EPSON
EXCEED YOUR VISION

CMOS 8-BIT SINGLE CHIP MICROCOMPUTER

S5U1C88000C Manual II

(Integrated Tool Package for S1C88 Family)

Workbench/Development Tools/Assembler Package Old Version

SEIKO EPSON CORPORATION

NOTICE

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

The C compiler, assembler and tools explained in this manual are developed by TASKING, Inc.
Windows 2000 and Windows XP are registered trademarks of Microsoft Corporation, U.S.A.
PC/AT and IBM are registered trademarks of International Business Machines Corporation, U.S.A.
All other product names mentioned herein are trademarks and/or registered trademarks of their respective owners.

Configuration of product number

Devices

S1 C 88104 F 0A01 00

Packing specifications

00	: Besides tape & reel	
0A	: TCP BL	2 directions
0B	: Tape & reel	BACK
0C	: TCP BR	2 directions
0D	: TCP BT	2 directions
0E	: TCP BD	2 directions
0F	: Tape & reel	FRONT
0G	: TCP BT	4 directions
0H	: TCP BD	4 directions
0J	: TCP SL	2 directions
0K	: TCP SR	2 directions
0L	: Tape & reel	LEFT
0M	: TCP ST	2 directions
0N	: TCP SD	2 directions
0P	: TCP ST	4 directions
0Q	: TCP SD	4 directions
0R	: Tape & reel	RIGHT
99	: Specs not fixed	

Specification

Package
[D: die form; F: QFP, B: BGA]

Model number

Model name
[C: microcomputer, digital products]

Product classification
[S1: semiconductor]

Development tools

S5U1 C 88348 D1 1 00

Packing specifications
[00: standard packing]

Version
[1: Version 1]

Tool type

- Hx : ICE
- Ex : EVA board
- Px : Peripheral board
- Wx: Flash ROM writer for the microcomputer
- Xx : ROM writer peripheral board
- Cx : C compiler package
- Ax : Assembler package
- Dx : Utility tool by the model
- Qx : Soft simulator

Corresponding model number
[88348: for S1C88348]

Tool classification
[C: microcomputer use]

Product classification
[S5U1: development tool for semiconductor products]

MANUAL ORGANIZATION

The S1C88 Family Integrated Tool Package contains the tools required to develop software for the S1C88 Family microcomputers. The S5U1C88000C Manual (S1C88 Family Integrated Tool Package) describes the tool functions and how to use the tools. The manual is organized into two documents as shown below.

I. C Compiler/Assembler/Linker

Describes the C Compiler and its tool chain ([Main Tool Chain] part shown in the figure on the next page).

II. Workbench/Development Tools/Assembler Package Old Version (this document)

Describes the Work Bench that provides an integrated development environment, Advanced Locator, the Mask Data Creation Tools ([Development Tool Chain] part shown in the figure on the next page), Debugger, and Structured Assembler ([Sub Tool Chain] part shown in the figure on the next page).

This manual assumes that the reader is familiar with C and Assembly languages.

Refer to the following manuals as necessary when developing an S1C88xxx microcomputer:

S1C88xxx Technical Manual

Describes the device specifications, control method and Flash EEPROM programming.

S5U1C88000Q Manual

Describes the operation of the tools included in the Simulator Package.

