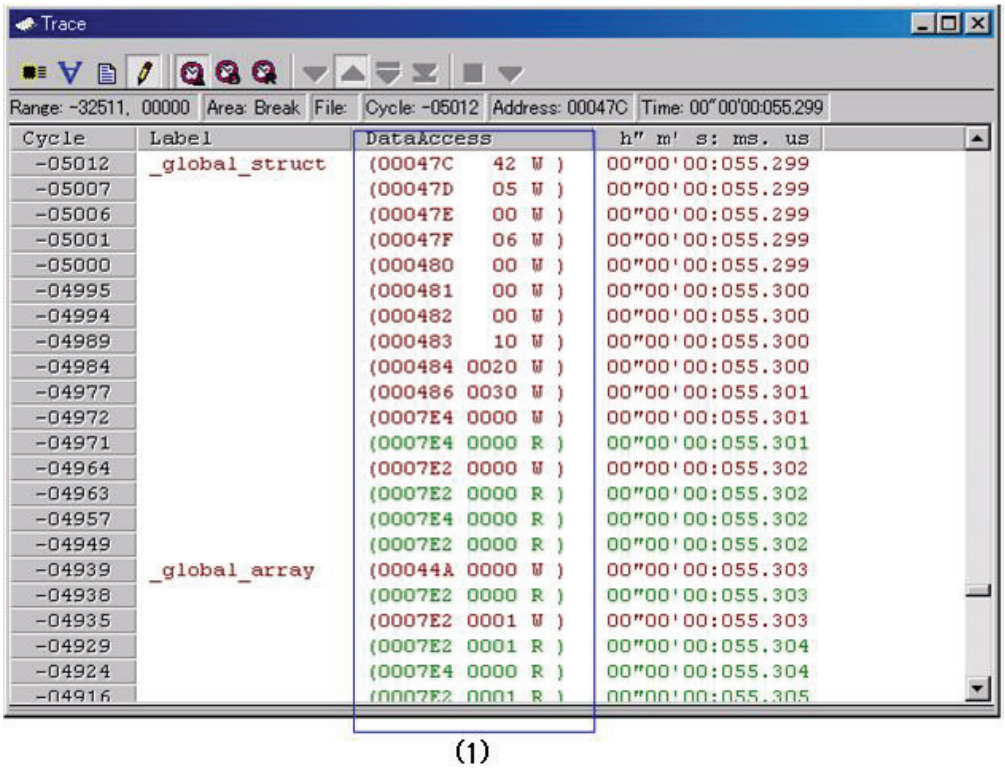
Other display areas are the same as in bus mode.

In addition to disassemble information, the window can display source line or data access information in combination. In this case, the display will be similar to the one shown below.



6.11.3 Configuration of Data Access Mode

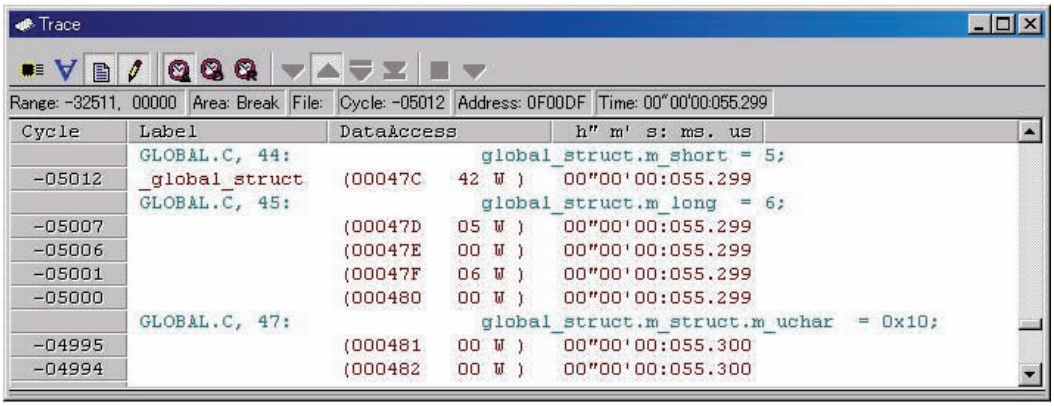
When data access mode is selected while bus mode and disassemble mode are unselected, trace information is displayed in data access mode. Data access mode is configured as shown below.



1. Data access display area:
   Shows data access information. If the information displayed here is "000400 1234 W," for example, it means that data "1234H" was written to the address 000400H in 2-byte width.

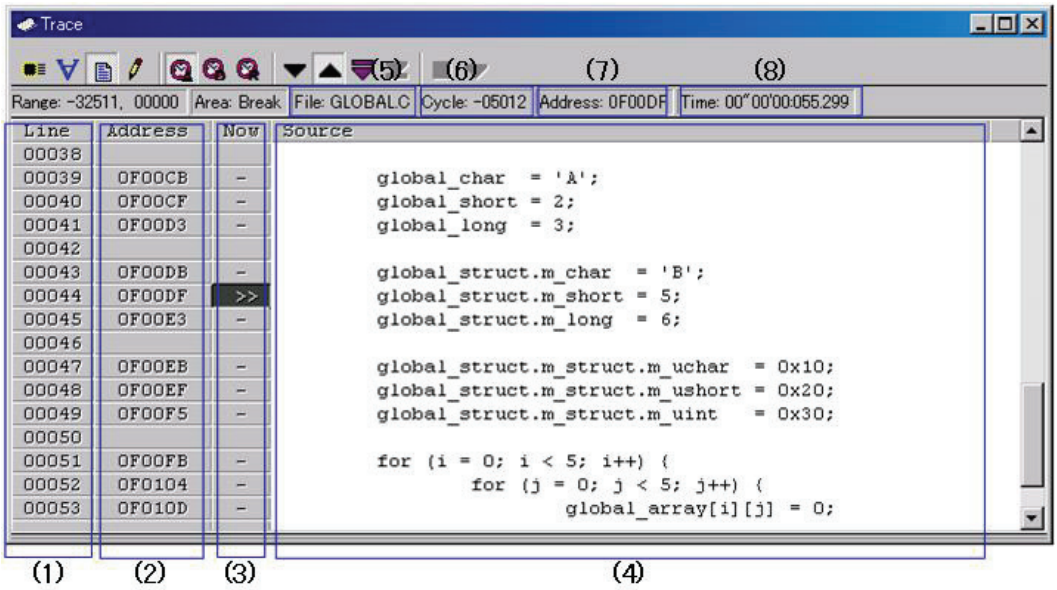
Other display areas are the same as in bus mode.

In addition to data access information, the window can display source line information in combination. In this case, the display will be similar to the one shown below.



6.11.4 Configuration of Source Mode

When only source mode is selected, trace information is displayed in source mode. Source mode is configured as shown below.



1. Line number display area:
   Shows the line number information of the displayed file. Double-click here to bring up a dialog box to change the displayed file.
2. Address display area:
   Shows addresses corresponding to source lines. Double-click here to bring up a dialog box to search for addresses.
3. Referenced cycle display area:
   Shows the currently referenced cycle that is marked by ">>." Furthermore, the addresses corresponding to source lines, if any, are marked by "-."
4. Source display area:
   Shows the content of the source file.
5. File name:
   Shows the file name of the currently displayed source file.
6. Referenced cycle:
   Shows the currently referenced cycle.
7. Referenced address:
   Shows the address corresponding to the currently referenced cycle.
8. Referenced time:
   Shows the time information corresponding to the currently referenced cycle.

Other display areas are the same as in bus mode.

6.11.5 Extended Menus

This window has the following popup menus that can be brought up by right-clicking in the window.

Menu		Function
BUS		Display the information of BUS mode.
DIS		Display the information of Disassemble mode.
SRC		Display the information of Source mode.
DATA		Display the information of Data access mode.
View	Cycle...	Changes the displayed position by specifying a cycle.
	Address...	Changes the displayed position by searching an address.
	Source...	Display a selected source file.
Time	Absolute Time	Shows elapsed time from the time the program started running up to now in terms of absolute time.
	Differences	Shows a differential time from the immediately preceding displayed cycle.
	Relative Time	Shows a relative time from the currently selected cycle.
Trace	Forward	Changes the direction of search to forward direction.
	Backward	Changes the direction of search to reverse direction.
	Step	Searches in Step mode in the specified direction of search.
	Come	Searches in Come mode in the specified direction of search.
	Stop	Stops trace measurement in the middle and displays the measured content at the present point of time.
	Restart	Restarts trace measurement.
Layout...		Change layout of the corrent view.
Copy		Copy selected lines.
Save...		Save trace data to file.
Load...		Load trace data from file.
Toolbar display		Display toolbar.
Customize toolbar...		Open toolbar customize dialog box.
Allow Docking		Allow window docking.
Hide		Hide window.

6.11.6 Display of bus information on the Simulator Debugger

From left to right, the contents are as follows:

- Address
  The status of the address bus
- Data
  The status of the data bus
- Size
  Indicates the data access size.

Product	Display format	Size
the R32C Debugger	DB DW DL	8bit 16bit 32bit
the M32C Debugger the M16C/R8C Debugger	DB DW	8bit 16bit

- Type
  Indicates that data has been accessed.

Display format	Status
Code *1	Instruction fetch
Data	Data access

*1 The Code data displayed by the R32C Debugger are fixed to 32 bits long, with the rest of data omitted.
The Code data displayed by the M32C Debugger, M16C/R8C Debugger are fixed to 16 bits long, with the rest of data omitted.

- R/W
  Indicates the data access status.

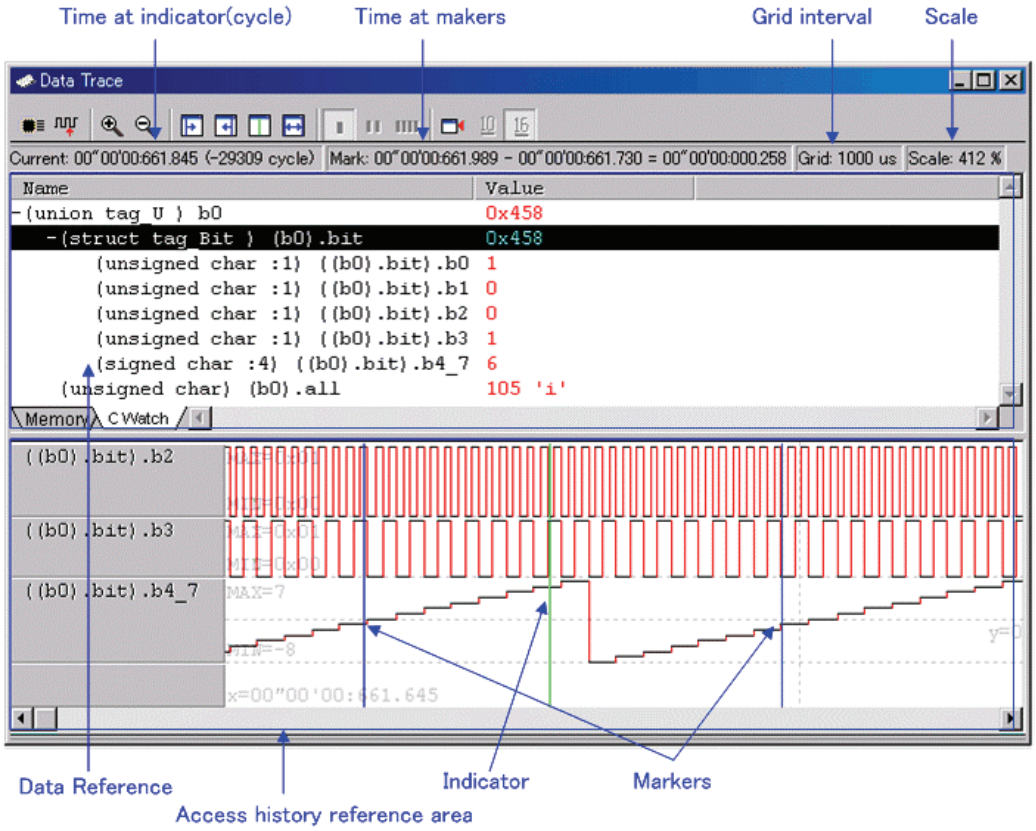
Display format	Status
R	Read
W	Write

If Type is Code, the status is always R (code read).

- h" m' s: ms.us
  Show the elapsed time from the target program beginning.
  The value enclosed in ( ) that follows indicates a total amount of instruction execution cycles reckoning from when the program started to run.

6.12 Data Trace Window

The Data Trace Window is used to analyze the results of real-time trace measurements and graphically show data access information.
The debugger for 740 doesn't support this function.
It operates in conjunction with Trace Window.



- In the data reference area, you can inspect memory values at the point of a cycle currently in interest or the values of registered C variables.
- In the access history reference area, you can see the history of accesses to registered addresses in chart form.
- In conjunction with the Trace Window, you can inspect memory values at the point of a cycle you are watching in the Trace Window. Conversely, you can show the cycle in the Trace Window which you are watching in the Data Trace Window.

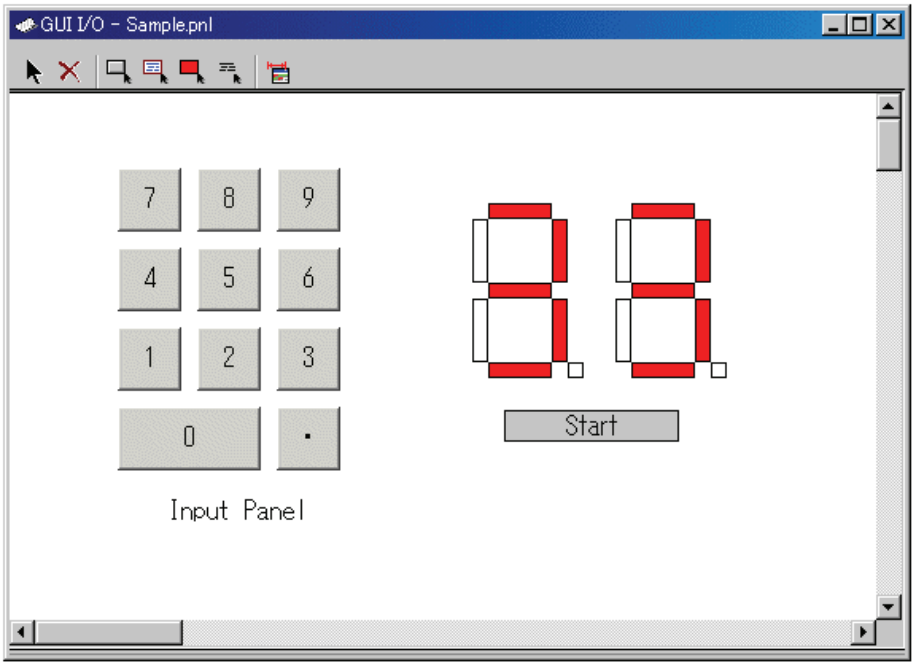
6.12.1 Extended Menus

This window has the following popup menus that can be brought up by right-clicking in the window.

Menu		Function
Analyze Trace Data		Analyze the realtime-trace data.
Set Cycle...		Specify the display cycle.
Sync with Trace Window		Synchronize with Trace Window.
Data Length	1byte	Display in 1Byte unit.
	2bytes	Display in 2Byte unit.
	4bytes	Display in 4Byte unit.
Radix	Hex	Display in Hexadecimal.
	Dec	Display in Decimal.
Address...		Display from specified address.
Add C Watch		Add C watchpoint.
Remove C Watch		Remove the selected C watchpoint.
Hide Type Name		Hide type names from variables.
Add...		Adds new watch item into Access History Reference Area.
Remove		Removes the selected watch item from Access History Reference Area.
Zoom	Zoom In	Increase the display scale.
	Zoom Out	Decrease the display sacle.
	Zoom...	Specify the display scale.
Marker	Start Marker	Move the start marker in the display area.
	End Marker	Move the end marker in the display area.
	Indicator	Move the indicator in the display area.
	Adjust	Set cycle range between markers.
Change Grid Interval...		Change the grid interval.
Change Row Setting...		Change setting of the selected row.
Color...		Change the display color.
Toolbar display		Display toolbar.
Customize toolbar...		Open toolbar customize dialog box.
Allow Docking		Allow window docking.
Hide		Hide window.

6.13 GUI I/O Window

The GUI I/O window allows you for port input by creating a user target system key input panel (button) in the window and clicking the created button. And this window also allows you to implement the user target system output panel in the window.



- You can arrange the following parts on the window.
  - Label (character string)
    Displays/erases a character string specified by the user when any value is written to the specified address (bit).
  - LED
    Changes the display color of any area when any value is written to the specified address (bit). (Substitution for LED ON)
  - Button
    A virtual port input or virtual interrupt (the simulator debugger only for the latter) can be executed at the time the button is pressed.
  - Text
    Display the text string.
- You can also save the created panel in a file and reload it.
- You can set up to 200 address points to the created part. If different addresses are set to the individual parts, you can arrange up to 200 parts.

6.13.1 Extended Menus

This window has the following popup menus that can be brought up by right-clicking in the window.

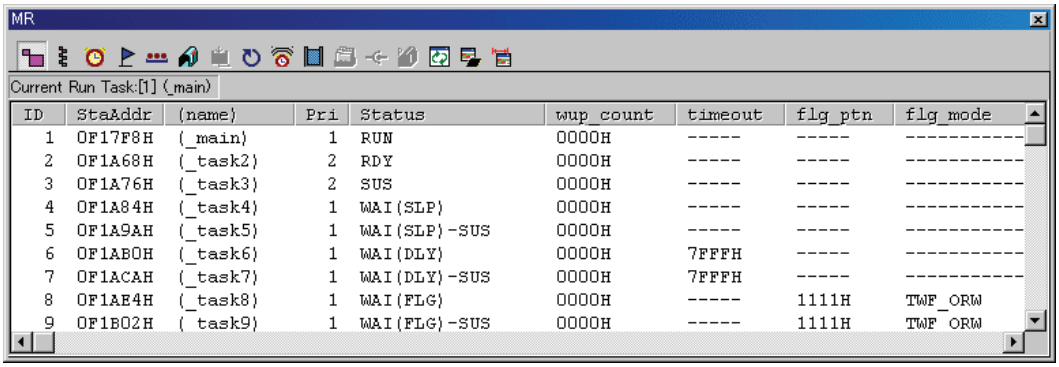
Menu	Function
Select Item	Select an I/O item.
Delete	Delete the selected I/O item.
Copy	Copy the selected I/O item.
Paste	Paste the copied I/O item.
Create Button	Create a new button item.
Create Label	Create a new label item.
Create LED	Create a new LED item.
Create Text	Create a new text item.
Display grid	Display the grid line.
Save...	Save I/O panel file.
Load...	Load I/O panel file.
Sampling Period...	Set RAM monitor sampling period.
Toolbar display	Display toolbar.
Customize toolbar...	Open toolbar customize dialog box.
Allow Docking	Allow window docking.
Hide	Hide window.

6.14 MR Window

Use the MR Window to display the status of the realtime OS.

The debugger for R32C doesn't support this function.
The debugger for 740 doesn't support this function.

You can only use the MR Window when you have downloaded a program that uses the realtime OS (if the downloaded program does not use the MR, nothing is displayed in the MR Window when it is opened.)



ID	StaAddr	(name)	Pri	Status	wup_count	timeout	flg_ptn	flg_mode
1	0F17F8H	(_main)	1	RUN	0000H	-----	-----	----------
2	0F1A68H	(_task2)	2	RDY	0000H	-----	-----	----------
3	0F1A76H	(_task3)	2	SUS	0000H	-----	-----	----------
4	0F1A84H	(_task4)	1	WAI(SLP)	0000H	-----	-----	----------
5	0F1A9AH	(_task5)	1	WAI(SLP)-SUS	0000H	-----	-----	----------
6	0F1AB0H	(_task6)	1	WAI(DLY)	0000H	7FFFH	-----	----------
7	0F1ACAH	(_task7)	1	WAI(DLY)-SUS	0000H	7FFFH	-----	----------
8	0F1AE4H	(_task8)	1	WAI(FLG)	0000H	-----	1111H	TWF_ORW
9	0F1B02H	(_task9)	1	WAI(FLG)-SUS	0000H	-----	1111H	TWF_ORW

- You can open the MR window as many as the number of display modes .
- By clicking the desired button, the MR window display mode changes and the display data also changes.
- By double-clicking the desired task line, you can display the context data of the task.
- You can drag the cursor to change the width of the display area in each mode.
- If the downloaded program does not use MR, you cannot select all the menu which will select the display mode.
- The supported display mode is as follows.

If a target program created by MR30 or MR308 conformed to uITRON4 is downloaded, this window supports the displays listed below.

- Task status
- Ready queue status
- Timeout queue status
- Event flag status
- Semaphore status
- Mailbox status
- Data queue status
- Cyclic handler status
- Alarm handler status
- Memory pool status

If a target program created by MR30 or MR308 conformed to uITRON3 is downloaded, this window supports the displays listed below.

If a target program created on MR30 V.1.00 is downloaded, the MPL mode cannot be used on MR30. (You cannot select the menu which changes the current mode to the MPL mode.)

- Task status
- Ready queue status
- Timeout queue status
- Event flag status
- Semaphore status
- Mailbox status
- Cyclic handler status
- Alarm handler status
- Memory pool status

ATTENTION

Please use the startup file (crt0mr.axx/start.axx) whoes contents matches with the version of MRxx, when you make downloaded program. The MR Window and MR command will not run properly if the startup file you uses don't match with the version of MRxx.

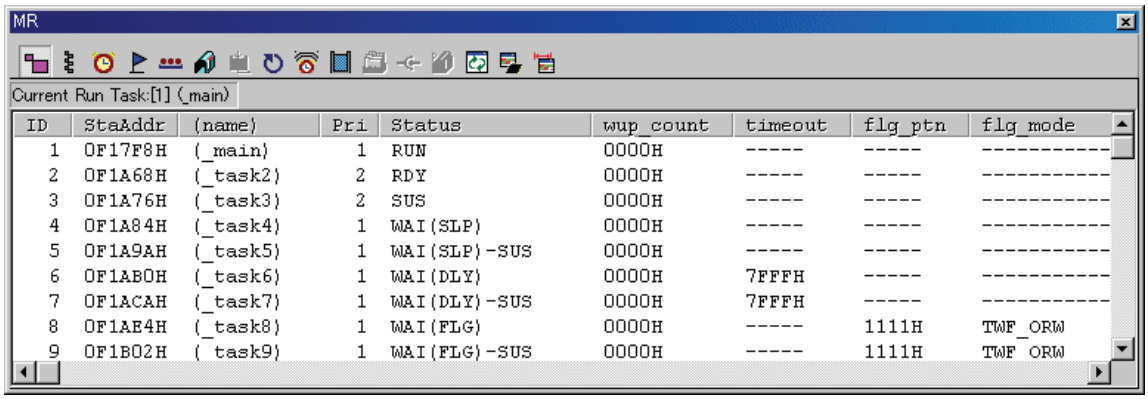
6.14.1.1 Extended Menus

This window has the following popup menus that can be brought up by right-clicking in the window.