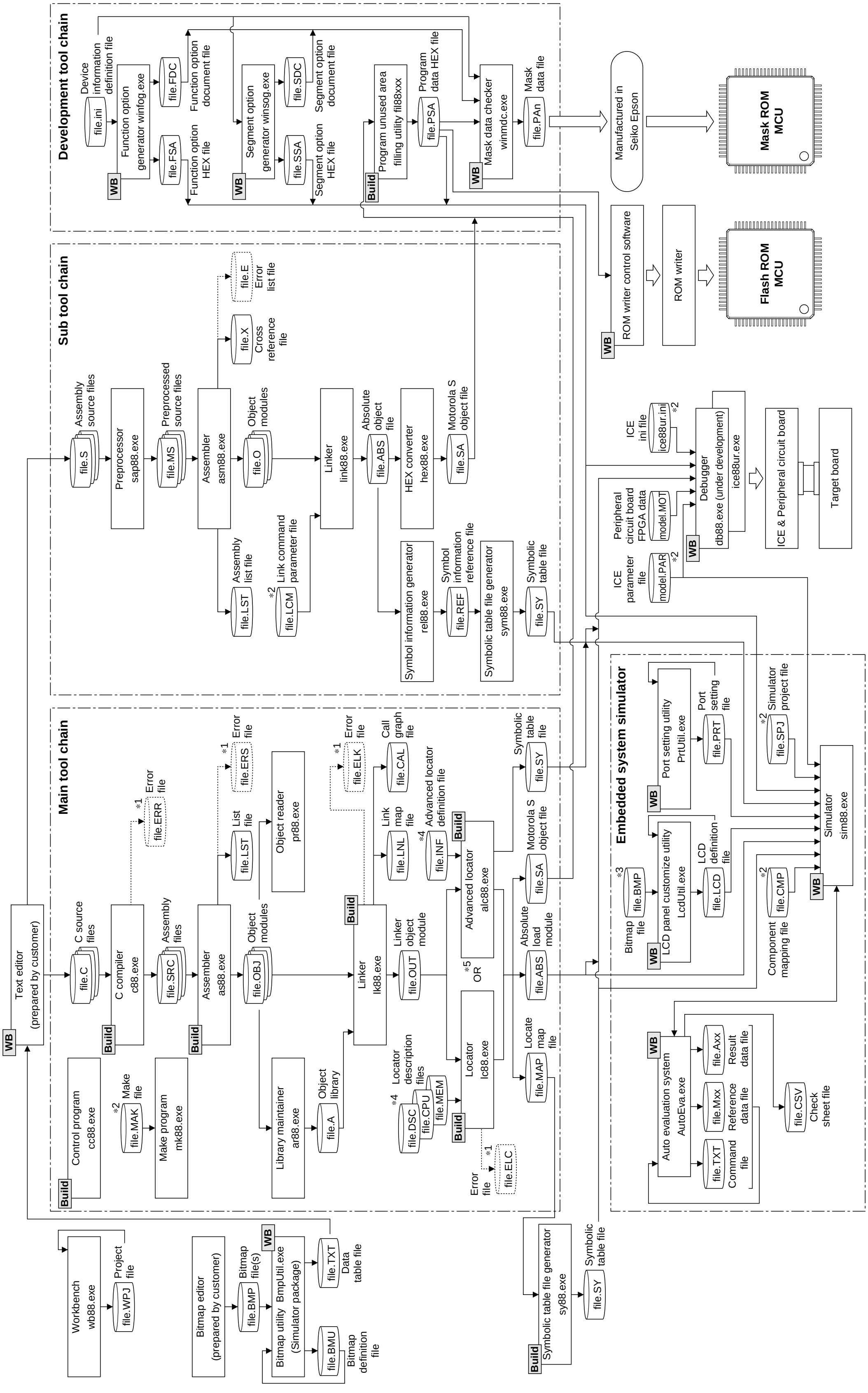
S5U1C88000H5 Manual

Describes the operation of the ICE (S5U1C88000H5).

S5U1C88xxxP Manual

Describes the operation of the peripheral circuit board installed in the ICE.

S1C88 FAMILY INTEGRATED DEVELOPMENT ENVIRONMENT



WB Can be invoked from the workbench wb88. Build Tools executed automatically during build process by wb88.

*1: If the error file is generated, wb88 displays the contents of the file in the message view and allows a tag jump function. *2: Created using a text editor. *3: Created using a bitmap editor. *4: Created using the wb88 section editor (or a text editor). *5: Selected by wb88.

CONTENTS

CHAPTER 1 GENERAL

1.1 Features

The S1C88 Family Integrated Tool Package contains software development tools that are common to all the models of the S1C88 Family. The package comes as an efficient working environment for development tasks, ranging from compiling / assembly source program to debugging.
The principal features are as follows:

Integrated working environment

The work bench wb88, a Windows GUI application, provides an integrated working environment that allows management of all files as a project, execution of make process, invocation of tools including the editor specified by the user.

Supports C and S1C88 Family assembly languages

This package contains C compiler tools as well as the conventional structured assembler tools.