Menu		Function
Mode	Task	Displays Task status.
	Ready Queue	Displays Ready status.
	Timeout Queue	Displays Timeout status.
	Event Flag	Displays Event Flag status.
	Semaphore	Displays Semaphore status.
	Mailbox	Displays Mailbox status.
	Data Queue	Displays Data Queue status.
	Cyclic Handler	Displays Cyclic Handler status.
	Alarm Handler	Displays Alarm Handler status.
	Memory Pool	Displays Memory Pool status.
	Message Buffer	Displays Message Buffer status.
	Port	Displays Port status.
	Mailbox(with Priority)	Displays Mailbox(with Priority) status.
Context...		Displays Context.
Layout	Status Bar	Switch display or non-display of status bar.
Refresh		Refresh memory data.
RAM Monitor	Enable RAM Monitor	Switch enable or disable RAM Monitor function.
	Sampling Period...	Set RAM Monitor sampling period.
Toolbar display		Display toolbar.
Customize toolbar...		Open toolbar customize dialog box.
Allow Docking		Allow window docking.
Hide		Hide window.

6.14.2 Display the Task Status

In the MR window, select Popup Menu - [Mode] -> [Task].



By double-clicking any line, the information on the task context is displayed in the Context dialog. For details on the Context dialog, see "6.14.12 Display the Task Context"
The following data is displayed in the status bar.

6.14.2.1 Display the Task Status(When the realtime OS is MRxx conformed to uITRON specifications V.3.0.)

All the tasks defined in the configuration are listed in the order of ID number. The function of each item is as described below. (When the realtime OS is MRxx conformed to uITRON specifications V.3.0.)

Items	Contents
ID	Task ID
StaAddr	Starting address of task
(name)	Task name
Pri	Priority
Status*1	Task status
wup_count	Wake-up count
timeout	Timeout value
flg_ptn	Wait bit pattern of event flag
flg_mode*2	Wait cancellation condition of event flag

- *1Task Status

Display	Status
RUN	Run status
RDY	Ready Status
SUS	Suspend status
DMT	Dormant status
WAI(SLP)	Sleep state
WAI(SLP)-SUS	Sleep state(double wait)
WAI(DLY)	Time wait state duw to dly_tsk
WAI(DLY)-SUS	Time wait state duw to dly_tsk(double wait)
WAI(FLG)	Event flagwait status
WAI(FLG)-SUS	Event flagwait status(double wait)
WAI(SEM)	Semaphore wait status
WAI(SEM)-SUS	Semaphore wait status(double wait)
WAI(MBX)	Message wait status
WAI(MBX)-SUS	Message wait status(double wait)
WAI(SLP-TMO)	Sleep state with time-out
WAI(SLP-TMO)-SUS	Sleep state with time-out(double wait)
WAI(FLG-TMO)	Event flag wait state with time-out
WAI(FLG-TMO)-SUS	Event flag wait state with time-out(double wait)
WAI(SEM-TMO)	Semaphore wait state with time-out
WAI(SEM-TMO)-SUS	Semaphore wait state with time-out(double wait)
WAI(MBX-TMO)	Message wait state with time-out
WAI(MBX-TMO)-SUS	Message wait state with time-out(double wait)

- *2Display the Wait Cancellation Condition of Event Flag

flg_mode	Status
TWF_ANDW	Waits for all bits set in the wait bit pattern to be set (AND wait)
TWF_ANDW+TWF_CLR	Clears the event flag to 0 when an AND wait has occurred and the task wait status has been cancelled
TWF_ORW	Waits for any one bit set in the wait bit pattern to be set (OR wait)
TWF_ORW+TWF_CLR	Clears the event flag to 0 when an OR wait has occurred and the task wait status has been cancelled

6.14.2.2 Display the Task Status(When the realtime OS is MRxx conformed to uITRON specifications V.4.0.)

All the tasks defined in the configuration are listed in the order of ID number. The function of each item is as described below. (When the realtime OS is MRxx conformed to uITRON specifications V.4.0.)

Items	Contents
ID	Task ID
Name	Task name
Pri	Priority
Status*1	Task status
Wupcnt	Wake-up count
Actcnt	Activated count
Tmout	Timeout value
Flgptn	Wait bit pattern of event flag
Wfmode*2	Wait cancellation condition of event flag

- *1Task Status

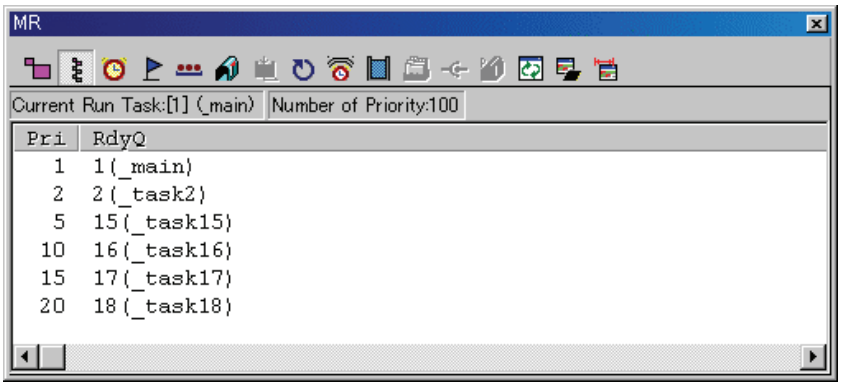
Display	Status
RUN	Run status
RDY	Ready Status
SUS	Suspend status
DMT	Dormant status
WAI(SLP)	Sleep state
WAI(SLP)-SUS	Sleep state(double wait)
WAI(DLY)	Time wait state due to dly_tsk
WAI(DLY)-SUS	Time wait state due to dly_tsk(double wait)
WAI(FLG)	Event flagwait status
WAI(FLG)-SUS	Event flagwait status(double wait)
WAI(SEM)	Semaphore wait status
WAI(SEM)-SUS	Semaphore wait status(double wait)
WAI(MBX)	Message wait status
WAI(MBX)-SUS	Message wait status(double wait)
WAI(SDTQ)	Transmission data wait status
WAI(SDTQ)-SUS	Transmission data wait status(double wait)
WAI(RDTQ)	Reception data wait status
WAI(RDTQ)-SUS	Reception data wait status(double wait)
WAI(VSDTQ)	Transmission extended data wait status
WAI(VSDTQ)-SUS	Transmission extended data wait status(double wait)
WAI(VRDTQ)	Reception extended data wait status
WAI(VRDTQ)-SUS	Reception extended data wait status(double wait)
WAI(MPF)	Fixed length memory pool wait
WAI(MPF)-SUS	Fixed length memory pool wait(double wait)
WAI(SLP-TMO)	Sleep state with timeout
WAI(SLP-TMO)-SUS	Sleep state with timeout(double wait)
WAI(FLG-TMO)	Event flag wait state with timeout
WAI(FLG-TMO)-SUS	Event flag wait state with timeout(double wait)
WAI(SEM-TMO)	Semaphore wait state with timeout
WAI(SEM-TMO)-SUS	Semaphore wait state with timeout(double wait)
WAI(MBX-TMO)	Message wait state with time-out
WAI(MBX-TMO)-SUS	Message wait state with timeout(double wait)
WAI(SDTQ-TMO)	Transmission data with timeout wait status
WAI(SDTQ-TMO)-SUS	Transmission data with timeout wait status(double wait)
WAI(RDTQ-TMO)	Reception data with timeout wait status
WAI(RDTQ-TMO)-SUS	Reception data with timeout wait status(double wait)
WAI(VSDTQ-TMO)	Transmission extended data with timeout wait status
WAI(VSDTQ-TMO)-SUS	Transmission extended data with timeout wait status(double wait)
WAI(VRDTQ-TMO)	Reception extended data with timeout wait status
WAI(VRDTQ-TMO)-SUS	Reception extended data with timeout wait status(double wait)
WAI(MPF-TMO)	Fixed length memory pool with timeout wait
WAI(MPF-TMO)-SUS	Fixed length memory pool with timeout wait(double wait)

- *2Display the Wait Cancellation Condition of Event Flag

Wfmode	Status
TWF_ANDW	Waits for all bits set in the wait bit pattern to be set (AND wait)
TWF_ORW	Waits for any one bit set in the wait bit pattern to be set (OR wait)

6.14.3 Display the Ready Queue Status

In the MR window, select Popup Menu - [Mode] -> [Ready Queue].



The following data is displayed in the status bar.

6.14.3.1 Display the Ready Queue Status(When the realtime OS is MRxx conformed to uITRON specifications V.3.0.)

The function of each item is as described below. (When the realtime OS is MRxx conformed to uITRON specifications V.3.0.)

Item	Contents
Pri	Displays priority
RdyQ	Shows the ID Nos. and task names of tasks in the ready queue

- Up to 8 characters of the task name is displayed in the RdyQ field. When the task name exceeds 8 characters, the extra characters are omitted.

6.14.3.2 Display the Ready Queue Status(When the realtime OS is MRxx conformed to uITRON specifications V.4.0.)

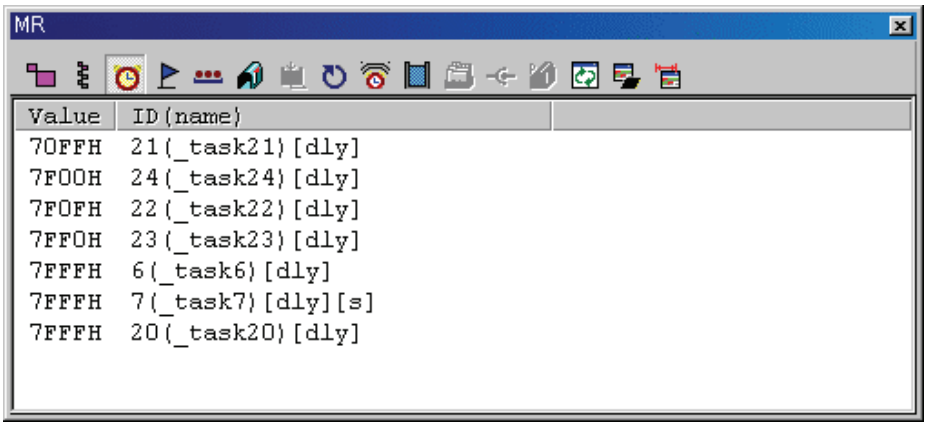
The function of each item is as described below. (When the realtime OS is MRxx conformed to uITRON specifications V.4.0.)

Item	Contents
Pri	Displays priority
Ready Queue	Shows the ID Nos. and task names of tasks in the ready queue

- Up to 8 characters of the task name is displayed in the Ready Queue field. When the task name exceeds 8 characters, the extra characters are omitted.

6.14.4 Display the Timeout Queue Status

In the MR window, select Popup Menu - [Mode] -> [Timeout Queue].



6.14.4.1 Display the Timeout Queue Status(When the realtime OS is MRxx conformed to uITRON specifications V.3.0.)

The function of each item is as described below.
Tasks waiting at present are displayed in the descending order of timeout value. (When the realtime OS is MRxx conformed to uITRON specifications V.3.0.)

Item	Contents
Value	Shows the timeout value of each task
ID(name)	Shows the ID No. and task name of the tasks in the timeout queue

- Following character strings are used to indicate the type of wait state.

Character string	Wait state
[slp]	Wait due to tslp_tsk
[dly]	Wait due to dly_tsk
[flg]	Wait due to twai_flg
[sem]	Wait due to twai_sem
[mbx]	Wait due to trcv_msg

- When a task connected to the timeout queue is in the state of forced waiting (double waiting), a string "[s]", which indicates double waiting, is appended to a string displayed in the ID (name) field.

Normal display	26(_task26)
Display when in WAIT-SUSPEND	26(_task26)[s]

6.14.4.2 Display the Timeout Queue Status(When the realtime OS is MRxx conformed to uITRON specifications V.4.0.)

The function of each item is as described below.
Tasks waiting at present are displayed in the descending order of timeout value. (When the realtime OS is MRxx conformed to uITRON specifications V.4.0.)

Item	Contents
Tmout	Shows the timeout value (ms) of each task
ID(Name)	Shows the ID No. and task name of the tasks in the timeout queue

- Following character strings are used to indicate the type of wait state.

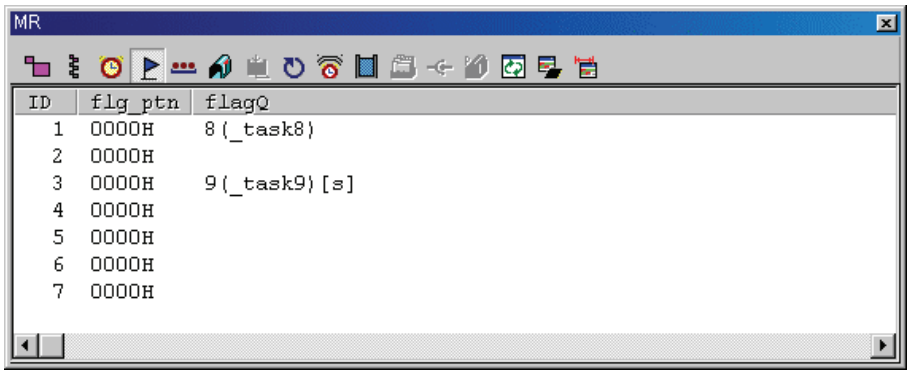
Character string	Wait state
[slp]	Wait due to tslp_tsk
[dly]	Wait due to dly_tsk
[flg]	Wait due to twai_flg
[sem]	Wait due to twai_sem
[mbx]	Wait due to trcv_mbx
[mpf]	Wait due to tget_mpf
[sdtq]	Wait due to tsnd_dtq
[rdtq]	Wait due to trcv_dtq
[vsdtq]	Wait due to vtsnd_dtq
[vrdtq]	Wait due to vtrcv_dtq

- When a task connected to the timeout queue is in the state of forced waiting (double waiting), a string "[s]", which indicates double waiting, is appended to a string displayed in the ID(Name) field.

Normal display	26(_task26)
Display when in WAIT-SUSPEND	26(_task26)[s]

6.14.5 Display the Event Flag Status

In the MR window, select Popup Menu - [Mode] -> [Event Flag].



6.14.5.1 Display the Event Flag Status(When the realtime OS is MRxx conformed to uITRON specifications V.3.0.)

All the event flags defined in the configuration are listed in the order of ID number. The function of each item is listed below. (When the realtime OS is MRxx conformed to uITRON specifications V.3.0.)

Item	Contents
ID	ID No. of event flag
flg_ptn	Bit pattern of each event flag
flagQ	Task ID Nos. and task names in the event flag queue

- When a task connected to the event flag queue is in the state of waiting with timeout enabled (waiting in twai_flg), a string "[tmo]", which indicates a state of waiting with timeout enabled, is appended to a string displayed in the flag Q field.
  When a task connected to the event flag queue is in the state of forced waiting (double waiting), a string "[s]", which indicates double waiting, is appended to a string displayed in the flag Q field.

Normal Display	26(_task26)
Display when in WAIT-SUSPEND	26(_task26)[s]
Display when in WAIT-SUSPEND with time out	26(_task26)[tmo][s]

- Up to 8 characters can be displayed in the task name in the flag Q field.
  If a task name exceeds 8 characters, the extra characters are omitted.

6.14.5.2 Display the Event Flag Status(When the realtime OS is MRxx conformed to uITRON specifications V.4.0.)

All the event flags defined in the configuration are listed in the order of ID number. The function of each item is listed below. (When the realtime OS is MRxx conformed to uITRON specifications V.4.0.)

Item	Contents
ID	ID No. of event flag
Flgatr	Attribute of each event flag
Flgptn	Bit pattern of each event flag
Flag Queue	Task ID Nos. and task names in the event flag queue

- The following are displayed in the Flgatr area:

TA_TFIFO	Task wait queue is in FIFO order
TA_TPRI	Task wait queue is in task priority order
TA_WSGL	Only one task is allowed to be in the waiting state for the eventflag
TA_WMUL	Multiple tasks are allowed to be in the waiting state for the eventflag
TA_CLR	Eventflag's bit pattern is cleared when a task is released from the waiting state for that eventflag

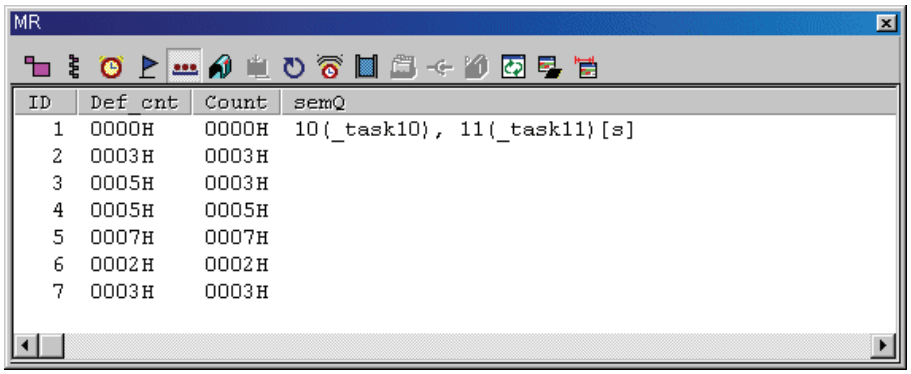
- When a task connected to the event flag queue is in the state of waiting with timeout enabled (waiting in twai_flg), a string "[tmo]", which indicates a state of waiting with timeout enabled, is appended to a string displayed in the Flag Queue field.
  When a task connected to the event flag queue is in the state of forced waiting (double waiting), a string "[s]", which indicates double waiting, is appended to a string displayed in the Flag Queue field.

Normal Display	26(_task26)
Display when in WAIT-SUSPEND	26(_task26)[s]
Display when in WAIT-SUSPEND with time out	26(_task26)[tmo][s]

- Up to 8 characters can be displayed in the task name in the Flag Queue field.
  If a task name exceeds 8 characters, the extra characters are omitted.

6.14.6 Display the Semaphore Status

In the MR window, select Popup Menu - [Mode] -> [Semaphore].



6.14.6.1 Display the Semaphore Status(When the realtime OS is MRxx conformed to uITRON specifications V.3.0.)

All the SEMs defined in the configuration are listed in the order of ID number. The function of each item is listed below. (When the realtime OS is MRxx conformed to uITRON specifications V.3.0.)

Item	Contents
ID	ID No. of semaphore
Def_cnt	Default value of semaphore counter
Count	Semaphore count
semQ	Task ID Nos. and task names in the semaphore queue

- When a task connected to the SEM queue is in the state of waiting with timeout enabled (waiting in twai_sem), a string "[tmo]", which indicates a state of waiting with timeout enabled, is appended to a string displayed in the semQ field.
  When a task connected to the SEM queue is in the state of forced waiting (double waiting), a string "[s]", which indicates double waiting, is appended to a string displayed in the semQ field.

Normal Display	26(_task26)
Display when in WAIT-SUSPEND	26(_task26)[s]
Display when in WAIT-SUSPEND with time out	26(_task26)[tmo][s]

- Up to 8 characters can be displayed in the task name in the semQ field.
  If a task name exceeds 8 characters, the extra characters are omitted.

6.14.6.2 Display the Semaphore Status (When the realtime OS is MRxx conformed to uITRON specifications V.4.0.)

All the SEMs defined in the configuration are listed in the order of ID number. The function of each item is listed below. (When the realtime OS is MRxx conformed to uITRON specifications V.4.0.)

Item	Contents
ID	ID No. of semaphore
Sematr	Attribute of each semaphore
Semcnt	Semaphore count
Semaphore Queue	Task ID Nos. and task names in the semaphore queue

- The following are displayed in the Sematr area:

TA_TFIFO	Task wait queue is in FIFO order
TA_TPRI	Task wait queue is in task priority order

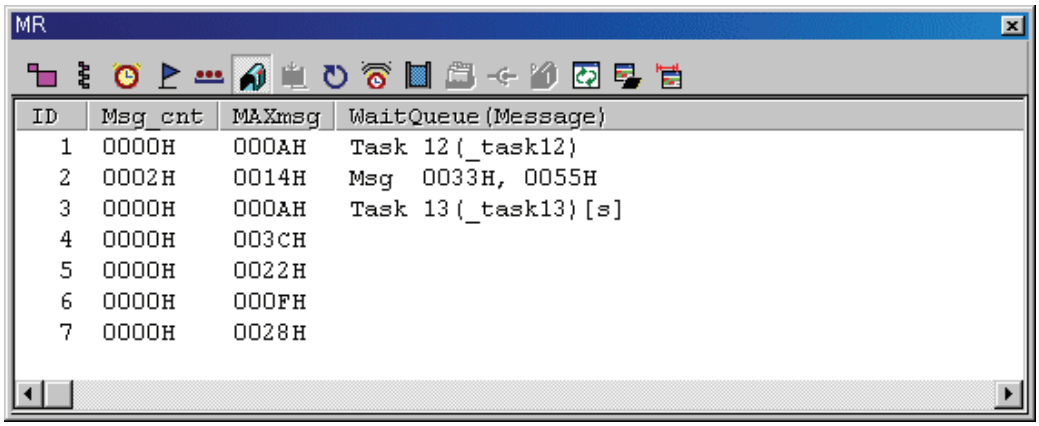
- When a task connected to the SEM queue is in the state of waiting with timeout enabled (waiting in twai_sem), a string "[tmo]", which indicates a state of waiting with timeout enabled, is appended to a string displayed in the Semaphore Queue field.
  When a task connected to the SEM queue is in the state of forced waiting (double waiting), a string "[s]", which indicates double waiting, is appended to a string displayed in the Semaphore Queue field.

Normal Display	26(_task26)
Display when in WAIT-SUSPEND	26(_task26)[s]
Display when in WAIT-SUSPEND with time out	26(_task26)[tmo][s]

- Up to 8 characters can be displayed in the task name in the Semaphore Queue field.
  If a task name exceeds 8 characters, the extra characters are omitted.

6.14.7 Display the Mailbox Status

In the MR window, select Popup Menu - [Mode] -> [Mailbox].



6.14.7.1 Display the Mailbox Status (When the realtime OS is MRxx conformed to uITRON specifications V.3.0.)

All the mail boxes defined in the configuration are listed in the order of ID number. The function of each item is listed below. (When the realtime OS is MRxx conformed to uITRON specifications V.3.0.)

Item	Contents
ID	ID No. of mailbox
Msg_cnt	Number of messages in each mailbox
MAXmsg	Maximum number of messages that can be contained in each mailbox
Wait Queue(Message)	The messages stored in the mailbox or ID No. and task name of tasks waiting for messages

- The WaitQueue (Message) field shows a string "Msg" when a message is stored (when Msg_cont as described above is non-zero), and then displays the stored message.
  When no message is stored (when Msg_cont is zero), the WaitQueue field displays a string "Task" if a task waiting for a message exists, and then displays the ID number and name of the task waiting for a message.
- When a task connected to the mail box queue is in the state of waiting with timeout enabled (waiting in trcv_msg), a string "[tmo]", which indicates the state of timeout enabled, is appended to a string displayed in the WaitQueue (Message) field.
  When a task connected to the mail box queue is in the state of forced waiting (Double waiting), a string "[s]", which indicates the state of double waiting, is appended to a string displayed in the WaitQueue (Message) field.

Normal Display	26(_task26)
Display when in WAIT-SUSPEND	26(_task26)[s]
Display when in WAIT-SUSPEND with time out	26(_task26)[tmo][s]

- Up to 8 characters can be displayed in the task name in the WaitQueue (Message) field.
  If a task name exceeds 8 characters, the extra characters are omitted.

6.14.7.2 Display the Mailbox Status (When the realtime OS is MRxx conformed to uITRON specifications V.4.0.)

All the mail boxes defined in the configuration are listed in the order of ID number. The function of each item is listed below. (When the realtime OS is MRxx conformed to uITRON specifications V.4.0.)

Item	Contents
ID	ID No. of mailbox
Mbxatr	Attribute of each mailbox
Mailbox Queue (Wait)	ID No. and task name of tasks waiting for messages
Mailbox Queue (Message)	The messages stored in the mailbox

- The following are displayed in the Mbxatr area:

TA_TFIFO	Task wait queue is in FIFO order
TA_TPRI	Task wait queue is in task priority order
TA_MFIFO	Message queue is in FIFO order
TA_MPRI	Message queue is in message priority order

- When a task connected to the mail box queue is in the state of waiting with timeout enabled (waiting in trcv_mbx), a string "[tmo]", which indicates the state of timeout enabled, is appended to a string displayed in the Mailbox Queue (Wait) field.

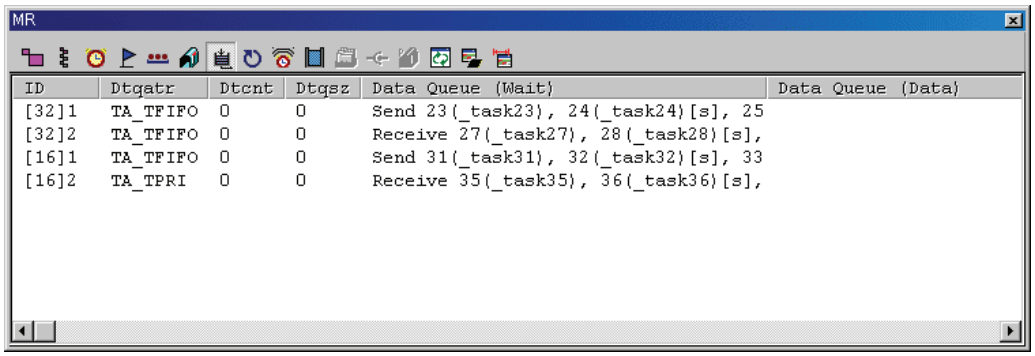
When a task connected to the mail box queue is in the state of forced waiting (Double waiting), a string "[s]", which indicates the state of double waiting, is appended to a string displayed in the Mailbox Queue (Wait) field.

Normal Display	26(_task26)
Display when in WAIT-SUSPEND	26(_task26)[s]
Display when in WAIT-SUSPEND with time out	26(_task26)[tmo][s]

- Up to 8 characters can be displayed in the task name in the Mailbox Queue (Wait) field.
  If a task name exceeds 8 characters, the extra characters are omitted.

6.14.8 Display the Data Queue Status

In the MR window, select Popup Menu - [Mode] -> [Data Queue].



6.14.8.1 Display the Data Queue Status(When the realtime OS is MRxx conformed to uITRON specifications V.4.0.)

All the data queues defined in the configuration are listed in the order of ID number. The function of each item is listed below. (When the realtime OS is MRxx conformed to uITRON specifications V.4.0.)

Item	Contents
ID	ID No. of data queue
Dtqatr	Attribute of each date queue
Dtcnt	Number of messages in each data queue
Dtqsz	Maximum number of messages that can be contained in each data queue
Data Queue (Wait)	ID No. and task name of tasks waiting for message transmission waiting or message reception waiting
Data Queue (Data)	The messages stored in the data queue

- The display of the ID field varies depending on which one is specified, the standard data(32 bits) or the extended data(16 bits).
  MR308/4
  - If the standard data(32 bits), the ID field displays a string "[32]" and data queue ID number.
  - If the extended data(16 bits), the ID field displays a string "[16]" and data queue ID number.

  MR30/4
  - If the standard data(16 bits), the ID field displays a string "[16]" and data queue ID number.
  - If the extended data(32 bits), the ID field displays a string "[32]" and data queue ID number.

- The following are displayed in the Dtqatr area:

TA_TFIFO	Task wait queue is in FIFO order
TA_TPRI	Task wait queue is in task priority order

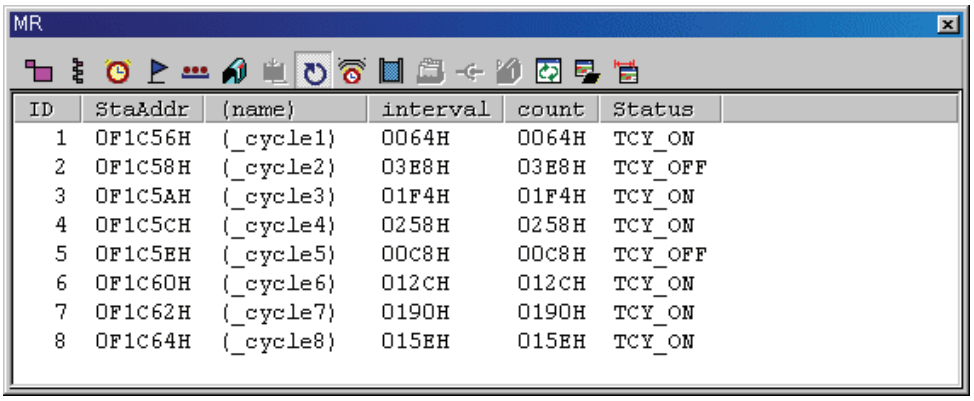
- The Data Queue (Wait) field displays a string "Send" if a task waiting for a message sending, and then displays the ID number and name of the task waiting for a message sending. Also, if a task waiting for a message receiving, displays a string "Receive" and then displays the ID number and name of the task waiting for a message receiving.
- When a task connected to the date queue is in the state of waiting with timeout enabled , a string "[tmo]", which indicates the state of timeout enabled, is appended to a string displayed in the Data Queue (Wait) field.
  When a task connected to the data queue is in the state of forced waiting (Double waiting), a string "[s]", which indicates the state of double waiting, is appended to a string displayed in the Data Queue (Wait) field.

Normal Display	26(_task26)
Display when in WAIT-SUSPEND	26(_task26)[s]
Display when in WAIT-SUSPEND with time out	26(_task26)[tmo][s]

Up to 8 characters can be displayed in the task name in the Data Queue (Wait) field.
If a task name exceeds 8 characters, the extra characters are omitted.

6.14.9 Display the Cycle Handler Status

In the MR window, select Popup Menu - [Mode] -> [Cyclic Handler].



6.14.9.1 Display the Cycle Handler Status(When the realtime OS is MRxx conformed to uITRON specifications V.3.0.)

All the cycle handlers defined in the configuration are listed in the order of ID number. The function of each item is listed below. (When the realtime OS is MRxx conformed to uITRON specifications V.3.0.)

Item	Contents
ID	ID No. of cycle handler
StaAddr	Starting address of cycle handler
(name)	Name of cycle handler
interval	Interrupt interval
count	Interrupt count
Status	Activity status of cycle start handler

- The following are displayed in the Status area:

TCY_ON	Cycle handler enabled
TCY_OFF	Cycle handler disabled

6.14.9.2 Display the Cycle Handler Status(When the realtime OS is MRxx conformed to uITRON specifications V.4.0.)

All the cycle handlers defined in the configuration are listed in the order of ID number. The function of each item is listed below. (When the realtime OS is MRxx conformed to uITRON specifications V.4.0.)

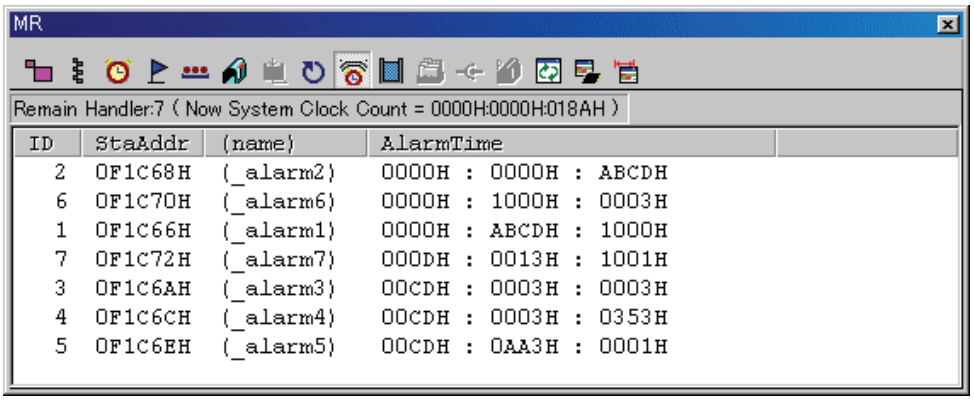
Item	Contents
ID	ID No. of cycle handler
Name	Name of cycle handler
Cycphs	The activation phase (by the millisecond)
Cyctim	The activation cycle time (by the millisecond)
Tmout	The amount of time by the millisecond remaining before the cyclic handler's next activation time
Status	Activity status of cycle start handler

- The following are displayed in the Status area:

TCYC_STA	Cycle handler is in an operational state
TCYC_STP	Cycle handler is in a non-operational state

6.14.10 Display the Alarm Handler Status

In the MR window, select Popup Menu - [Mode] -> [Alarm Handler].



When the realtime OS is MRxx conformed to uITRON specifications V.3.0, the following data is displayed in the status bar.

6.14.10.1 Display the Alarm Handler Status(When the realtime OS is MRxx conformed to uITRON specifications V.3.0.)

Of all the cycle start handlers defined in the configuration, only those which are not started yet at present are listed in the ascending order of start time. The function of each item is listed below. (When the realtime OS is MRxx conformed to uITRON specifications V.3.0.)

Item	Contents
ID	ID No. of alarm handler
StaAddr	Starting address of alarm handler
(name)	Name of alarm handler
AlarmTime	Starting time of alarm handler

6.14.10.2 Display the Alarm Handler Status(When the realtime OS is MRxx conformed to uITRON specifications V.4.0.)

Of all the cycle start handlers defined in the configuration, only those which are not started yet at present are listed in the ascending order of start time. The function of each item is listed below. (When the realtime OS is MRxx conformed to uITRON specifications V.4.0.)

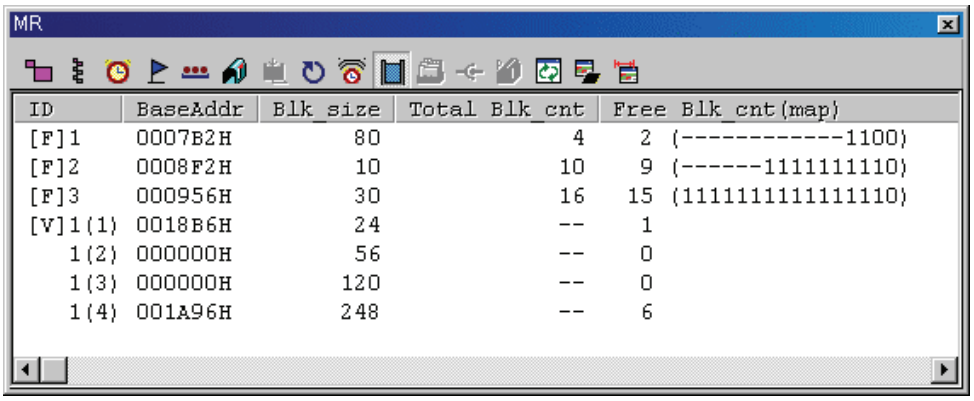
Item	Contents
ID	ID No. of alarm handler
Name	Name of alarm handler
Almtim	The amount of time by the millisecond remaining before the alarm handler's activation time
Status	Activity status of alarm handler

The following are displayed in the Status area:

TALM_STA	Alarm handler is in an operational state
TALM_STP	Alarm handler is in a non-operational state

6.14.11 Display the Memory Pool Status

In the MR window, select Popup Menu - [Mode] -> [Memory Pool].



6.14.11.1 Display the Memory Pool Status(When the realtime OS is MRxx conformed to uITRON specifications V.3.0.)

All the memory pools defined in the configuration are listed in the order of ID number. (The fixed length data comes first, and the optional length data comes after the fixed length data.) The function of each item is listed below. (When the realtime OS is MRxx conformed to uITRON specifications V.3.0.)

Item	Contents
ID	ID No. of memory pool
BaseAddr	Base address of memory pool
Blk_Size	Block size of memory pool
Total Blk_cnt	Tot a l block count of memory pool
Free Blk_cnt(map)	Number of unused blocks and information on unused memory blocks (bit information)

- The display of the ID field varies depending on which one is specified, fixed length or optional length.
  - If the data is of fixed length, the ID field displays a string "[F]" and memory pool ID number.
  - For an arbitrary length, the contents displayed on the first line are the character string "[V]," a memory pool ID number, and a block ID number. Displayed on the second to fourth lines are the memory pool ID and block ID numbers. The block ID numbers are enclosed in parentheses.
- When specifying the optional length memory pool, "--" is displayed in the Total Mlk_cut field. No bit information is displayed in the Free Blk_cnt (map) field.

- When specifying the fixed-length memory pool, the display format of each bit in the memory block information in Free Blk_cnt (map) is as shown below:

item	Contents
'0'	Memory block in use (busy)
'1'	Memory block not in use (ready)
'-'	No memory block

6.14.11.2 Display the Memory Pool Status(When the realtime OS is MRxx conformed to uITRON specifications V.4.0.)

All the memory pools are listed in the order of ID number. The function of each item is listed below. (When the realtime OS is MRxx conformed to uITRON specifications V.4.0.)

Item	Contents
ID	ID No. of memory pool
Mplatr	Attribute of each memory pool
Mpladr	Base address of memory pool
Mplsz	Size of memory pool
Blkcnt	Total block count of fixed length memory pool
Fblkcnt	Number of unused blocks and information on unused memory blocks
Memory Pool Queue	Displays the ID number and name of tasks waiting in the memory pool.

- The following are displayed in the Mplatr area:

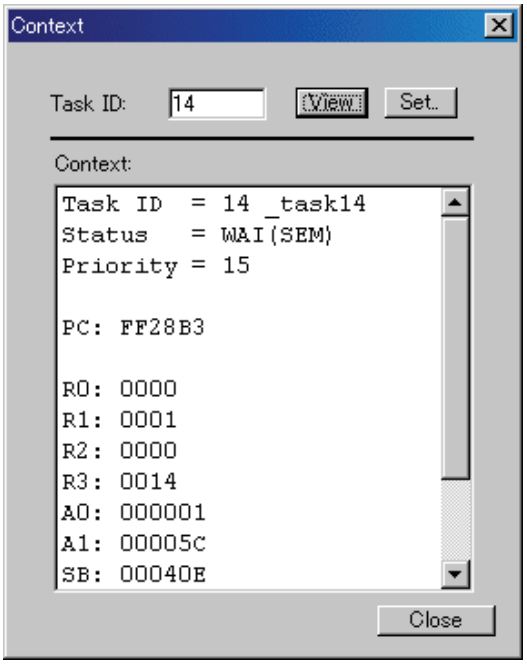
TA_TFIFO	Task wait queue is in FIFO order
TA_TPRI	Task wait queue is in task priority order

- The display of the ID field varies depending on which one is specified, fixed length or optional length.
  - If the data is of fixed length, the ID field displays a string "[F]" and memory pool ID number.
  - For an arbitrary length, the contents displayed on the first line are the character string "[V]," a memory pool ID number, and a block ID number. Displayed on the second to fourth lines are the memory pool ID and block ID numbers. The block ID numbers are enclosed in parentheses.

6.14.12 Display the Task Context

6.14.12.1 Display the Task Context

In the MR window, select Popup Menu - [Context...].
The Context dialog box is opened. The Context dialog box is used to reference/specify the context information of the specified task.
You can also open the Context dialog box by double-clicking the data display area in the task state display mode .

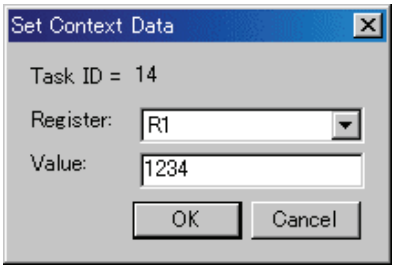


Enter the task ID number in the Task ID field and click the View button (or press the Enter key).
The context of the specified task appears in the Context field.

- If the task entered in the Task ID field is "RUN" or "DMT" when clicking the View button, the context is not displayed. (In the Context field, only the task ID and task state are displayed.)
- If a task ID number which does not exist is entered in the Task ID field when clicking the View button, an error occurs.

6.14.12.2 Change the task context

Enter the task ID number in the Task ID field in the Context dialog and click the Set button. The Set Context dialog is opened.
The Set Context dialog is used to set the specified context register value of the specified task.



Specify the register to be changed in the Register field list box and enter the value to be set in the "Value:" field.
If an expression description set in the "Value:" field is wrong, or if the specified value is outside the allowable range set for the specified register, an error occurs.

6.15 MR Trace Window

The MR Trace window measures the task execution history of a program using the real time OS and displays the result graphically.
The debugger for R32C doesn't support this function.
The debugger for 740 doesn't support this function.

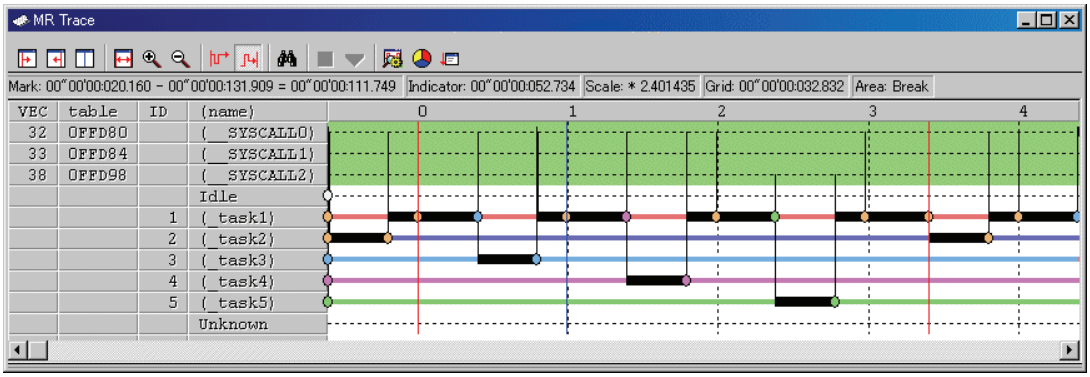
In addition to the task execution history, a history of various other operations each are traced and displayed, including interrupt processing, task state transition, and system call issuance.
This window is available only when a target program which uses our real time OS (MRxx) is downloaded.

For MR30

- For MR30, this window is available for V. 2.00 or later version. If a target program crated on MR30 V. 1.00 is downloaded, the MR Trace window will not function and not display any data.

For MR308

- The history of the high-speed interrupt can not record and display.

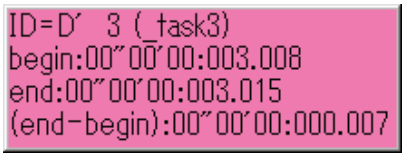


The content of each item is as follows.

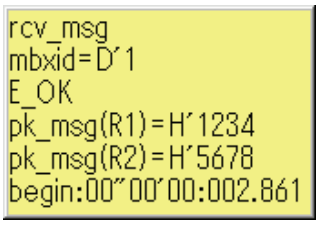
Items	Contents
VEC*1	Indicates a software interrupt number.
table	Indicates the interrupt vector table number.
ID	Indicates a task ID number.
(name)	Indicates an interrupt routine name, task name, idle processing (display "idle"), and unknown name(displayed "unknown").

When moving the mouse to the information displayed in the window, the pop up window as below is opened, showing the detailed information.

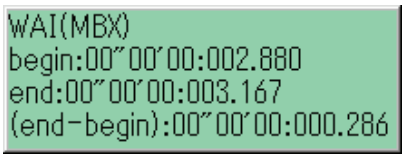
Interrupt handling or task execution history



System call issue history



Task state transition history



Following information is displayed in the status bar.

- Time value at which start marker is positioned
- Time value at which end marker is positioned
- Time width of a range indicated by start and end markers
- Time value at which indicator is positioned
- Scale factor of display
- Time width of grid line interval
- Range of measurement (trace) result

The grid lines are displayed using the start marker as the radix point.
The grid lines are displayed using the start marker as the radix point. The scale is displayed, using the time at which the start marker is positioned as 0, with the left (forward in time) set to "minus" and the right (backward in time) set to "plus".
The grid lines allow you to roughly understand the interrupt occurrence cycle and process time.
The interval time width of the displayed grid lines appears in the "Grid" area of the status bar.

The time value in the MR Trace window means the execution elapsed time using the program execution start time as 0 in all the cases. On the contrary, the numeric value above the grid lines (scale) in the MR Trace window is a relative value using the start marker as 0 (the grid interval is specified in the Value dialog).
It has nothing to do with the time value. (This is provided so that you can see the window easily.)

Note

The software interrupt number*1 is different according to product. For details about which interrupt number is assigned to which system call, refer to the MRxx Reference Manual, "Assemble Language Interface."

6.15.1 Extended Menus

This window has the following popup menus that can be brought up by right-clicking in the window.

Menu		Function
Start Marker		Move the start marker in the display area.
End Marker		Move the end marker in the display area.
Indicator		Move the indicator in the display area.
Adjust		Adjust range of start and end marker to full width of display area Adjust.
Expand		Expand scale factor of display.
Reduce		Reduce scale factor of display.
Trace Stop		Stop measuring.
Trace Restart		Restarts measuring.
Search...		Search for history of system calls.
Trace Range	After	Set measurement range condition to After.
	Break	Set measurement range condition to Break.
Value...		Set value.
Color...		Change display color.
Init Order...		Reset the task order on the display.
Toolbar display		Display toolbar.
Customize toolbar...		Open toolbar customize dialog box.
Allow Docking		Allow window docking.
Hide		Hide window.

6.15.2 Refer the Execution History of Task(MRxx Window)

You can reference the task execution history in the MR Trace window. You can reference the execution history statistical processing result in the MR Analyze window. These windows are available for a target program using our real time OS (MRxx).

6.15.2.1 Select the Trace Range

To measure the task execution history, the real time trace function is used. Click the "After" button (or, select Menu - [Trace Range] ->[After]) or "Break" button (or, select Menu - [Trace Range] -> [Break]) in the MR Trace window.