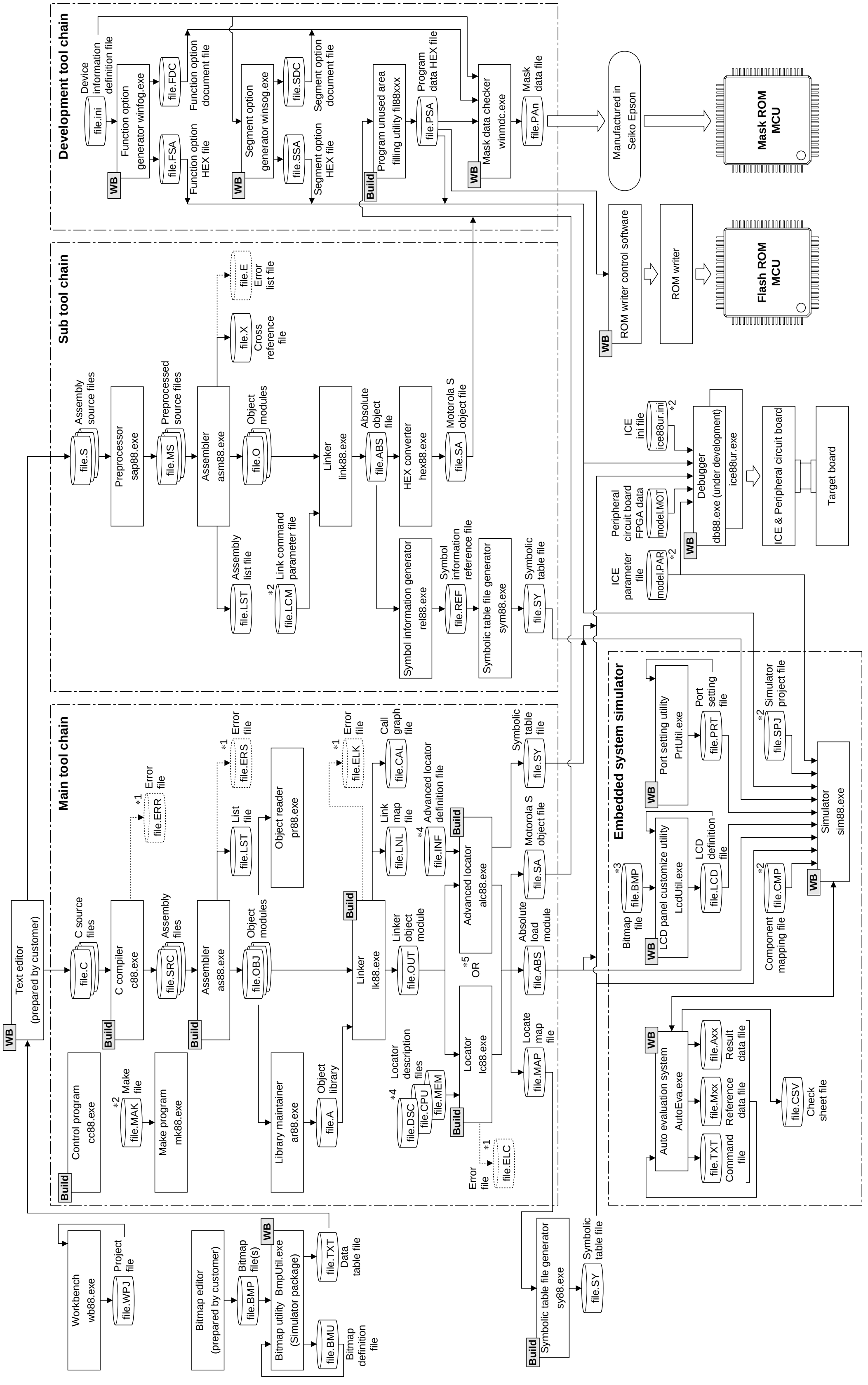
Supports simulator, auto evaluation system and ICE as debugging tools

The work bench invokes the ICE (S5U1C88000H5) an optional development tool for the S1C88 Family or the simulator after automatically generating a command file, this makes it possible to debug an object immediately after building.

Common to all S1C88 chips

The tools included in this package are common to all S1C88 Family models. The chip dependent information is read from the parameter file and device information definition file for each chip.

1.2 S1C88 Family Integrated Development Environment



WB Can be invoked from the workbench wb88.   Build Tools executed automatically during build process by wb88.

*1: If the error file is generated, wb88 displays the contents of the file in the message view and allows a tag jump function.   *2: Created using a text editor.   *3: Created using a bitmap editor.   *4: Created using the wb88 section editor (or a text editor).   *5: Selected by wb88.

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

Integrated working environment

Work Bench (wb88.exe)

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

Main tool chain

C compiler (c88.exe)

Compiles C source files to generate assembly source files.

Assembler (as88.exe)

Assembles the assembly source files generated by the C compiler to convert into relocatable object files.

Linker (lk88.exe)

Links relocatable object files and libraries to generate a combined relocatable object file.

Locator (lc88.exe)

Relocates the relocatable object file generated by the linker to generate a load module that has absolute address. This file is used for debugging and creating mask data.

Advanced locator (alc88.exe)

Realizes the locator's relocation functions without using description files in DELFEE. Moreover, it incorporates a new function that helps to optimize branching. See Chapter 5 for details about advanced locator.

The tools available in the Main tool chain, except advanced locator, are detailed in the document titled "S5U1C88000C Manual I".

Sub tool chain

Preprocessor (sap88.exe)

Expands the preprocessor instructions in assembly source files into the source codes that can be assembled.

Assembler (as88.exe)

Assembles the assembly source files generated by the preprocessor to convert into relocatable object files.

Linker (lk88.exe)

Relocates the relocatable object files generated by the assembler to generate an absolute object file.

Hex converter (hx88.exe)

Converts the absolute object file generated by the linker into a HEX data file in the Motorola S format. This HEX file is used for debugging and creating mask data.