After	Stores the cycles of trace data after the trace point.
Break	Stores the cycles to the point at which the trace point is passed.

Execute the target program. Record the information required to know the task execution history in the trace memory.

ATTENTION

A trace point set in the Trace Point Setting dialog is disabled.

6.15.2.2 Stop the Task Execution History Measurement

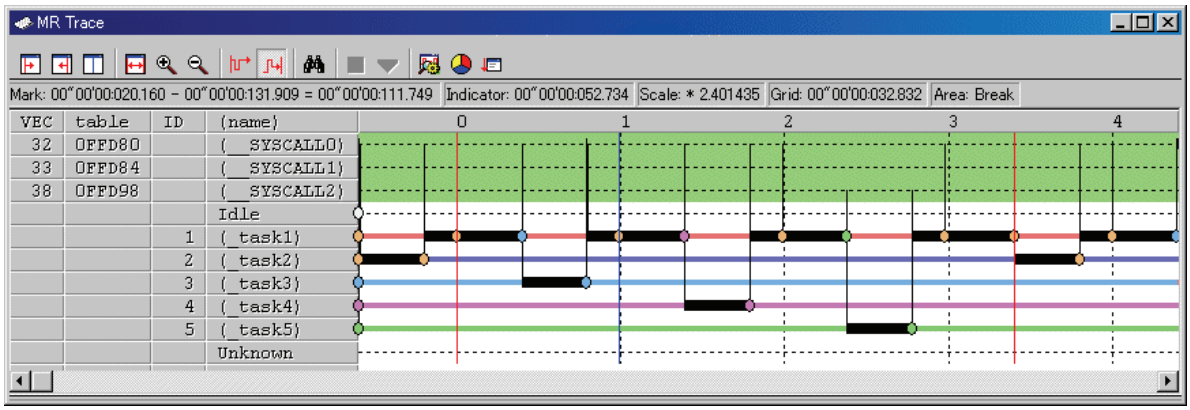
Click the "Stop" button in the Task Trace window. (Or, select Menu - [Trace Stop].) The measurement results so far are displayed in the MR Trace window.

6.15.2.3 Restart the Task Execution History Measurement

Click the "Restart" button in the Task Trace window. (Or, select Menu - [Trace Restart].) When restarting the trace measurement, all the measurement results so far are deleted.

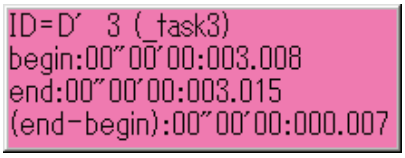
6.15.2.4 Refer the Execution History of Task

You can reference the task execution transition in the MR Trace window.

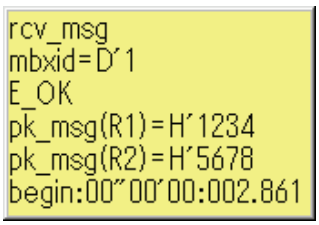


By moving the mouse to any information displayed in the window, the following window is opened, showing the detailed information.

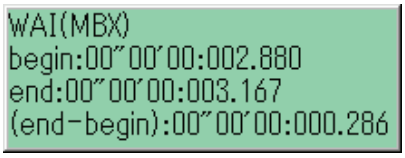
Interrupt handling or task execution history



System call issue history

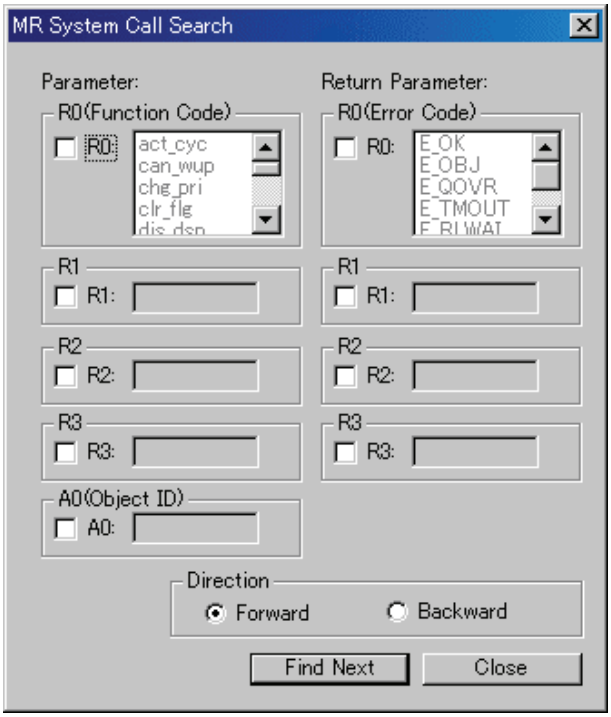


Task state transition history



6.15.2.4.1. Search the History of System Call Issue

Click the "Search" button in the tool bar. The Search dialog is opened. (Or, select Menu - [Search ...].)

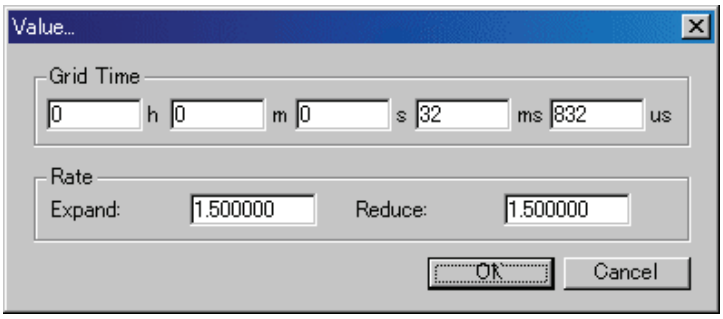


Specify the search condition.
With the function code (R0: Function Code) and error code (R0: Error Code), you can specify multiple values (OR condition). Other items are searched based on the AND condition.
Then, specify the search direction. The debugger searches the items in the direction specified in the dialog, using the position pointed by the indicator as the radix point.
When the debugger does not check all the search items, the subsequent system call issuance history in the search direction will be a search result. Click the Find Next button. The debugger searches the system call issuance history corresponding to the specified condition. The specified items are searched using the AND condition.
If the search condition is met, the indicator is moved to that point.

6.15.2.4.2. Change the display magnification

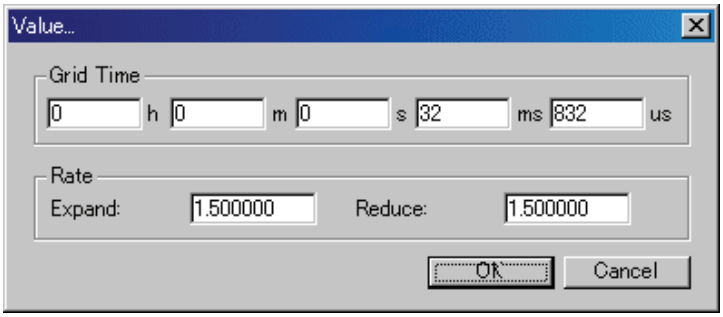
Click the "Expand" button or "Reduce" button in the tool bar. (Or, select Menu - [Expand] or [Reduce].)
The display is expanded or reduced using the left corner of the graph area as the radix point. By default, the display is expanded or reduced with display scale of 1.5.
The display scale appears in the "Scale:*" field in the status bar.

The default expansion/reduction scale is 1.5. To change the scale, select Menu - [Value ...].
The Value dialog is opened. Specify the display expansion/reduction scale.



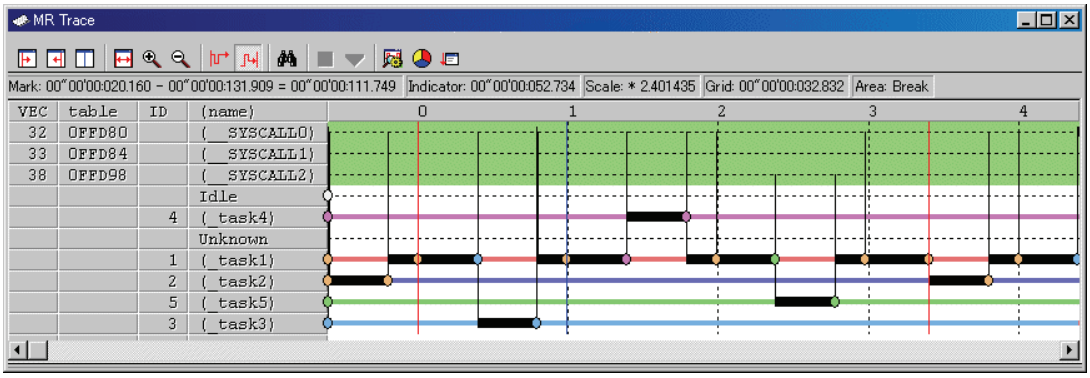
6.15.2.4.3. Change the grid line display interval

Select Menu - [Value ...]. The Value dialog is opened. Specify the display time interval.



6.15.2.4.4. Change the task display order

Drag the task/interrupt routine to be moved (the left portion of the graph) to the destination.



To initialize the display order, select Menu - [Init Order].

6.15.2.4.5. Display the specific task only

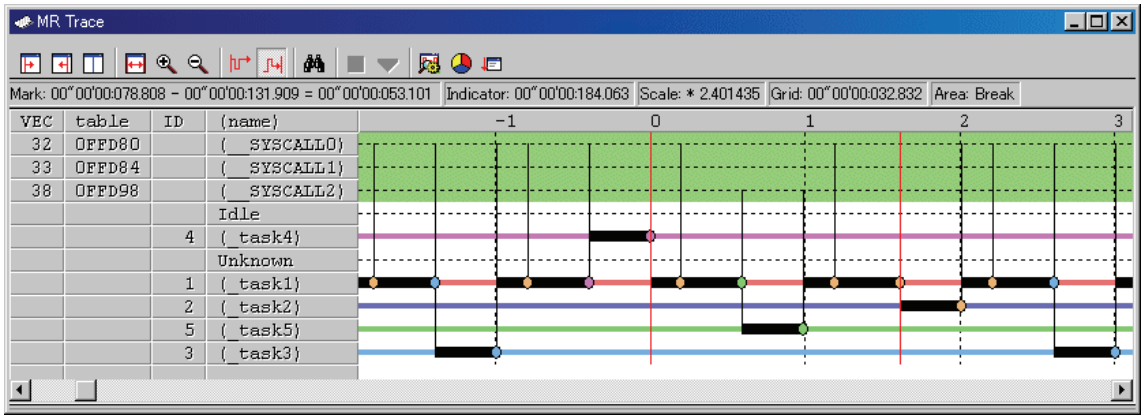
Click the task/interrupt routine to be hidden (the left portion of the graph). Every time you click, the setting is switched between "Display" and "Hide".

6.15.2.4.6. Change the display color

Select Menu - [Color...]. The Color dialog is opened.
Click the button corresponding to the desired item. The Color Setting dialog is opened. Change the display color in the dialog.

6.15.2.5 Measure the Execution Time of Task

You can measure the execution time between the markers by changing the positions of start marker and end marker in the MR Trace window.



Drag the start marker position and end marker position.
The time interval between the markers is displayed in the status bar.

Note

[Definition of time value in the MR Trace window]
The time value in the MR Trace window indicates the execution elapsed time which sets the program execution start point to 0 in all the cases.
On the contrary, a numeric value above the grid line (scale) in the MR Trace window is a relative value which sets the start marker to 0 (the grid interval is specified in the Value dialog), which has nothing to do with the time value. (It is provided so that you can see the window easily.)

6.15.2.5.1. Move the Marker

Each marker can be moved by dragging. When moving the mouse on the marker, the cursor shape changes. Then, start dragging.

The start marker moves into the window (left portion) by clicking the "Start Marker" button in the tool bar. (Or, select Menu - [Start Marker].)
The end marker moves into the window (right portion) by clicking the "End Marker" button. (Or, select Menu - [End Marker].)
The indicator moves into the window (center) by clicking the "Indicator" button. (Or, select Menu - [Indicator].)
The other markers can move only to the specified positions listed below.

- Position to which the interrupt processing or task execution transits
- Position to which the task state transits
- Position where the system call issuance history is displayed

6.16 MR Analyze Window

The MR Analyze window displays the result of the measurement data statistically analyzed within the range specified by the start marker and the end marker in the MR Trace window.
The debugger for R32C doesn't support this function.
The debugger for 740 doesn't support this function.

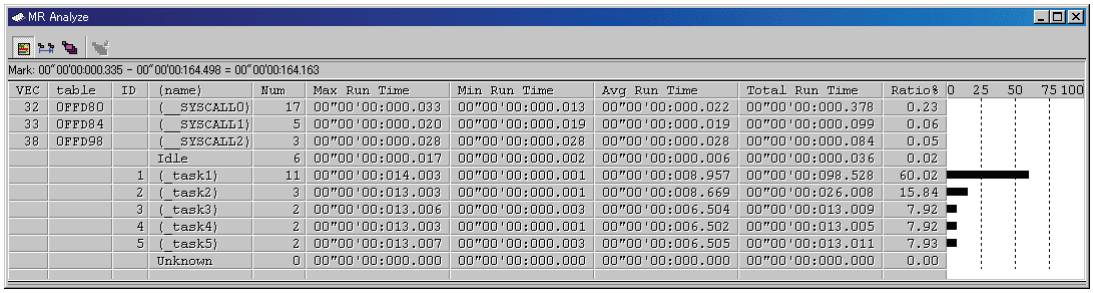
The MR Analyze window supports three display mode as below:

- CPU occupation state by interrupt/task
- Ready time by task
- List of system call issuance histories (You can extract and display the history based on the specific condition.)

The MR Analyze window functions together with the MR Trace window.
This window is available only when a target program using our real time OS (MRxx) is downloaded.

6.16.1 Configuration of CPU Occupancy Status Display Mode

The CPU occupation state display mode is used to display the CPU occupation time and ratio by interrupt/task.
The MR Trace window shows the statistical results within the range specified by the start marker and end marker.

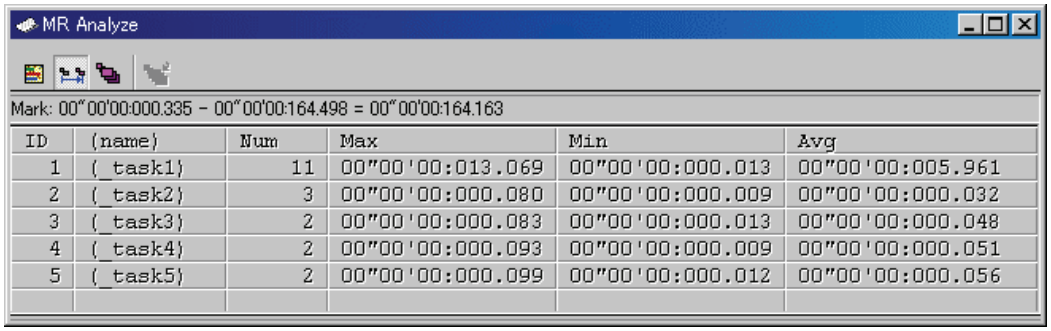


VEC	table	ID	(name)	Num	Max Run Time	Min Run Time	Avg Run Time	Total Run Time	Ratio% 0 25 50 75 100
32	OFFD80		(__SYSCALL0)	17	00"00'00:000.033	00"00'00:000.013	00"00'00:000.022	00"00'00:000.378	0.23
33	OFFD84		(__SYSCALL1)	5	00"00'00:000.020	00"00'00:000.019	00"00'00:000.019	00"00'00:000.099	0.06
38	OFFD98		(__SYSCALL2)	3	00"00'00:000.028	00"00'00:000.028	00"00'00:000.028	00"00'00:000.084	0.05
			Idle	6	00"00'00:000.017	00"00'00:000.002	00"00'00:000.006	00"00'00:000.036	0.02
		1	(_task1)	11	00"00'00:014.003	00"00'00:000.001	00"00'00:008.957	00"00'00:098.528	60.02
		2	(_task2)	3	00"00'00:013.003	00"00'00:000.001	00"00'00:008.669	00"00'00:026.008	15.84
		3	(_task3)	2	00"00'00:013.006	00"00'00:000.003	00"00'00:006.504	00"00'00:013.009	7.92
		4	(_task4)	2	00"00'00:013.003	00"00'00:000.001	00"00'00:006.502	00"00'00:013.005	7.92
		5	(_task5)	2	00"00'00:013.007	00"00'00:000.003	00"00'00:006.505	00"00'00:013.011	7.93
			Unknown	0	00"00'00:000.000	00"00'00:000.000	00"00'00:000.000	00"00'00:000.000	0.00

By clicking the maximum execution time/minimum execution time display area of each line, you can search interrupt to the clicked line or process history at the maximum/minimum execution time of the task.
The search result is pointed by the indicator which moves to the target position in the MR Trace window.

6.16.2 Configuration of Ready State Duration Display Mode

The ready state time display mode by task is used to display the results generated from statistical process of the time required from execution ready to transition to execution by task.
The statistical result is displayed within the range specified by the start marker and end marker in the MR Trace window.

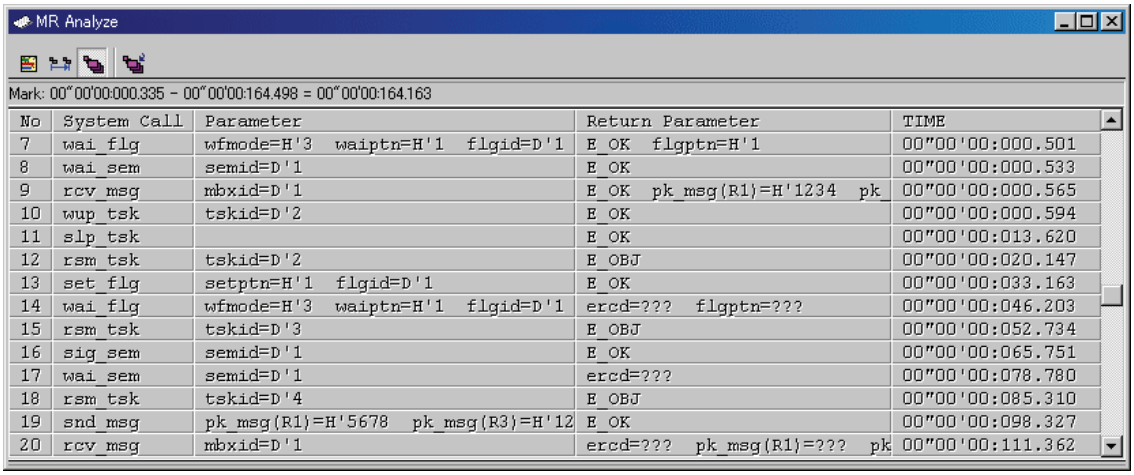


ID	(name)	Num	Max	Min	Avg
1	(_task1)	11	00"00'00:013.069	00"00'00:000.013	00"00'00:005.961
2	(_task2)	3	00"00'00:000.080	00"00'00:000.009	00"00'00:000.032
3	(_task3)	2	00"00'00:000.083	00"00'00:000.013	00"00'00:000.048
4	(_task4)	2	00"00'00:000.093	00"00'00:000.009	00"00'00:000.051
5	(_task5)	2	00"00'00:000.099	00"00'00:000.012	00"00'00:000.056

By clicking the maximum ready time/minimum ready time display area of the desired line, you can search the process history of the maximum ready time/minimum ready time of the task corresponding to the clicked line.
The search result is pointed by the indicator which moves to the target position in the MR Trace window.

6.16.3 Configuration of System Call History Display Mode

The system call issuance history list mode is used to list the system calls issued.
The system call issuance history is listed within the range specified by the start marker and end marker in the MR Trace window.
The number indicates a numeric value counted from the top system call within the measurable range.



No	System Call	Parameter	Return Parameter	TIME
7	wai_flg	wfmode=H'3 waiptn=H'1 flgid=D'1	E_OK flgptn=H'1	00"00'00:000.501
8	wai_sem	semid=D'1	E_OK	00"00'00:000.533
9	rcv_msg	mbxid=D'1	E_OK pk_msg(R1)=H'1234 pk_	00"00'00:000.565
10	wup_tsk	tskid=D'2	E_OK	00"00'00:000.594
11	slp_tsk		E_OK	00"00'00:013.620
12	rsm_tsk	tskid=D'2	E_OBJ	00"00'00:020.147
13	set_flg	setptn=H'1 flgid=D'1	E_OK	00"00'00:033.163
14	wai_flg	wfmode=H'3 waiptn=H'1 flgid=D'1	ercd=??? flgptn=???	00"00'00:046.203
15	rsm_tsk	tskid=D'3	E_OBJ	00"00'00:052.734
16	sig_sem	semid=D'1	E_OK	00"00'00:065.751
17	wai_sem	semid=D'1	ercd=???	00"00'00:078.780
18	rsm_tsk	tskid=D'4	E_OBJ	00"00'00:085.310
19	snd_msg	pk_msg(R1)=H'5678 pk_msg(R3)=H'12	E_OK	00"00'00:098.327
20	rcv_msg	mbxid=D'1	ercd=??? pk_msg(R1)=??? pk	00"00'00:111.362

By clicking the desired line, you can search the system call issuance history to the clicked line.
The search result is pointed by the indicator which moves to the target position in the MR Trace window.

6.16.4 Extended Menus

This window has the following popup menus that can be brought up by right-clicking in the window.

Menu	Function
Run Time	CPU occupancy status display mode.
Rdy->Run	Ready state duration display mode.
System Call	System call history display mode.
Pick Up System Call...	Extract specified system calls display mode.
Toolbar display	Display toolbar.
Customize toolbar...	Open toolbar customize dialog box.
Allow Docking	Allow window docking.
Hide	Hide window.

6.16.5 Analyze the Execution History of Task

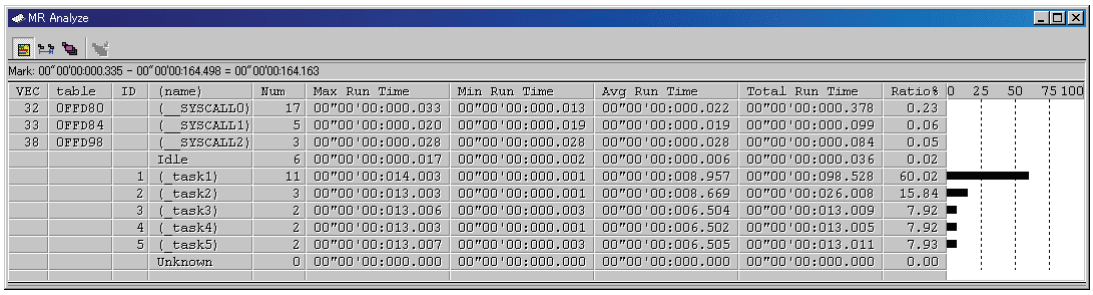
6.16.5.1 Analyze the Execution History of Task

You can reference the execution history statistical processing in the MR Analyze window. The MR Analyze window functions together with the MR Trace window. If the MR Trace window is not open, or the MR Trace window does not show any data, the MR Analyze window will not function.

The execution history statistical processing function allows you to reference the following topics.

6.16.5.1.1. Refer the CPU Occupation State

Click the Run Time button in the tool bar. (Or, select Menu - [Run Time].) The MR Analyze window changes to the CPU occupation state display mode.



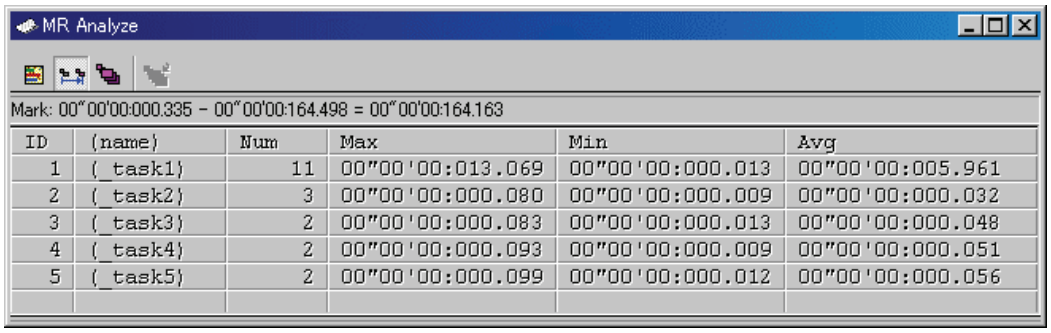
VEC	table	ID	(name)	Num	Max Run Time	Min Run Time	Avg Run Time	Total Run Time	Ratio%
32	OFFD80		(__SYSCALL0)	17	00"00'00:000.033	00"00'00:000.013	00"00'00:000.022	00"00'00:000.378	0.23
33	OFFD84		(__SYSCALL1)	5	00"00'00:000.020	00"00'00:000.019	00"00'00:000.019	00"00'00:000.099	0.06
38	OFFD98		(__SYSCALL2)	3	00"00'00:000.028	00"00'00:000.028	00"00'00:000.028	00"00'00:000.084	0.05
			Idle	6	00"00'00:000.017	00"00'00:000.002	00"00'00:000.006	00"00'00:000.036	0.02
		1	(_task1)	11	00"00'00:014.003	00"00'00:000.001	00"00'00:008.957	00"00'00:098.528	60.02
		2	(_task2)	3	00"00'00:013.003	00"00'00:000.001	00"00'00:008.669	00"00'00:026.008	15.84
		3	(_task3)	2	00"00'00:013.006	00"00'00:000.003	00"00'00:006.504	00"00'00:013.009	7.92
		4	(_task4)	2	00"00'00:013.003	00"00'00:000.001	00"00'00:006.502	00"00'00:013.005	7.92
		5	(_task5)	2	00"00'00:013.007	00"00'00:000.003	00"00'00:006.505	00"00'00:013.011	7.93
			Unknown	0	00"00'00:000.000	00"00'00:000.000	00"00'00:000.000	00"00'00:000.000	0.00

The window shows the CPU occupation time and ratio by interrupt processing and by task.
The data displayed is the statistical results for the range specified with the start marker and end marker in the MR Trace window.

By clicking the maximum execution time/minimum execution time display field of each line, you can search the processing history at the maximum execution time/minimum execution time of the task corresponding to the clicked line.
The search result is pointed by the indicator in the MR Trace window after the indicator moves to the destination position.

6.16.5.2 Refer the Ready Queue Time

Click the Ready->Run button in the tool bar. (Or, select Menu - [Rdy -> Run].)



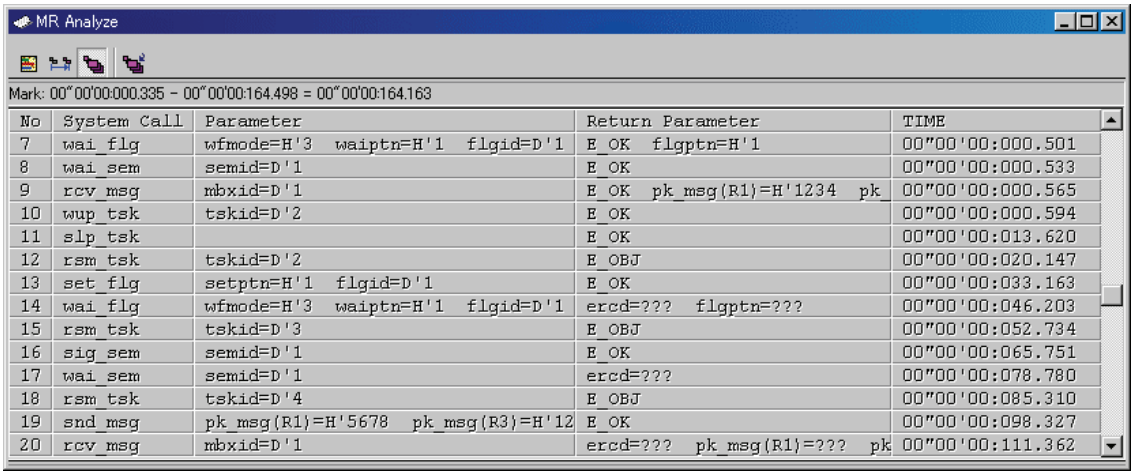
ID	(name)	Num	Max	Min	Avg
1	(_task1)	11	00"00'00:013.069	00"00'00:000.013	00"00'00:005.961
2	(_task2)	3	00"00'00:000.080	00"00'00:000.009	00"00'00:000.032
3	(_task3)	2	00"00'00:000.083	00"00'00:000.013	00"00'00:000.048
4	(_task4)	2	00"00'00:000.093	00"00'00:000.009	00"00'00:000.051
5	(_task5)	2	00"00'00:000.099	00"00'00:000.012	00"00'00:000.056

The time required from execution ready state to transition to execution state by task is processed statistically and displayed.
The data displayed is the statistical results of the range specified with the start marker and end marker in the MR Trace window.

By clicking the maximum ready time/minimum ready time display field of each line, you can search the processing history at the maximum ready time/minimum ready time of the task corresponding to the clicked line.
The search result is pointed by the indicator in the MR Trace window after the indicator moves to the destination position.

6.16.5.3 Refer the System Call Issuance History

Click the "System Call" button in the tool bar. (Or, select Menu - [System Call].)



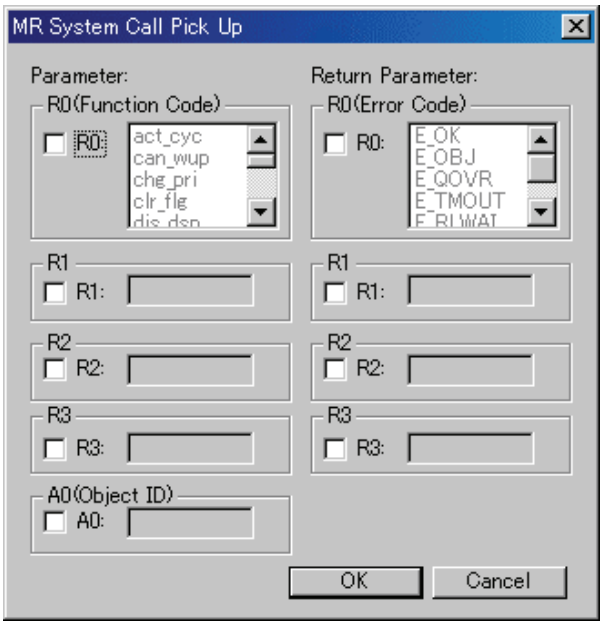
No	System Call	Parameter	Return Parameter	TIME
7	wai_flg	wfmode=H'3 waiptn=H'1 flgid=D'1	E_OK flgptn=H'1	00"00'00:000.501
8	wai_sem	semid=D'1	E_OK	00"00'00:000.533
9	rcv_msg	mbxid=D'1	E_OK pk_msg(R1)=H'1234 pk_	00"00'00:000.565
10	wup_tsk	tskid=D'2	E_OK	00"00'00:000.594
11	slp_tsk		E_OK	00"00'00:013.620
12	rsm_tsk	tskid=D'2	E_OBJ	00"00'00:020.147
13	set_flg	setptn=H'1 flgid=D'1	E_OK	00"00'00:033.163
14	wai_flg	wfmode=H'3 waiptn=H'1 flgid=D'1	ercd=??? flgptn=???	00"00'00:046.203
15	rsm_tsk	tskid=D'3	E_OBJ	00"00'00:052.734
16	sig_sem	semid=D'1	E_OK	00"00'00:065.751
17	wai_sem	semid=D'1	ercd=???	00"00'00:078.780
18	rsm_tsk	tskid=D'4	E_OBJ	00"00'00:085.310
19	snd_msg	pk_msg(R1)=H'5678 pk_msg(R3)=H'12	E_OK	00"00'00:098.327
20	rcv_msg	mbxid=D'1	ercd=??? pk_msg(R1)=??? pk	00"00'00:111.362

The issued system calls are listed in chronological order of system call.
The data displayed is the statistical results for the range specified with the start marker and end marker in the MR Trace window.

By clicking each line, you can search the system call issuance history corresponding to the clicked line.
The search result is pointed by the indicator in the MR Trace window after the indicator moves to the destination position.

6.16.5.3.1. Extract the Issuance History

Click the "Pick Up" button in the tool bar. (Or, select Menu - [Pick Up System Call...].)
The dialog shown below is opened. Specify the search condition of the system call to be extracted and displayed.

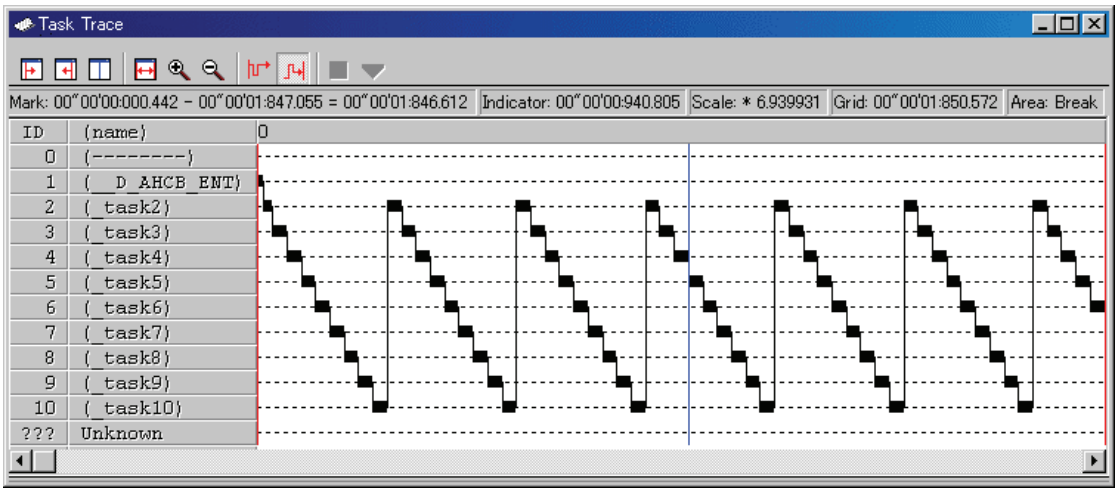


Extract the issuance history of the system call which meets the specified condition and display it.

6.17 Task Trace Window

The Task Trace window measures the task execution history of a program using the real time OS and display it graphically.
This window is available even when a target program using an OS other than our real time OS (MRxx) is downloaded.

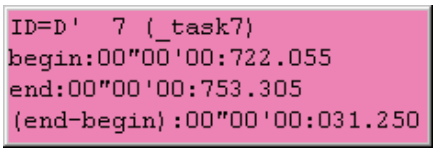
The debugger for R32C doesn't support this function.
The debugger for 740 doesn't support this function.



The content of each item is as follows.

Items	Contents
ID	Indicates a task ID number.
(name)	Indicates an interrupt routine name, task name, idle processing (display "idle"), and unknown name(displayed "unknown").

When moving the mouse to the information displayed in the window, the pop up window as below is opened, showing the detailed information.



The following information is displayed in the status bar.

- Time value at the start marker position
- Time value at the end marker position
- Time interval between the start marker and the end marker
- Time value at the indicator position
- Display scale
- Time width at grid line interval
- Measurement (trace) range

The grid lines are displayed using the start marker as the radix point.
The scale is displayed, using the time at which the start marker is positioned as 0, with the left (forward in time) set to "minus" and the right (backward in time) set to "plus".
The grid lines allow you to roughly understand the interrupt occurrence cycle and process time.
The interval time width of the displayed grid lines appears in the "Grid" area of the status bar.

The time value in the Task Trace window means the execution elapsed time using the program execution start time as 0 in all the cases.
On the contrary, the numeric value above the grid lines (scale) in the Task Trace window is a relative value using the start marker as 0 (the grid interval is specified in the Value dialog). It has nothing to do with the time value. (This is provided so that you can see the window easily.)

6.17.1 Extended Menus

This window has the following popup menus that can be brought up by right-clicking in the window.

Menu		Function
Start Marker		Move the start marker in the display area.
End Marker		Move the end marker in the display area.
Indicator		Move the indicator in the display area.
Adjust		Adjust range of start and end marker to full width of display area Adjust.
Expand		Increase scale factor of display.
Reduce		Decrease scale factor of display.
Trace Stop		Stop measuring.
Trace Restart		Restarts measuring.
Trace Range	After	Set measurement range condition to After.
	Break	Set measurement range condition to Break.
Value...		Set value.
Color...		Change display color.
RTOS...		Set target RTOS information.
Toolbar display		Displays the toolbar.
Customize toolbar...		Opens the toolbar customize dialog box.
Allow Docking		Allows the window docking.
Hide		Hides the window.

6.17.2 Refer the Execution History of Task(Taskxx Window)

You can reference the task execution history in the Task Trace window.
You can reference the execution history statistical processing result in the Task Analyze window.
These windows are also available for a target program using an OS other than our real time OS (MRxx).

6.17.2.1 Prepare the Measurement

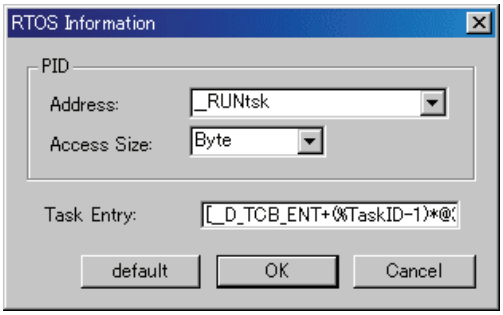
To measure the task execution history of the program using the real time OS, you must select the trace range in the Task Trace window and then execute the target program.

6.17.2.1.1. Set the Information of Realtime OS.

To use the Task Trace window, you must set the following information concerning the real time OS (the target real time OS) which are used by the downloaded program.

- Label name (address value) of the execution task ID storage area and its size
- Task start address expression

Open the Task Trace window. Select Menu - [View] -> [RTOS] -> [Task Trace].
When you select this menu at first time after starting PDxx, the RTOS Information dialog is opened before the Task Trace window is opened.



- When using our real time OS (MRxx)

1. Click the "default" button. The MRxx information is set.
2. Click the OK button. The Task Trace window is opened.

- When using a real time OS other than MRxx

1. Specify the label (address is also available) of the execution task ID storage area in the PID Address field ; specify the size of the execution task ID storage area in the Size list box.
   If this information is not set correctly, you cannot use the Task Trace window.
2. Specify the task start address expression in the Task Entry field.
   Describe the expression in the format in accordance with the description rules.
   Use a macro variable [% TaskID] in the address where the task ID number is supposed to assign.
   If this information is not set correctly, the task name is not displayed in the Task Trace window.
3. Click the OK button. The Task Trace window is opened.

On debuger for 740, simply by clicking the default button, the OSEK OS information is set.
Once the real time OS information is set in this dialog, the information becomes effective from the next time.
To change the setting data, select [RTOS...] from popup menu by right-clicking on the window. The RTOS Information dialog is re-opened.

ATTENTION

When specifying WORD in the access size when performing PID setting in the RTOS Information dialog, you must observe the following limits. (If these conditions are not met, the system does not operate normally.)

- The PID information storage area is allocated to an even address.
- The PID information storage area is allocated to an area accessed with 16-bit bus width.

6.17.2.1.2. Select the Trace Range

The real time trace function is used for task execution history measurement.
Click the After button (or select [After] from popup menu by right-clicking on the window) or Break button (or or select [Break] from popup menu) in the Task Trace window.

After	Records a task execution history until the trace memory is filled with recorded data.
Break	Records a task execution history (as much as trace memory available) until before the target program stops.

Only an specific cycle required to know the task execution history is recorded in the trace memory.

ATTENTION

A trace point set in the Trace Point Setting dialog is disabled.

6.17.2.1.3. Start the Target Program

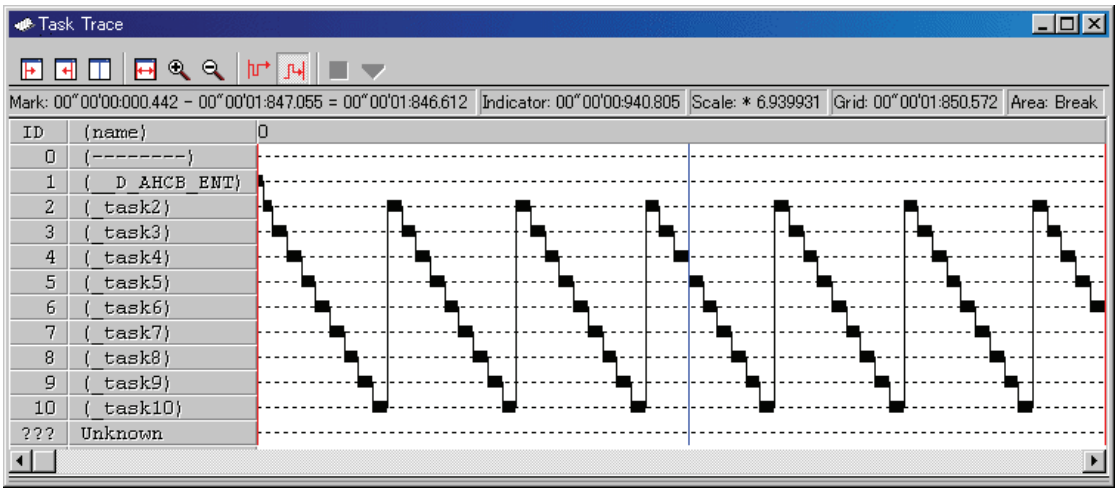
Execute the target program. Record the information required to know the task execution history in the trace memory.
When you select After for the trace range, the execution history is displayed in the Task Trace window immediately after the trace memory is filled or immediately after the target program stops.
When you select Break for the trace range, the execution history is displayed in the Task Trace window immediately after the target program stops.

You can stop task execution history measurement.
To do this, click the Stop button in the Task Trace window. (Or, select [Trace Stop] from popup menu by right-clicking on the window.)

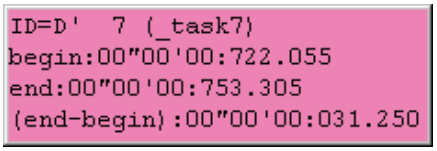
To restart task execution history measurement, click the Restart button in the Task Trace window. (Or, select [Trace Restart] from popup menu by right-clicking on the window.)

6.17.2.2 Refer the Execution History of Task

You can reference task trace transition in the Task Trace window.



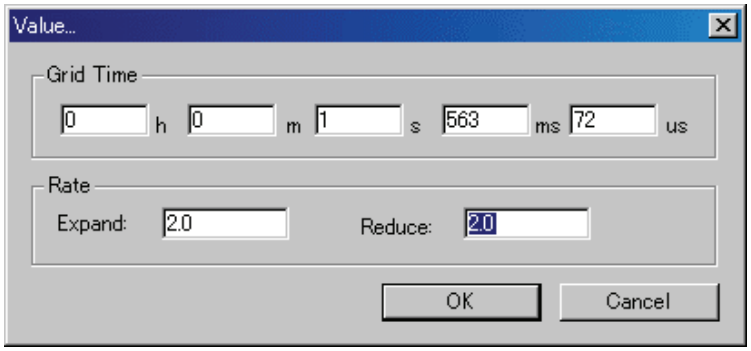
By moving the mouse to the information displayed in the window, a window of the following example is opened, showing the detailed information.



6.17.2.2.1. Change the display magnification

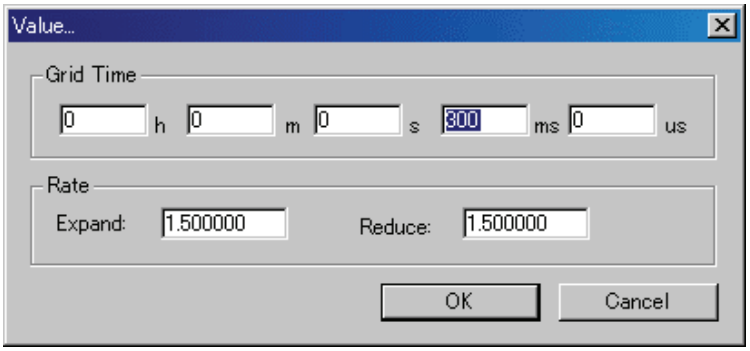
Click the Expand button or Reduce button in the tool bar. (Or, select [Expand] or [Reduce] from popup menu by right-clicking on the window.)
The display is expanded or reduced using the left corner of the graph area as the radix point. By default, the display is expanded or reduced with display scale of 1.5.
The display scale appears in the "Scale:*" field in the status bar.

The default expansion/reduction scale is 1.5. To change the scale, select [Value ...] from popup menu by right-clicking on the window. The Value dialog is opened. Specify the display expansion/reduction scale.



6.17.2.2.2. Change the grid line display interval

Select [Value ...] from popup menu by right-clicking on the window. The Value dialog is opened. Specify the time interval in the display.

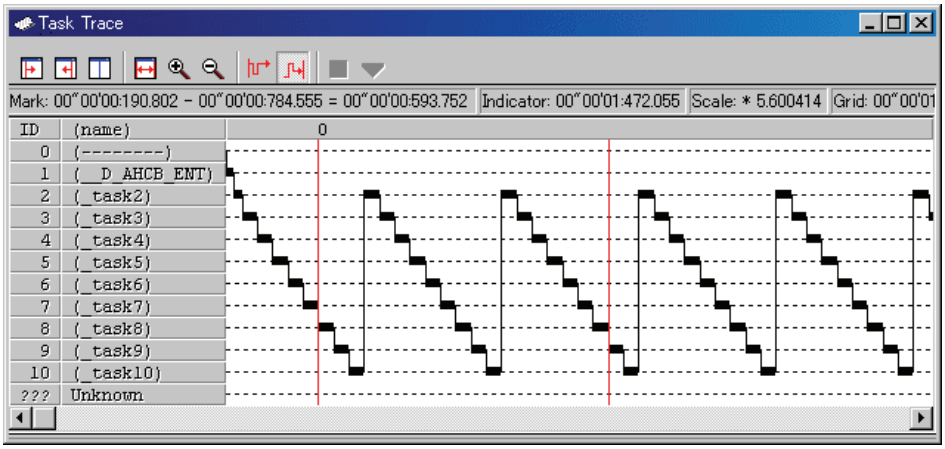


6.17.2.2.3. Change the task display order

Select [Color ...] from popup menu by right-clicking on the window. The Color dialog is opened.
Click the button corresponding to the desired item. The Color Setting dialog is opened. Change the display color in the dialog.

6.17.2.2.4. Measure the Execution Time of Task

By changing the start marker position and end marker position in the Task Trace window, you can measure the execution time between the markers.



Drag the start marker position and end marker position.
The time interval between the markers is displayed in the status bar.

Note

- Definition of time value in the Task Trace window
  The time value in the Task Trace window indicates the execution elapsed time which sets the program execution start point to 0 in all the cases.
  On the contrary, a numeric value above the grid line (scale) in the Task Trace window is a relative value which sets the start marker to 0 (the grid interval is specified in the Value dialog), which has nothing to do with the time value. (It is provided so that you can see the window easily.)

6.17.2.2.5. Move the Marker

Each marker can be moved by dragging. When moving the mouse on the marker, the cursor shape changes. Then, start dragging.

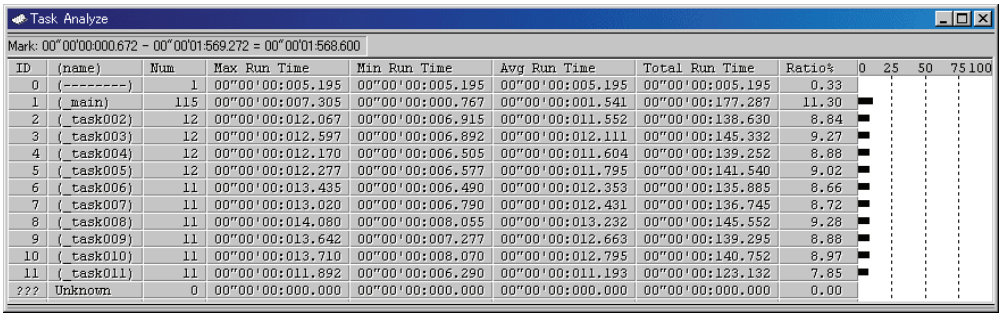
The start marker moves into the window (left portion) by clicking the Start Marker button in the tool bar. (Or, select [Start Marker] from popup menu by right-clicking on the window.)
The end marker moves into the window (right portion) by clicking the End Marker button. (Or, select [End Marker] from popup menu.)
The indicator moves into the window (center) by clicking the "Indicator" button. (Or, select [Indicator] from popup menu.)
Note that each marker can move only to the point where an event is established.

6.18 Task Analyze Window

The Task Analyze window displays the result of the measurement data statistically analyzed within the range specified by the start marker and the end marker in the Task Trace window.
The Task Analyze window displays the CPU occupation state.
The Task Analyze window functions together with the Task Trace window.

The debugger for R32C doesn't support this function.
The debugger for 740 doesn't support this function.

This window is available even when a target program using an OS other than our real time OS (MRxx) is downloaded.
The CPU occupation state display mode is used to display the CPU occupation time and ratio by task.
This mode shows the statistical result within the range specified by the start marker and end marker in the Task Trace window.



By clicking the maximum execution time/minimum execution time display area of each line, you can search process history of the task for the clicked line at the maximum/minimum execution time.
The search result is pointed by the indicator which moves to the target position in the Task Trace window.

6.18.1 Extended Menus

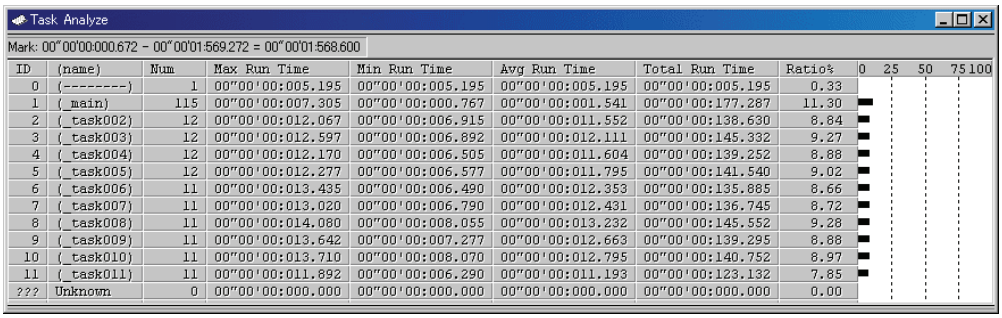
This window has the following popup menus that can be brought up by right-clicking in the window.

Menu	Function
Allow Docking	Allows the window docking.
Hide	Hides the window.

6.18.2 Analyze the Execution History of Task

You can reference the execution history statistical processing in the Task Analyze window. This window shows the CPU occupation time and ratio by task.

The Task Analyze window functions together with the Task Trace window. If the Task Trace window is not open, or the Task Trace window does not show any data, the Task Analyze window will not function.



The displayed data is the statistical results of the range specified by the start marker and the end marker in the Task Trace window.

By clicking the maximum ready time/minimum ready time display field of each line, you can search the processing history at the maximum ready time/minimum ready time of the task corresponding to the clicked line.

The search result is pointed by the indicator in the Task Trace window after the indicator moves to the destination position.

7. Table of Script Commands

The following script commands are prepared.
The commands with yellow color displaying can be executed at run time.
The command to which "*" adheres behind is not supported according to the product.

7.1 Table of Script Commands (classified by function)

7.1.1 Execution Commands

Command Name	Short Name	Contents
Go	G	Program execution with breakpoints
GoFree	GF	Free run program execution
GoProgramBreak*	GPB	Run target program with software break point
GoBreakAt*	GBA	Run target program with software break point
Stop	-	Stops program execution
Status	-	Checks the operating status of the MCU
Step	S	Halts for user input until the specified time has elapsed
StepInstruction	SI	Step execution of instructions
OverStep	O	Overstep execution of source lines
OverStepInstruaction	OI	Overstep execution of instructions
Return	RET	Executes a source line return
ReturnInstruction	RETI	Executes an instruction return
Reset	-	Resets the target MCU

7.1.2 File Operation Commands

Command Name	Short Name	Contents
Load	L	Downloads the target program
LoadHex	LH	Downloads an Intel HEX-format file
LoadMot*	LM	Downloads a Motorola S-format file
LoadSymbol	LS	Loads source line/ASM symbol information
LoadIeee*	LI	Downloads IEEE-695 absolute-format files
Reload	-	Re-downloads the target program
UploadHex	UH	Outputs data to an Intel HEX-format file
UploadMot*	UM	Outputs data to a Motorola S-format file

7.1.3 Register Operation Commands

Command Name	Short Name	Contents
Register	R	Checks and sets a register value

7.1.4 Memory Operation Commands

Command Name	Short Name	Contents
DumpByte	DB	Displays the contents of memory (in 1-byte units)
DumpWord*	DW	Displays the contents of memory (in 2-byte units)
DumpLword*	DL	Displays the contents of memory (in 4-byte units)
SetMemoryByte	MB	Checks and changes memory contents (in 1-byte units)
SetMemoryWord*	MW	Checks and changes memory contents (in 2-byte units)
SetMemoryLword*	ML	Checks and changes memory contents (in 4-byte units)
FillByte	FB	Fills a memory block with the specified data (in 1-byte units)
FillWord*	FW	Fills a memory block with the specified data (in 2-byte units)
FillLword*	FL	Fills a memory block with the specified data (in 4-byte units)
Move	-	Moves memory blocks
MoveWord*	MOVEW	Moves memory blocks(in 2-byte units)
MoveLword*	MOVEL	Moves memory blocks(in 4-byte units)

7.1.5 Assemble/Disassemble Commands

Command Name	Short Name	Contents
Assemble	A	Line-by-line assembly
DisAssemble	DA	Disassembles memory contents line by line
Module	MOD	Displays modules names
Scope	-	Sets and checks the effective local symbol scope
Section	SEC	Checks section information
Bit*	-	Checks and sets bit symbols
Symbol	SYM	Checks assembler symbols
Label	-	Checks assembler labels
Express	EXP	Displays an assembler expression

7.1.6 Software Break Setting Commands

Command Name	Short Name	Contents
SoftwareBreak	SB	Sets and checks software breaks
SoftwareBreakClear	SBC	Clears software breaks
SoftwareBreakClearAll	SBCA	Clears all software breaks
SoftwareBreakDisable	SBD	Disables software breakpoints
SoftwareBreakDisableAll	SBDA	Disables all software breaks
SoftwareBreakEnable	SBE	Enables software breakpoints
SoftwareBreakEnableAll	SBEA	Enables all software breaks
BreakAt	-	Sets a software breakpoint by specifying a line No.
BreakIn	-	Sets a software breakpoint by specifying a function

7.1.7 Hardware Break Setting Commands

Command Name	Short Name	Contents
HardwareBreak	HB	Sets and checks a hardware break
HardwareBreakClear	HBC	Clears hardware breaks
HardwareBreakClearAll	HBCA	Clears all hardware breaks
BreakMode	BM	Sets and checks hardware break mode

7.1.8 Real-time Trace Commands

Command Name	Short Name	Contents
TracePoint*	TP	Sets and checks a trace points
TraceData*	TD	Realtime trace data display
TraceList*	TL	Displays disassembled realtime trace data

7.1.9 Coverage Measurement Commands

Command Name	Short Name	Contents
Coverage	CV	Specifies and displays coverage measurement

7.1.10 Stack Utilization Monitor Command

Command Name	Short Name	Contents
StackMonitor	SM	Sets and checks stack utilization measurement

7.1.11 Cycle Count Monitor Command

Command Name	Short Name	Contents
Cycle	CY	Sets and checks cycle counting

7.1.12 Script/Log File Commands

Command Name	Short Name	Contents
Script	-	Opens and executes a script file
Exit	-	Exits the script file
Wait	-	Waits for an event to occur before command input
Pause	-	Waits for user input
Sleep	-	Halts for user input until the specified time has elapsed
Logon	-	Outputs the screen display to a log file
Logoff	-	Stops the output of the screen display to a log file
Exec	-	Executes external application

7.1.13 Program Display Commands

Command Name	Short Name	Contents
Func	-	Checks function names and displays the contents of functions
Up*	-	Displays the calling function
Down*	-	Displays a called function
Where*	-	Displays a function call status
Path	-	Sets and checks the search path
AddPath	-	Adds the search path
File	-	Checks a filename and displays the contents of that file

7.1.14 Map Commands

Command Name	Short Name	Contents
Map*	-	Checks and sets mapping data

7.1.15 C Language Debugging Commands

Command Name	Short Name	Contents
Print	-	Check value of specified C variable expression
Set	-	Set specified data in specified C variable expression

7.1.16 Real-time OS Command

Command Name	Short Name	Contents
MR*	-	Displays status of realtime OS (MRxx)

7.1.17 Utility Commands

Command Name	Short Name	Contents
Radix	-	Sets and checks the radix for numerical input
Alias	-	Specifies and checks command alias definitions
UnAlias	-	Cancels the alias defined for a command
UnAliasAll	-	Cancels all aliases defined for commands
Version	VER	Displays the version No.
Date	-	Displays the date
Echo	-	Displays messages
CD	-	Window open

7.2 Table of Script Commands (alphabetical order)

Command Name	Short Name	Contents
AddPath	-	Adds the search path
Alias	-	Specifies and checks command alias definitions
Assemble	A	Line-by-line assembly
Bit*	-	Checks and sets bit symbols
BreakAt	-	Sets a software breakpoint by specifying a line No.
BreakIn	-	Sets a software breakpoint by specifying a function
BreakMode	BM	Sets and checks hardware break mode
CD	-	Specifies and checks the current directory
Coverage	CV	Specifies and displays coverage measurement
Cycle	CY	Sets and checks cycle counting
Date	-	Displays the date
DisAssemble	DA	Disassembles memory contents line by line
Down*	-	Displays a called function
DumpByte	DB	Displays the contents of memory (in 1-byte units)
DumpLword*	DL	Displays the contents of memory (in 4-byte units)
DumpWord*	DW	Displays the contents of memory (in 2-byte units)
Echo	-	Displays messages
Exec	-	Executes external application
Exit	-	Exits the script file
Express	EXP	Displays an assembler expression
File	-	Checks a filename and displays the contents of that file
FillByte	FB	Fills a memory block with the specified data (in 1-byte units)
FillLword*	FL	Fills a memory block with the specified data (in 4-byte units)
FillWord*	FW	Fills a memory block with the specified data (in 2-byte units)
Func	-	Checks function names and displays the contents of functions
Go	G	Program execution with breakpoints
GoBreakAt*	GBA	Run target program with software break point
GoFree	GF	Free run program execution
GoProgramBreak*	GPB	Run target program with software break point
HardwareBreak	HB	Sets and checks a hardware break
HardwareBreakClear	HBC	Clears hardware breaks
HardwareBreakClearAll	HBCA	Clears all hardware breaks
Label	-	Checks assembler labels
Load	L	Downloads the target program
LoadHex	LH	Downloads an Intel HEX-format file
LoadIeee*	LI	Downloads IEEE-695 absolute-format files
LoadMot*	LM	Downloads a Motorola S-format file
LoadSymbol	LS	Loads source line/ASM symbol information
Logoff	-	Stops the output of the screen display to a log file
Logon	-	Outputs the screen display to a log file
Map*	-	Checks and sets mapping data
Module	MOD	Displays modules names
Move	-	Moves memory blocks
MoveLword*	MOVEL	Moves memory blocks(in 4-byte units)
MoveWord*	MOVEW	Moves memory blocks(in 2-byte units)

MR*	-	Displays status of realtime OS (MRxx)
OverStep	O	Overstep execution of source lines
OverStepInstruaction	OI	Overstep execution of instructions
Path	-	Sets and checks the search path
Pause	-	Waits for user input
Print	-	Check value of specified C variable expression.
Radix	-	Sets and checks the radix for numerical input
Register	R	Checks and sets a register value
Reload	-	Re-downloads the target program
Reset	-	Resets the target MCU
Return	RET	Executes a source line return
ReturnInstruction	RETI	Executes an instruction return
Scope	-	Sets and checks the effective local symbol scope
Script	-	Opens and executes a script file
Section	SEC	Checks section information
Set	-	Set specified data in specified C variable expression
SetMemoryByte	MB	Checks and changes memory contents (in 1-byte units)
SetMemoryLword*	ML	Checks and changes memory contents (in 4-byte units)
SetMemoryWord*	MW	Checks and changes memory contents (in 2-byte units)
Sleep	-	Halts for user input until the specified time has elapsed
SoftwareBreak	SB	Sets and checks software breaks
SoftwareBreakClear	SBC	Clears software breaks
SoftwareBreakClearAll	SBCA	Clears software breaks
SoftwareBreakDisable	SBD	Disables software breakpoints
SoftwareBreakDisableAll	SBDA	Disables all software breaks
SoftwareBreakEnable	SBE	Enables software breakpoints
SoftwareBreakEnableAll	SBEA	Enables all software breaks
StackMonitor	SM	Sets and checks stack utilization measurement
Status	-	Checks the operating status of the MCU
Step	S	Step execution of source line
StepInstruction	SI	Step execution of instructions
Stop	-	Stops program execution
Symbol	SYM	Checks assembler symbols
TraceData*	TD	Realtime trace data display
TraceList*	TL	Displays disassembled realtime trace data
TracePoint*	TP	Sets and checks a trace points
UnAlias	-	Cancels the alias defined for a command
UnAliasAll	-	Cancels all aliases defined for commands
Up*	-	Displays the calling function
UploadHex	UH	Outputs data to an Intel HEX-format file
UploadMot*	UM	Outputs data to a Motorola S-format file
Version	VER	Displays the version No.
Wait	-	Waits for an event to occur before command input
Where*	-	Displays a function call status

8. Writing Script Files

This debugger allows you to run script files in a Script Window. The script file contains the controls necessary for automatically executing the script commands.

8.1 Structural Elements of a Script File

You can include the following in script files:

- Script commands
- Assign statements
- Conditional statements (if, else, endi)
  Program execution branches to the statement(s) to be executed according to the result of the conditional expression.
- Loop statements (while, endw)
  A block of one or more statements is repeatedly executed according to the expression.
- break statement
  Exits from the innermost loop.
- Comment statements
  You can include comments in a script file. The comment statements are ignored when the script commands are executed.

Specify only one statement on each line of the script file. You cannot specify more than one statement on a line, or write statements that span two or more lines.

Notes

- You cannot include comments on the same lines as script commands.
- You can nest script files up to five levels.
- You can nest if statements and while statements up to 32 levels.
- If statements must be paired with endi statements, and while statements with endw statements in each script file.
- Expressions included in script files are evaluated as unsigned types. Therefore, operation cannot be guaranteed if you use negative values for comparison in if or while statements.
- You can specify up to 4096 characters per line. An error occurs if a line exceeds this number of characters.
- When a script file containing illegal commands is automatically executed (when you select [Option] -> [Script]-> [Run] from the Script Window menu after opening a script file, or click the button in the Script Window), execution of the script file continues even after the error is detected, except when the script line itself cannot be read. If an error is detected and the script file continues to be executed, operation after detection of the error cannot be guaranteed. Reliability cannot therefore be placed on the results of execution after an error has been detected.

8.1.1 Script Command

You can use the same script commands that you enter in the Script Window. You can also call script files from within other script files (nesting up to 10 levels).

8.1.2 Assign Statement

Assign statement s define and initialize macro variables and assign values. The following shows the format to be used.

```
%macro-variable = expression
```

- You can use alphanumerics and the underscore (_) in macro variable names. However , you cannot use a numeric to start a macro variable name.
- You can specify any expression of which the value is an integer between 0h and FFFFFFFFh to be assigned in a macro variable. If you specify a negative number, it is processed as twos complement.
- You can use macro variables within the expression.
- Always precede macro variables with the "%" sign.

8.1.3 Conditional Statement

In a conditional statement, different statements are executed according to whether the condition is true or false. The following shows the format to be used.

```
if ( expression )
    statement 1
 else
    statement 2
 endi
```

- If the expression is t rue (other than 0), statement 1 is executed. If false, (0), statement 2 is executed.
- You can omit the else statement. If omitted and the expression is false, execution jumps to the line after the endi statement.
- if statements can be nested (up to 32 levels).

8.1.4 Loop Statement(while,endw) and Break Statement

In loop statements, execution of a group of statements is repeated while the expression is true. The following shows the format to be used.

```
while ( expression )
    statement
 endw
```

- If the expression is t rue, the group of statements is repeated. If false, the loop is exited (and the statement following the endw statement is executed).
- You can nest while statements up to 32 levels.
- Use the break statement to forcibly exit a while loop. If while statements are nested, break exits from the inner most loop.

8.1.5 Comment statements

You can include comments in a script file. Use the following format.

```
        ;character string
```

- Write the statement after a semicolon (;). You can include only spaces and tabs in front of the semicolon
- Lines with comment statements are ignored when the script file is executed.

8.2 Writing Expressions

This debugger allows you to use expressions for specifying addresses, data, and number of passes, etc. The following shows example commands using expressions.

```
>DumpByte TABLE1
>DumpByte TABLE1+20
```

You can use the following elements in expressions:

- Constants
- Symbols and labels
- Macro variables
- Register variables
- Memory variables
- Line Nos.
- Character constants
- Operators

8.2.1 Constants

You can use binary, octal, decimal, or hexadecimals. The prefix or suffix symbol attached to the numerical value indicates which radix is used.
The debugger for M32C and M16C/R8C and 740

	Hexadecimal	Decimal	Octal	Binary *
Prefix	0x,0X	@	None	%
Suffix	h,H	None	o,O	b,B
Examples	0xAB24 AB24h	@1234	1234o	%10010 10010b

*You can only specify % when the predetermined radix is hexadecimal.

- If you are inputting a radix that matches the predetermined radix, you can omit the symbol that indicates the radix (excluding binary).
- Use the RADIX command to set the predetermined value of a radix. However, in the cases shown below, the radix is fixed regardless of what you specify in a RADIX command.

Type	Radix
Address	Hex
Line No. No. of executions No. of passes	Dec

8.2.2 Symbols and labels

You can include symbols and labels defined in your target program, or symbols and labels defined using the Assemble command.

- You can include alphanumerics, the underscore (_), period (.), and question mark (?) in symbols and labels. However, do not start with a numeric.
- Symbols and labels can consist of up to 255 characters.
- Uppercase and lowercase letters are unique.

Product Name	Notes
The debugger for M32R, The debugger for M32C, The debugger for M16C/R8C,	• You cannot include the assembler structured instructions, pseudo instructions, macro instructions, operation code, or reserved words (.SECTION, .BYTE, switch, if, etc.). • You cannot use strings that start with two periods (..) for symbols or labels.
The debugger for 740	• You cannot use the register name.(A,X,Y,S,PC,PS,P) • You cannot include the assembler structured instructions, pseudo instructions, macro instructions, operation code, or reserved words (.SECTION, .BYTE, switch, if, etc.). • You cannot use strings that start with two periods (..) for symbols or labels. .D0 to .D65535, .F0 to .F65535, .I0 to .I56635, .S0 to .S65535, ..0 to ..65535, ??0 to ??65535

8.2.2.1 Local label symbol and scope

This debugger supports both global label symbols, which can be referenced from the whole program area, and local label symbols, which can only be referenced within the file in which they are declared. The effective range of local label symbols is known as the scope, which is measured in units of object files. The scope is switched in this debugger in the following circumstances:

- When a command is entered
  The object file that includes the address indicated by the program counter becomes the current scope. When the SCOPE command is used to set the scope, the specified scope is the active scope.
- During command execution
  The current scope automatically switches depending on the program address being handled by the command.

8.2.2.2 Priority levels of labels and symbols

The conversion of values to labels or symbols, and vice versa, is subject to the following levels of priority:

- Conversion of address values

1. Local labels
2. Global labels
3. Local symbols
4. Global symbols
5. Local labels outside scope
6. Local symbols outside scope

- Conversion of data values

1. Local symbols
2. Global symbols
3. Local labels
4. Global labels
5. Local symbols outside scope
6. Local labels outside scope

- Conversion of bit values

1. Local bit symbols
2. Global bit symbols
3. Local bit symbols outside scope

8.2.3 Macro Variables

Macro variables are defined by assign statements in the script file. See Section "8.1.2 Assign Statement" in the Reference part for details. Precede variables with '%' for use as macro variables.

- You can specify alphanumerics and/or the underbar (_) in the variable name following the percent sign (%). However , do not star t the names with a numeric.
- You cannot use the names of registers as variable names.
- Uppercase and lowercase letters are differentiated in variable names.
- You can define a maximum of 32 macro variables. Once defined, a macro variable remains valid until you quit the debugger.

Macro variables are useful for specifying the number of iterations of the while statement.

8.2.4 Register variables

Register variables are used for using the values of registers in an expression. Precede the name of the register with '%' to use it as a register variable. Use the following format. (The debugger for 740 can use '_' instead of '%'.)

Product Name	Register name
The debugger for M32C	PC, USP, ISP, INTB, FLB, SVF, SVP, VCT, DMD0,DMD1, DCT0, DCT1, DRC0, DRC1, DMA0,DMA1, DCA0, DCA1, DRA0, DRA1, 0R0, 0R1, 0R2, 0R3, 0A0, 0A1, 0FB, 0SB <- Bank 0 Register 1R0, 1R1, 1R2, 1R3, 1A0, 1A1, 1FB, 1SB <- Bank 1 Register
The debugger for M16C/R8C	PC, USP, ISP, SB, INTB, FLG 0R0, 0R1, 0R2, 0R3, 0A0, 0A1, 0FB <- Bank 0 Register 1R0, 1R1, 1R2, 1R3, 1A0, 1A1, 1FB <- Bank 1 Register
The debugger for 740	PC, A, X, Y, S, PS

Uppercase and lowercase letters are not unique in register names. You can specify either.

8.2.5 Memory variables

Use memory variables to use memory values in expressions. The format is as follows:
[Address].data-size

- You can specify expressions in addresses (you can also specify memory variables).
- The data size is specified as shown in the following table. (The debugger for 740 doesn't support four byte length.)

data Length	Debugger	Specification
1 Byte	All	B or b
2 Bytes	The debugger for M32R	H or h
	Other	W or w
4 bytes	The debugger for M32R	W or w
	The debugger for M32R, M16C/R8C	L or l

Example: Referencing the contents of memory at address 8000h in 2 bytes
[0x8000].W

- The default data size is word, if not specified.

8.2.6 Line Nos.

These are source file line Nos. The format for line Nos. is as follows:

```
#line_no
#line_no."source file name"
```

- Specify line Nos. in decimal.
- You can only specify line Nos. in which software breaks can be set. You cannot specify lines in which no assembler instructions have been generated, including comment lines and blank lines.
- If you omit the name of the source file, the line Nos. apply to the source file displayed in active Editor(Source) Window.
- Include the file attribute in the name of the source file.
- Do not include any spaces between the line No. and name of the source file.

8.2.7 Character constants

The specified character or character string is converted into ASCII code and processed as a constant.

- Enclose characters in single quote marks.
- Enclose character strings in double quote marks.
- The character string must consist of one or two characters (16 bits max.). If more than two characters are specified, the last two characters of the string are processed. For example, "ABCD" would be processed as "CD", or value 4344h.

8.2.8 Operators

The table below lists the operators that you can use in expressions.

- The priority of operators is indicated by the level, level 1 being the highest and level 8 the lowest. If two or more operators have the same level of priority, they are evaluated in order from the left of the expression.

Operator	Function	Priority level
( )	Brackets	level 1
+, -, ~	Monadic positive, monadic negative, monadic logical NOT	level 2
*, /	Dyadic multiply, dyadic divide	level 3
+, -	Dyadic add, dyadic subtract	level 4
>>,	Right shift, left shift	level 5
&	Dyadic logical AND	level 6
|, ^	Dyadic logical OR, dyadic exclusive OR	level 7
<, <=, >, >=, ==, !=	Dyadic comparison	level 8

9. I/O Script

Settings of virtual port inputs or virtual interrupts can be written to a file in script form.
This script is called the "I/O script." Also, the files that contain a description of I/O scripts are called the "I/O script file."
Using I/O scripts, you can set virtual port inputs and virtual interrupts in a more flexible manner than can be set from the I/O Timing Setting Window. For example, you can make the following settings that cannot be made from the I/O Timing Setting Window:

- If you want to generate a cyclic virtual interrupt like timer interrupts, you can use the while statement to specify a repetition of virtual interrupt generation.
- You can specify that the priority levels set in the interrupt control register's interrupt priority level select bits be referenced to resolve the interrupt priority of virtual interrupts generated.
- As conditions for entering virtual port inputs or generating virtual interrupts, you can specify a combination of program fetch, memory access for read/write, or memory comparison.

In addition to the above, various other I/O settings are possible.

9.1 Method for Writing I/O Script

This section explains the method for defining virtual port inputs, virtual interrupts, and other I/Os to be written in I/O script by using definition examples. To define an I/O script, write a procedure for it. Enclose a procedure with braces "{ }" as you write it. Multiple procedures can be written in one file. In each procedure, write settings, timings, etc. of virtual port inputs or virtual interrupts. Each of the multiple procedures thus defined are processed in parallel with program execution. However, the order in which each procedure is evaluated is indeterminate. Register the I/O script file you've created in debugger using the popup menus [Load] in the I/O Timing Setting Window. Multiple I/O script files can be registered.
However, the total number of procedures that can be registered is limited to 50. Also, if you are using the Printf function output function in the Output Port Window, the number of virtual port input, virtual interrupt, and I/O script procedures that can be set is limited to a total of 48.

- Procedure 1 in the example below defines the timer mode of timer A0.
  In this example, a timer A0 interrupt is generated every divide-by-ratio (number of cycles) specified for the timer A0. The value specified in the interrupt control register is referenced to determine the priority of this timer interrupt.
- Procedure 2 in the example below defines a cycle-synchronized virtual port input.
  In this example, data is input from virtual port to memory when the program has been executed 10,000 cycles. Although the I/O Timing Setting Window supports virtual port inputs in only bytes, I/O scripts allow for virtual port inputs in words or long words.

```
; Definition example of I/O script file                → Comment statement.
; Definition of procedure 1 (example for virtual interrupt)
{                                                      → Beginning of procedure 1.
    while(1){                                          → while statement.
        if( ([0x380].b & 0x01) == 0x01){               → Checks timer A0's count start flag.
            waitc [0x386].w + 1                        → Keeps execution of I/O script waiting for the number
                                                         of cycles equal to the divide-by-ratio that is set for
                                                         timer A0.
            int 21, [0x55] & 0x7                       → Generates a timer A0 virtual interrupt.
                                                         (Interrupt control register is referenced to determine
                                                         priority.)
        }else{
            waiti 100                                  → Keeps execution of I/O script waiting for
                                                         100 instructions.
        }
    }
}                                                      → Terminates procedure 1.
; Definition of procedure 2 (example for virtual port input)
{                                                      → Beginning of procedure 2.
    waitc 10000                                        → Keeps execution of I/O script waiting for 10,000 cycles.
    set [0x3e0] = 0x20                                 → Inputs 0x20 in address 0x3e0.
    waitc 10000↵
    set [0x3e0].w = 0x4143                             → Inputs 2-byte data 0x4143 from address 0x3e0.
}                                                      → Terminates procedure 2.
```

9.2 Composition of I/O Script

Following statements can be written in I/O script:

- Procedure
- I/O script statement
- Judgment (if, else)
  Execution statements are branched off by judging the evaluation result of expression.
- Repeat statement (while) and Break statement
  Statements are executed repeatedly by judging the evaluation result of expression.
- Comment statement
  Comments can be written in I/O script. Comment statements are ignored when executing I/O script.

9.2.1 Procedure

Procedures specify a definition block of an I/O script. Multiple procedures can be written in one file. However, the total number of procedures that can be registered is limited to 50. Also, if you are using the Printf function output function in the Output Port Window, the number of virtual port input, virtual interrupt, and I/O script procedures that can be set is limited to a total of 48. The following shows a description format.

```
{
   Statements
}
{
   Statements
}

In the same way, multiple procedures can be defined below.
:
```

9.2.2 I/O Script Statements

Following five statements can be used in I/O script:

9.2.2.1 waiti statement

Format: waiti number of machine instructions
Function:
Execution of the next statement is kept waiting for a specified number of machine instructions.
Right-side expressions can be used to specify the number of machine instructions.
(Specification of right-side expressions is described later.)
For example, if following statements are written

```
waiti 100
set [0x800] = 0x10
```

the set statement is executed only after executing 100 machine instructions.

9.2.2.2 waitc statement

Format: waitc number of cycles
Function:
Execution of the next statement is kept waiting for a specified number of cycles.
Right-side expressions can be used to specify the number of cycles. (Specification of right-side expressions is described later.)
For example, if following statements are written

```
waitc 10000
set [0x800] = 0x10
```

the set statement is executed only after executing the program 10,000 cycles.

9.2.2.3 int statement

Format: int vector number , priority
Function:
The virtual interrupt of a specified vector number is generated in a specified order of priority.
Right-side expressions can be used to specify the vector number and priority.
(Specification of right-side expressions is described later.)
For example, if following statements are written

```
int 21 , 5
```

a timer A0 (vector number 21) interrupt is generated at priority level 5.

9.2.2.4 set statement

There are following three formats for the set statement:
Format 1: set memory address = input value
Function:
The input value is input to a specified memory address (virtual port input to memory).
Left-side expressions can be used to specify the memory address, and right-side expressions can be used to specify the input value. (Specifications of left-side and right-side expressions are described later.)
For example, if following statements are written

```
set [0x3e0] = 0x1d
```

data 0x1d is input to memory address 0x3e0.

Format 2: set condition expression, memory address = input value 1,input value 2, ...
Function:
The input value 1, input value 2, etc. are sequentially input to a specified memory address each time the conditional expression is established.
Left-side expressions can be used to specify the memory address, and right-side expressions can be used to specify the conditional expressions and input values.
(Specifications of left-side and right-side expressions are described later.)
For example, if following statements are written

```
set #isfetch:0xf0000 , [0x3e1] = 0x10 , 0x20
```

; #isfetch becomes true (established) when the program executes a specified address.

data 0x10 and 0x20 are sequentially input to memory address 0x3e1 each time the program executes address 0xf0000.
Namely, data 0x10 is input to memory address 0x3e1 when address 0xf0000 is executed first, and data 0x20 is input when the address is executed next.

Format 3: set % macro variable = right-side expression
Function:
The right-side expression is placed in a specified macro variable. (Specification of macro variables is described later.)
For example, following macro variables can be written:

```
set %val = 10; Macro variable val is initialized to 10.
set %val = %val + 1; Value of the macro variable is incremented by 1.
```

9.2.2.5 pass statement

Format: pass conditional expression, pass count
Function:
Execution of the next statement is skipped a number of times as specified by the pass count until the conditional expression is met.
Right-side expressions can be used to specify the conditional statement and pass count. (Specification of right-side expressions is described later.)
For example, if following statements are written

```
pass #isint:21 , 3
```

; #isint becomes true (established) when a specified virtual interrupt is generated.

```
set [0x800] = 0x10
```

The set statement is executed only after a timer A0 interrupt (vector number 21) occurs three times.

9.2.3 Judge Statements (if, else)

Judge statements judge the results of expressions, thereby causing the statements to be executed to branch off. The following shows a description format.

```
if (conditional expression) {
    Statement 1
} else if (conditional expression) {
    Statement 2
} else {
    Statement 3
}
```

- When if (conditional expression) is true (not 0) statement 1 is executed. If the conditional expression is false (= 0), else if (conditional expression) is evaluated to see whether it is true or false. If the conditional expression is true, statement 2 is executed. Otherwise, statement 3, the else statement, is executed.
- The else if and else statements can be omitted.
- The if statement can be nested in up to 32 levels.
- Right-side expressions can be used for the conditional expression.
- The conditional expressions written in I/O script are calculated as unsigned type. Therefore, if negative values are compared in an if statement, the operation to be performed by this debugger is indeterminate.

9.2.4 Repeat Statement (while) and break Statement

Repeat statements judge the results of expressions, thereby executing statements repeatedly. The following shows a description format.

```
while (conditional expression) {
   statement or break statement
}
```

- If the conditional expression is true, the statement is executed repeatedly. If the conditional expression is false, program execution exits from the loop.
- The while statement can be nested in up to 32 levels.
- A break statement is used if it is necessary to forcibly exit the while statement. If the while statement is nested, program execution exits from the innermost loop.
- Right-side expressions can be used for the conditional expression.
- The conditional expressions written in I/O script are calculated as unsigned type.
  Therefore, if negative values are compared in an while statement, the operation to be performed by this debugger is indeterminate.

9.2.5 Comment Statements

Comment statements are used to write comments in I/O script. The following shows a description format.

```
;character string
```

- A comment statement starts from a semicolon (;).
- A range of statement from the semicolon (;) till the end of the line is handled as a comment.
- Lines of comment statements are ignored when executing I/O scripts.

9.3 Method for Writing Right-side Expressions

Right-side expressions can be used to write the number of machine instructions or cycles,vector numbers, priority levels, input values, conditional expressions, or pass counts in I/O script statements, as well as write expressions in if and while statements. The following shows an example of an I/O script statement written using right-side expressions.

```
waitc LABEL
waiti [0x800] + 20
if( [0x1ff] == 0x30 )
while( #isfetch:0xf0000 )
```

Right-side expressions may be composed of the following:

- Constant
- Symbol and label
- Macro variable
- Memory variable
- Character constant
- Operator
- #isfetch, #isint, #isread, #iswrite

Each part of right-side expressions are described below.

9.3.1 Constants

Binary, decimal, and hexadecimal numbers can be input. The radix of numerals is discriminated by a symbol added at the beginning or end of a numeric value.

	Hexadecimal	Decimal	Binary
Beginning	0x,0X	None	%
Example	0xAB24	1234	%10010

9.3.2 Symbols and Labels

The global symbols and global labels defined in the target program can be used.

- Symbol and label names may consist of alphanumeric characters, underscore (_),period (.), and question mark (?). However, numbers cannot be used at the beginning of symbol and label names.
- Symbol and label names can be written in up to 255 characters.
- Symbol and label names are discriminated between uppercase and lowercase letters.
- The structured instructions, pseudo-instructions, macro-instructions, and reserved op-code words of assembler asxx cannot be used in symbol and label names.(These, for example, include .SECTION, .BYTE, switch, and if.)
- Character strings that begin with ".." cannot be used in symbol and label names.

9.3.3 Macro Variables

Macro variables are used by adding "%" at the beginning of each variable name.

- Variable names following the percent character (%) may consist of alphanumeric characters and underscore (_). However, numbers cannot be used at the beginning of macro variable names.
- Register names cannot be used in variable names.
- Variable names are discriminated between uppercase and lowercase letters.
- Up to 32 macro variables can be defined. Once defined, the macro variables remain effective until the debugger is terminated.

9.3.4 Memory Variables

Memory variables are used when using memory values in expressions. The following shows a format of memory variables.
[address].data-size

- Expressions can be written in address. (Memory variables also can be used.)
- Specify data size as shown in the table below.

For byte size	B or b
For word (2-byte) size	W or w
For long word (4-byte) size	L or l

- If specification of data size is omitted, the data size is assumed to be byte long.

Example 1: To reference memory contents at address 800016 in bytes

```
[0x8000].B or [0x8000]
```

Example 2: To reference memory contents at address 800016 in words

```
[0x8000].w
```

Example 3: To reference memory contents at address 800016 in long words

```
[0x8000].L
```

9.3.5 Character Constants

Specified characters or character strings are handled as constants after being converted into ASCII code.

- Characters must be enclosed with single quotations.
- Character strings must be enclosed with double quotations.
- Character strings must be within 2 characters (16 bits in length).

If a character string consists of more than two characters, only the last two characters written in the string are operated on. For example, if you write "ABCD," only the last two characters in this string, i.e., "CD," are operated on, the value of which is 434416.

9.3.6 Operators

The following lists the operators that can be written in expressions.
The priorities of operators are such that level 1 is the highest, and level 12, the lowest.
If operators in an expression have the same priority, they are calculated sequentially from left to right.

Operator	Meaning	Priority
( )	Parentheses	Level 1
+, -, ~	Unary plus, unary minus, unary bit logic	Level 2
*, /	Binary multiplication, binary division	Level 3
+, -	Binary addition, binary subtraction	Level 4
>>,	Shift right, shift left	Level 5
<, <=, >, >=	Binary comparison	Level 6
==, !=	Binary comparison	Level 7
&	Binary logical AND	Level 8
^	Binary exclusive OR	Level 9
|	Binary logical OR	Level 10
&&	Logical AND	Level 11
||	Logical OR	Level 12

9.3.7 #isfetch, #isint, #isread, #iswrite

These statements are used in conditional expressions of I/O script statements and if and while statements.

9.3.7.1 #isfetch expression

Format: #isfetch: address
Function:
The value of the expression becomes true (= 1) when the program's PC value goes to a specified address. Otherwise, the expression is false (= 0). For example, the if statement below

```
if ( #isfetch:0xfc000)
```

becomes true (= 1) when the program's address (PC value) becomes 0xfc000.

9.3.7.2 #isint expression

Format: #isint: vector number
Function:
The value of the expression becomes true (= 1) immediately after a virtual interrupt of a specified vector number is generated. Otherwise, the expression is false (= 0).
For example, the if statement below

```
if ( #isint:13)
```

becomes true (= 1) if a virtual interrupt of vector number 13 had occurred immediately before the if statement was evaluated.

9.3.7.3 #isread expression

Format: #isread: address
Function:
The value of the expression becomes true (= 1) immediately after a specified memory address is accessed for read (to read data from memory). Otherwise, the expression is false (= 0).
For example, the if statement below

```
if ( #isread:0x800)
```

becomes true (= 1) if memory at address 0x800 had been accessed for read immediately before the if statement was evaluated.

9.3.7.4 #iswrite expression

Format: #iswrite: address
Function:
The value of the expression becomes true (= 1) immediately after a specified memory address is accessed for write (to write data to memory). Otherwise, the expression is false (= 0).
For example, the if statement below

```
if ( #iswirte:0x800 )
```

becomes true (= 1) if memory at address 0x800 had been accessed for write immediately before the if statement was evaluated.

9.4 Method for Writing Left-side Expressions

Left-side expressions can be written in memory addresses and macro variables of the set statement in I/O script statements. The following shows an example of an I/O script statement using left-side expressions.

```
set [0x3e0] = 0x1a
set %val = 10
```

Left-side expressions may be composed of the following:

- Macro variable
- Memory variable

Each part of left-side expressions are described below.

9.4.1 Macro Variables

Macro variables are used by adding "%" at the beginning of each variable name.

- Variable names following the percent character (%) may consist of alphanumeric characters and underscore (_). However, numbers cannot be used at the beginning of macro variable names.
- The values that can be handled by an expression that is substituted for macro variables are integers in the range of 0 to FFFFFFFF16. If negative numbers are used, they are handled as 2's complements.

When specifying a repeat count for the while statement, use of macro variables should prove convenient.

set %val = 0	;Macro variable %val is assigned 0.
while(%val	;while statement is repeated until %val = 10.
waitc 10000	
int 13,5	
set %val = %val + 1	;%val is incremented by 1.
}	

9.4.2 Memory Variables

This variable is used when writing values in memory. The following shows a format of memory variables.

[address].data-size

- Expressions can be written in address. (Memory variables cannot be used.)
- Specify data size as shown in the table below.

For byte size	B or b
For word (2-byte) size	W or w
For long word (4-byte) size	L or l

- If specification of data size is omitted, the data size is assumed to be byte long.

Example 1: When writing to memory at address 0x8000 in bytes

```
set [0x8000].B = 0x10 or set [0x8000] = 0x10
```

Example 2: When writing to memory at address 0x8000 in words

```
set [0x8000].w = 0x1234
```

Example 3: When writing to memory at address 0x8000 in long words

```
set [0x8000].L = 0x12345678
```

10. C/C++ Expressions

10.1 Writing C/C++ Expressions

You can use C/C++ expressions consisting of the tokens shown below for registering C watchpoints and for specifying the values to be assigned to C watchpoints.

Token	Example
Immediate values	10, 0x0a, 012, 1.12, 1.0E+3
Scope	::name, classname::member
Mathematical operators	+, -, *, /
Pointers	*, **, ...
Reference	&
Sign inversion	-
Member reference using dot operator	Object.Member
Member reference using arrow	Pointer->Member, this->Member
Pointers to Members	Object.*var, Pointer->*var
Parentheses	(,)
Arrays	Array[2], DArray[2] [3] , ...
Casting to basic types	(int), (char*), (unsigned long *), ...
Casting to typedef types	(DWORD), (ENUM), ...
Variable names and function names	var, i, j, func, ...
Character constants	'A', 'b', ...
Character string literals	"abcdef", "I am a boy.", ...

10.1.1 Immediate Values

You can use hexadecimals, decimals, octals as immediate values. Values starting with 0x are processed as hexadecimals, those with 0 as octals, and those without either prefix as decimals.
Floating-point numbers can also be used to assign values to variables.

Notes

- You cannot register only immediate values as C watchpoints.
- The immediate value is effective only when it is used in C/C++ language expressions that specify C/C++ watchpoints or when it is used to specify the value to be assigned to those expressions. When using floating-point numbers, operation cannot be performed on an expression like 1.0+2.0.

10.1.2 Scope Resolution

The scope resolution operator :: is available as following.
Global scope: ::valiable name

```
::x, ::val
```

Class scope: class name::member name, class name::class name::member name, e.g.

```
T::member, A::B::member
```

10.1.3 Mathematical Operators

You can use the addition (+), subtraction (-), multiplication (*), and division (/) mathematical operators. The following shows the order of priority in which they are evaluated.

```
(*), (/), (+), (-)
```

Notes

- There is no support currently for mathematical operators for floating point numbers.

10.1.4 Pointers

Pointers are indicated by the asterisk (*). You can use pointer to pointers **, and pointer to pointer to pointers ***, etc.

```
Examples: "*variable_name", "**variable_name", etc.
```

Notes

- Immediate values cannot be processed as pointers. That is, you cannot specify *0xE000, for example.

10.1.5 Reference

References are indicated by the ampersand (&). You can only specify "&variable_name".

10.1.6 Sign Inversion

Sign inversion is indicated by the minus sign (-). You can only specify "-immediate_value" or "-variable_name". No sign inversion is performed if you specify 2 (or any even number of) minus signs.

Notes

- There is no support currently for sign inversion of floating point numbers.

10.1.7 Member Reference Using Dot Operator

You can only use "variable_name.member_name" for checking the members of structures and unions using the dot operator.
Example:

```
class T {
public:
int member1;
char member2;
};
class T t_cls;
class T *pt_cls = &t_cls;
```

In this case, t_cls.member1, (*pt_cls).member2 correctly checks the members.

10.1.8 Member Reference Using Arrow

You can only use "variable_name->member_name" for checking the members of structures and unions using the arrow.
Example:

```
class T {
public:
int member1;
char member2;
};
class T t_cls;
class T *pt_cls = &t_cls;
```

In this case, (&t_cls)->member1, pt_cls->member2 correctly checks the members.

10.1.9 Pointers to Members

Pointers to members using the ".*" or "->*" operator can be refered only in the forms of variable name .* member name or variable name ->* member name.
Example:

```
class T {
public:
int member;
};
class T t_cls;
class T *pt_cls = &t_cls;

int T::*mp = &T::member;
```

In this case, t_cls.*mp and tp_cls->*mp can correctly reference the variable of pointer-to-member type.

Note

- Note that the expression *mp cannot considered as the variable of pointer-to-member type.

10.1.10 Parentheses

Use the '(' and ')' to specify priority of calculation within an expression.

10.1.11 Arrays

You can use the ' [' and '] ' to specify the elements of an array. You can code arrays as follows: "variable_name [(element_No or variable)] ", "variable_name [(element_No or variable)] [(element_No or variable)] ", etc.

10.1.12 Casting to Basic Types

You can cast to C basic types char, short, int, and long, and cast to the pointer types to these basic types. When casting to a pointer type, you can also use pointers to pointers and pointers to pointers to pointers, etc.
Note that if signed or unsigned is not specified, the default values are as follows:

Basic type	Default
char	unsigned
short	signed
int	signed
long	signed

Notes

- Of the basic types of C++, casts to bool type, wchar_t type, and floating-point type (float or double) cannot be used.
- Casts to register variables cannot be used.

10.1.13 Casting to typedef Types

You can use casting to typedef types (types other than the C basic types) and the pointer types to them. When casting to a pointer type, you can also use pointers to pointers and pointers to pointers to pointers, etc.

Notes

- You cannot cast to struct or union types or the pointers to those types.

10.1.14 Variable Name

Variable names that begin with English alphabets as required
under C/C++ conventions can be used.
The maximum number of characters for variable name is 255.
And 'this' pointer is available.

10.1.15 Function Name

Function names that begin with English alphabets as required
under C conventions can be used.
In the case of C++, no function names can be used.

10.1.16 Character Constants

You can use characters enclosed in single quote marks (') as character constants. For example, 'A', 'b' , etc. These character constants are converted to ASCII code and used as 1-byte immediate values.

Notes

- You cannot register character constants only as C watchpoints.
- Character constants are valid only when used in a C/C++ expression that specifies a C watchpoint, and when specifying a value to be assigned (character constants are processed in the same manner as immediate values).

10.1.17 Character String Literals

You can use character strings enclosed in double quote marks (") as character string literals. Examples are "abcde", "I am a boy.", etc.

Notes

- Character string literals can only be placed on the right side of an assignment operator in an expression. They can only be used when the left side of the assignment operator is a char array or a char pointer type. In all other cases, a syntax error results.

10.2 Display Format of C/C++ Expressions

C/C++ expressions in the data display areas of the C Watch Windows are displayed as their type name, C/C++ expression (variable name), and result of calculation (value), as shown below.
The following describes the display formats of the respective types.

10.2.1 Enumeration Types

- When the result (value) of calculation has been defined, its name is displayed.
  ```
  (DATE) date = Sunday(all Radices)
  ```
- If the result (value) of calculation has not been defined, it is displayed as follows:
  ```
  (DATE) date = 16 (when Radix is in initial state)
  (DATE) date = 0x10(when Radix is hex)
  (DATE) date = 0000000000010000B(when Radix is binary)
  ```

10.2.2 Basic Types

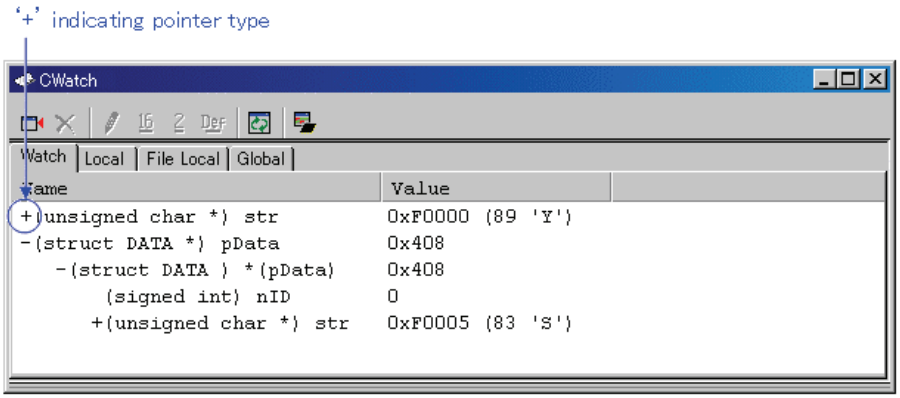
- When the result of calculation is a basic type other than a char type or floating point type, it is displayed as follows:
  ```
  (unsigned int) i = 65280(when Radix is in initial state)
  (unsigned int) i = 0xFF00(when Radix is hex)
  (unsigned int) i = 1111111100000000B(when Radix is binary)
  ```
- When the result of calculation is a char type, it is displayed as follows:
  ```
  (unsigned char) c = 'J'(when Radix is in initial state)
  (unsigned char) c = 0x4A(when Radix is hex)
  (unsigned char) c = 10100100B(when Radix is binary)
  ```
- When the result of calculation is a floating point, it is displayed as follows:
  ```
  (double) d = 8.207880399131839E-304(when Radix is in initial state)
  (double) d = 0x10203045060708(when Radix is hex)
  (double) d = 0000000010.....1000B(when Radix is binary)
  (..... indicates abbreviation)
  ```

10.2.3 Pointer Types

- When the result of calculation is a pointer type to other than a char* type, it is displayed in hexadecimal as follows:
  (unsigned int *) p = 0x1234(all Radices)
- When the result of calculation is a char* type, you can select the display format of the string or a character in the C Watch window's menu [Display String].
  - string types
    (unsigned char *) str = 0x1234 "Japan"(all Radices)
  - character types
    (unsigned char *) str = 0x1234 (74 'J')(all Radices)

l When the result of calculation is a char* type, it is displayed as follows:
(unsigned char *) str = 0x1234 "Jap(all Radices)

If the string contains a non-printing code prior to the code to show the end of the string (0), it is displayed up to the non-printing character and the closing quote mark is not displayed.



You can double-click on lines indicated by a '+' to see the members of that structure or union. The '+' changes to a '-' while the members are displayed. To return to the original display, double click the line, now indicated by the '-'.

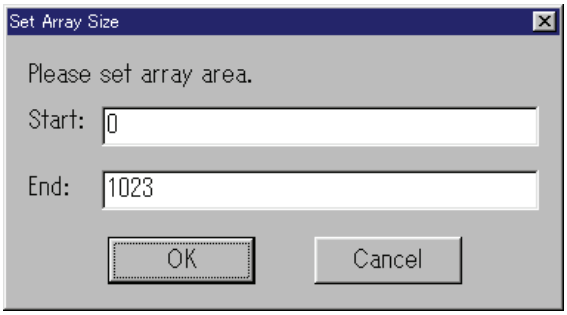
10.2.4 Array Types

- When the result of calculation is an array type other than a char [ ] type, the starting address is displayed in hex as follows:
  (signed int [10]) z = 0x1234(all Radices)
- When the result of calculation is a char [ ] type, it is displayed as follows:
  (unsigned char [10]) str = 0x1234 "Japan"(all Radices)

If the string contains a non-printing code prior to the code to show the end of the string (0), it is displayed up to the non-printing character and the closing quote mark is not displayed.

(unsigned char [10]) str = 0x1234 "Jap(all Radices)

Also if the string contains more than 80 characters, the closing quote mark is not displayed. When the C/C++ expression is an array type as same as pointer type, a '+' is display to the left of the type name. You can see the elements of the array by using this indicating. (for the details, refer 10.2.3 Pointer Types) When the number of the array elements is more than 100, the following dialog box open. Specify the number of the elements in the dialog box.



The elements from the index specified in "Start" to the index specified in "End" are displayed. If you specify the value more than the max index of the array, the value is regarded as max index of the array. When you click the "Cancel" button, the elements are not displayed.

10.2.5 Function Types

- When the result of calculation is a function type, the starting address is displayed in hex as follows:
  (void()) main = 0xF000(all Radices)

10.2.6 Reference Types

- When the result of calculation is a reference type, the reference address is displayed in hex as follows:
  (signed int &) ref = 0xD038(all Radices)

10.2.7 Bit Field Types

- When the result of calculation is a bit field type, it is displayed as follows:
  (unsigned int :13) s.f = 8191(when Radix is in initial state)
  (unsigned int :13) s.f = 0x1FFF(when Radix is hex)
  (unsigned int :13) s.f = 1111111111111B(when Radix is binary)

10.2.8 When No C Symbol is Found

If the calculated expression contained a C symbol that could not be found, it is displayed as follows:

```
() x = <not active>(all Radices)
```

10.2.9 Syntax Errors

- When the calculated expression contains a syntax error, it is displayed as follows:

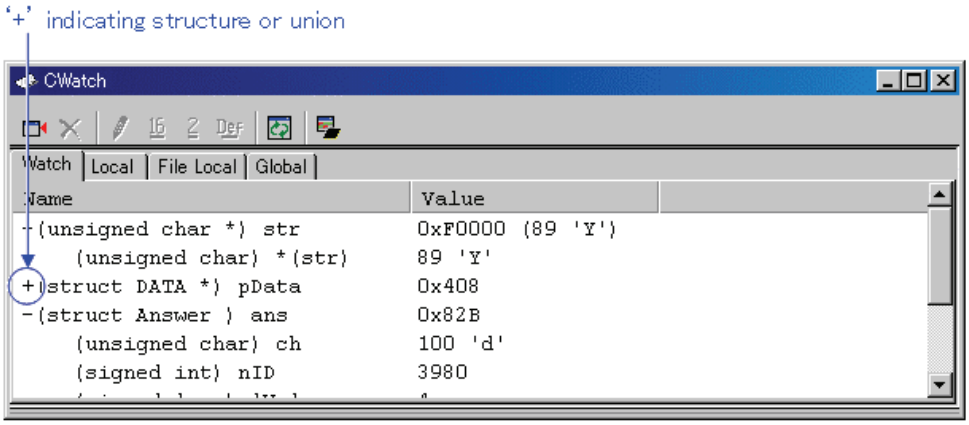
```
() str*(p = <syntax error>(all Radices)
(where str*(p is the syntax error)
```

10.2.10 Structure and Union Types

- When the result of calculation is a structure or union type, the address is displayed in hex as follows:

```
(Data) v = 0x1234  (all Radices)
```

If, as in structures and unions, the C/C++ expression consists of members, a '+' is displayed to the left of the type name (tag name).



You can double-click on lines indicated by a '+' to see the members of that structure or union. The '+' changes to a '-' while the members are displayed. To return to the original display, double click the line, now indicated by the '-'. This function allows you to check the members of structures and unions.

Attention

If a variable is declared with the same name as the type definition name declared by typedef, you cannot reference that variable.

- Register Variables

When the result of calculation is a register variable, "register" is displayed to the left of the type name as follows:

```
(register signed int) j = 100
```

11. Display the Cause of the Program Stoppage

If the program is stoped by the debug function, the cause of the stoppage is displayed in the Output window or Status window ([Platform] sheet).
The debugger for 740 doesn't support this function.

The contents of a display and the meaning of "the cause of the stoppage" are as follows.

Display	The cause of the stoppage
Halt	The stop by the [Halt Program] button/menu
S/W break	Software break
H/W break	Hardware break
Memory access error	Memory access error break
Undefined instruction	Undefined instruction break

12. Attention

12.1 Common Attention

12.1.1 File operation on Windows

The following points should be noted:

1. File Name and Directory Name
   - Do not use directory names or filenames that include blanks.
   - Operation is not guaranteed if your directory names and filenames include kanji.
   - Use only one period in a filename.
2. Specify the File and Directory
   - You cannot use "..." to specify two levels upper directories.
   - You cannot use a network pathname. You must allocate a drive.

12.1.2 Area where software breakpoint can be set

Software breakpoints can be set in the entire area.

12.1.3 Get or set C variables

- If a variable is declared with the same name as the type definition name declared by typedef, you cannot reference that variable.
- Values cannot be changed for register variables and bit fields.
- Values cannot be changed for 64 bit width variables (long long, double, and so on).
- Values cannot be changed for C/C++ expressions that do not indicate the memory address and size.
- For the sake of optimization, the C compiler may place different variables at the same address. In this case, values of the C variable may not be displayed correctly.
- Literal character strings can only be substituted for char array and char pointer type variables.
- No arithmetic operations can be performed on floating point types.
- No sign inversion can be performed on floating point types.
- Casting cannot be performed on floating point types.
- Casting cannot be performed on register variables.
- Casting cannot be performed on structure types, union types, or pointer types to structure or union types.
- Character constants and literal character strings cannot contain escape sequences.

12.1.4 Function name in C++

- When you input the address using the function name in setting display address, setting break points, and so on, you can not specify the member function, operator function, and overloaded function, of a class.
- You can not use function names for C/C++ expression
- No script commands (e.g., breakin and func) can be used in which function names are specified for arguments.
- In address value specifying columns of dialog boxes, no addresses can be specified using function names.

12.1.5 Debugging multi modules

If you register two or more absolute module file in one session, you can download only one file in same time.
If you register one absolute module file and one or more machine language file in one session, you can download all file in same time.

12.1.6 Syncronized debugging

Syncronized debugging function is not available.

12.1.7 Virtual port output functions

The maximum number of data that can be acquired on I/O timing setting window's virtual port output or output port windows can be specified in the Init dialog box. We recommend specifying a value less than 500,000 for this data count. In the 740 debugger, the maximum number of data is 30000.
Specifying any value greater than that may cause the performance of the simulator debugger or Windows to drop.

12.2 Attention of the R32C Debugger

12.2.1 Option of C Compiler/Assembler/Linker

The information may not be downloaded/debugged normally depending on the option designation of the compiler, assembler, and linker.
Please refer to the following for the option specification.
Refer to "12.6 Options for compiler, assembler, and linker"

The compiler that can be used by R32C debugger:

- NCxx

12.3 Attention of the M32C Debugger

12.3.1 Option of C Compiler/Assembler/Linker

The information may not be downloaded/debugged normally depending on the option designation of the compiler, assembler, and linker.
Please refer to the following for the option specification.
Refer to "12.6 Options for compiler, assembler, and linker"

The compiler that can be used by M32C debugger:

- NCxx
- the IAR EC++ Compiler
- the IAR C Compiler

12.4 Attention of the M16C/R8C Debugger

12.4.1 Options for compiler, assembler, and linker

The information may not be downloaded/debugged normally depending on the option designation of the compiler, assembler, and linker.
Please refer to the following for the option specification.
Refer to "12.6 Options for compiler, assembler, and linker"

The compiler that can be used by M16C/R8C debugger:

- NCxx
- the IAR EC++ Compiler
- the IAR C Compiler
- the TASKING C Compiler

12.4.2 TASKING C Compiler

When you debug programs compiled by the TASKING C Compiler "CCM16", the type of bit field is fixed on "unsigned short int". Because CCM16 outputs the debug information for the type of bit field as "unsigned short int."

12.4.3 Precautions on Using M16C/62 Group

This debugger does not support the M16C/62 group's memory extension mode. (Only the normal mode is supported.)

12.4.4 The number of cycles measuring on using R8C/Tiny Series

The M16C, R8C/Tiny Series simulator debugger is an instruction set simulator and does not include bus width, cue, or wait events when counting the number of cycles.
Rather, it uses the values indicated in the microcomputer software manual for the number of cycles.
The cycles count indicated for the R8C/Tiny Series is the number of cycles it takes to read or write 8 bits of data to or from the memory with no wait when the instruction codes are lined up in the instruction cue buffer.
Therefore, depending on the conditions of the instruction cue buffer, etc., when the program is run on the simulator debugger, the results may show half the number of cycles in comparison to the results when using the emulator debugger. Please keep this in mind when using these measurements.

For more details concerning how to calculate the number of cycles for the R8C/Tiny Series, please refer to the R8C/Tiny Series Software Manual.

12.5 Attention of the 740 Debugger

12.5.1 Options for compiler, assembler, and linker

The information may not be downloaded/debugged normally depending on the option designation of the compiler, assembler, and linker.
Please refer to the following for the option specification.
Refer to "12.6 Options for compiler, assembler, and linker"

The compiler that can be used by 740 debugger:

- the Assembler Package for 740 Family　SRA74
- the IAR C Compiler

12.5.2 Not support functions

- The simulator debugger for 740 does not support the real-time tracing. The following windows and script commands are not available.
  - Output Port Window
  - Trace Point Setting Window
  - Trace Window
  - Data Trace Window
  - TracePoint Command
  - TraceData Command
  - TraceList Command

- The whole of the memory range is available for debugging with the simulator debugger for 740.
  - The MAP command to assign memories is not supported.
  - It is not necessary to specify a range for coverage.

- The simulator engine for 740 can not detect factors of breaks. Therefore, the factors are not displayed in the Output window and the Status window([Platform] sheet).

- The debugger for 740 does not support stack tracing. Therefore, the following windows and script commands are not available.
  - Stack Trace Window
  - Up command
  - Down command
  - Where command

12.6 Options for compiler, assembler, and linker

The information may not be downloaded/debugged normally depending on the option designation of the compiler, assembler, and linker.
Please refer to the following for the option specification.

In the options other than the above-mentioned, the operation check is not done. Please acknowledge that the options other than the above-mentioned cannot be recommended.

12.6.1 When Using NCxx

When -O, -OR or -OS option is specified at compilation, the source line information may not be generated normally due to optimization, causing step execution to be operated abnormally.
To avoid this problem, specify -ONBSD (or -Ono_Break_source_debug) option together with -O, -OR or -OS option.

12.6.2 When Using the Assembler Package for 740 Family

Please assemble according to the following procedures and link.

At assemble

- "-c" option
  outputs debugging information concerned with source line to a relocatable file.

Note

When the directive comand .FUNC is specified to a function in a source file, if "-c" option is used, the name of the function will be not available. Please do not use the option to make the name available.

- "-s" option
  outputs local labels, local .equ symbols and local .bequ symbols to a relocatable file.

At link

- "-s" option
  generates a symbol file.

In the options other than the above-mentioned, the operation check is not done. Please acknowledge that the options other than the above-mentioned cannot be recommended.

12.6.2.1 Command Execution Examples

The following shows examples of entering commands depending on the product

- The Debugger for 740

```
>sra74 -c -s main.a74<Enter>
>sra74 -c -s sub.a74<Enter>
>link74 main sub ,,,-s<Enter>
```

12.6.3 When Using the IAR EC++ Compiler (EW)

Please specify the project setting by following process.

1. The Setting in the IAR Embedded Workbench
   When you select the menu [Project] -> [Options...], the dialog for "Options For Target " target"" will open. In this dialog, please select the "XLINK" as category, and set the project setting.
- Output Tab
  In the "Format" area, check the "Other" option, and select the "elf/dwarf" as "Output Format".
- Include Tab
  In the "XCL File Name" area, specify your XCL file (ex: lnkm32cf.xcl).

2. Edit the XCL file
   Add the command line option "-y" to your XCL file. The designation of "-y" option varies depending on the product.

Product Name	-y Option
The debugger for M32C	-yspc
The debugger for M16C/R8C	-yspc

3. Build your program after the setting above.

In the options other than the above-mentioned, the operation check is not done. Please acknowledge that the options other than the above-mentioned cannot be recommended.

12.6.4 When Using the IAR C Compiler (EW)

Please specify the project setting by following process.

1. The Setting in the IAR Embedded Workbench
   When you select the menu [Project] -> [Options...], the dialog for "Options For Target " target"" will open. In this dialog, please select the "XLINK" as category, and set the project setting.
   - Output Tab
     In the "Format" area, check the "Other" option, and select the "ieee-695" as "Output Format".
   - Include Tab
     In the "XCL File Name" area, specify your XCL file (ex: lnkm16c.xcl).

2. Edit the XCL file
   Add the command line option "-y" to your XCL file. The designation of "-y" option varies depending on the product.

Product Name	-y Option
The debugger for M32C	-ylmb
The debugger for M16C/R8C	-ylmb
The debugger for 740	-ylmba

3. Build your program after the setting above.

In the options other than the above-mentioned, the operation check is not done. Please acknowledge that the options other than the above-mentioned cannot be recommended.

12.6.5 When Using the IAR C Compiler (ICC)

12.6.5.1 Specify the Option

Please compile according to the following procedures and link.

- At compilation
  Specify the "-r" option.
- Before linking
  Open the linker's option definition file (extension .xcl) to be read when linking and add "-FIEEE695" and "-y" options. The designation of "-y" option varies depending on the product.

Product Name	-y Option
The debugger for M32C	-ylmb
The debugger for M16C/R8C	-ylmb
The debugger for 740	-ylmba

- At link
  Specify the linker's option definition file name using "-f" option.

In the options other than the above-mentioned, the operation check is not done. Please acknowledge that the options other than the above-mentioned cannot be recommended.

12.6.5.2 Command Execution Examples

The following shows examples of entering commands depending on the product

- The debugger for M32C

```
>ICCMC80 -r file1.c<Enter>
>ICCMC80 -r file2.c<Enter>
>XLINK -o filename.695 -f lnkm80.xcl file1 file2<Enter>
```

- The debugger for M16C/R8C

```
>ICCM16C -r file1.c<Enter>
>ICCM16C -r file2.c<Enter>
>XLINK -o filename.695 -f lnkm16c.xcl file1 file2<Enter>
```

- The debugger for 740

```
>ICC740 -r file1.c<Enter>
>ICC740 -r file2.c<Enter>
>XLINK -o filename.695 -f lnk7400t.xcl file1 file2<Enter>
```

The XCL file name varies depending on the product and memory model. For details, see the ICCxxxx manual.

12.6.6 When Using the TASKING C Compiler (EDE)

Please specify the project setting by following process.

1. Select menu - [EDE]->[C Compiler Option]->[Project Options...]. The "M16C C Compiler Options [Project Name]" dialog opens.
   Please set as follows by this dialog.
   - Optimeze Tab
     Please specify "No optimization" by Optimization level.
   - Debug Tab
     Please check only ""Enable generation of any debug information(including type checkeing)"" and "Genarate symbolic debug information".

2. Select menu - [EDE]->[Linker/Locator Options...]. The "M16C Linker/Locator Options [Project Name]" dialog opens.
   Please set as follows by this dialog.
   - Format Tab
     Please specify "IEEE 695 for debuggers(abs)" by Output Format.

3. Build your program after the setting above.

In the options other than the above-mentioned, the operation check is not done. Please acknowledge that the options other than the above-mentioned cannot be recommended.

12.6.7 When Using the TASKING C Compiler (CM)

12.6.7.1 Specify the Option

Please specify "-g" and "- O0" options when compiling.
In the options other than the above-mentioned, the operation check is not done. Please acknowledge that the options other than the above-mentioned cannot be recommended.

12.6.7.2 Command Execution Examples

The following shows examples of entering commands.

```
>CM16 -g -O0 file1.c<Enter>
```

[MEMO]

R32C Simulator Debugger V.1.00
User's Manual

Publication Date: Sep. 16, 2006 Rev.1.00

Published by: Sales Strategic Planning Div.
Renesas Technology Corp.

Edited by: Microcomputer Tool Development Department
Renesas Solutions Corp.

R32C Simulator Debugger V.1.00
User's Manual

Renesas Electronics Corporation
1753, Shimonumabe, Nakahara-ku, Kawasaki-shi, Kanagawa 211-8668 Japan

REJ10J1439-0100