Refer to Appendix for details of the tools in the Sub tool chain.

Development tool chain

Function option generator (winfog.exe)

This tool creates an ICE function option setup file after selecting the mask options of the S1C88xxx and the function option document file that is necessary to generate IC mask patterns.

Segment option generator (winsog.exe)

This tool creates an ICE segment option setup file after selecting the segment options of the S1C88xxx and the segment option document file that is necessary to generate IC mask patterns. The winsog is used only for the model that has segment options.

Program unused area filling utility (fil88xxx.exe)

This tool extracts the built-in ROM area from a program data HEX file and fills unused areas in the built-in ROM with FFH. It also sets a system code to the system-reserved area. This processing must be performed before debugging the program with the ICE as well as before generating a mask data with winmdc.

Mask data checker (winmdc.exe)

This tool checks the data in development-completed program file and option document files to create the mask data file that will be presented to Seiko Epson.

Refer to Chapters 6 through 12 for details of the tools in the Development tool chain.

Debug tool

db88 debugger (ice88ur.exe)

Controls the ICE (S5U1C88000H5) provided as a hardware tool for the S1C88 Family to debug programs. Commands that are used frequently, such as break and step, are registered on the tool bar, minimizing the necessary keyboard operations. Moreover, sources, registers, and command execution results can be displayed in multiple windows, with resultant increased efficiency in the debugging tasks. Refer to Chapter 13 for details of the db88 debugger.

ice88ur debugger (ice88ur.exe)

Controls the ICE (S5U1C88000H5) provided as a hardware tool for the S1C88 Family to debug programs. The operations are described in a Windows help file (.hlp) that can be opened from the start menu. (The help file in English can also be opened from the menu / tool bar in ice88ur.)

PROM writing tool

ROM writer control software

Controls the dedicated PROM writer to write data to the PROM in the built-in Flash EEPROM microcomputer. A different tool and firmware are provided for each microcomputer model and each type of PROM writer. Refer to the technical manual of each built-in Flash EEPROM microcomputer for PROM writers and how to write data.

CHAPTER 2 INSTALLATION

2.1 Package Components

The S1C88 Family Integrated Tool Package contains the items listed below. When it is unpacked, make sure that all items are supplied.

1. CD-ROM (Tools and PDF manuals are included)	One
2. Warranty card	One each in English and Japanese
3. Registration card	One each in English and Japanese

2.2 Operating Environment

For each tool to be used, the following operating environment is required:

Personal computer

IBM PC / AT or fully compatibles that can run the system software listed below. A personal computer using Pentium 200 MHz or greater as the CPU and incorporating 64 MB or more of RAM is recommended. Installation requires a CD-ROM drive.

To use the optional ICE (S5U1C88000H5), the personal computer also requires a USB port and Windows 2000 or Windows XP.

Display

A 800 × 600 dots display unit or higher is required.

System software

Each tool is designed to run under Microsoft Windows 2000 and Windows XP (in English or Japanese).

To use the optional ICE (S5U1C88000H5), Windows 2000 or Windows XP is necessary.

Other

To debug the target program using the in-circuit emulator system, the optional ICE (S5U1C88000H5) and a Peripheral Circuit Board (S5U1C88xxxP) are needed as the hardware tools.

2.3 *Installation Method*

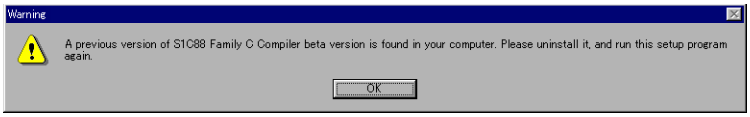
To install the development tools, use the installer (Setup.exe) on the CD-ROM included with the package.

To install the tools

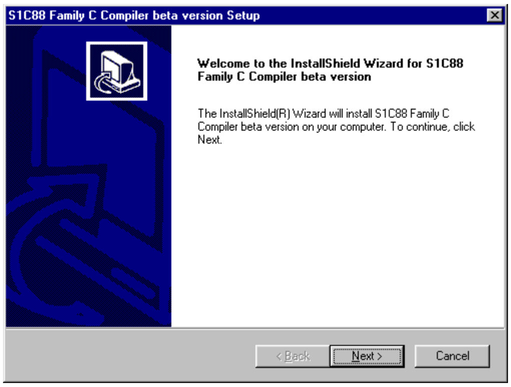
(1) Start Windows 2000 or Windows XP. If the OS is already active, close active programs.

(2) Insert the CD-ROM into the drive and display the contents.

(3) Double-click Setup.exe.
When old-version tools are installed, the installer displays a warning message and stops installation. In this case, uninstall the old-version tools and then run the installer again.



A previous version of S1C88 Family C Compiler beta version is found in your computer. Please uninstall it, and run this setup program again.

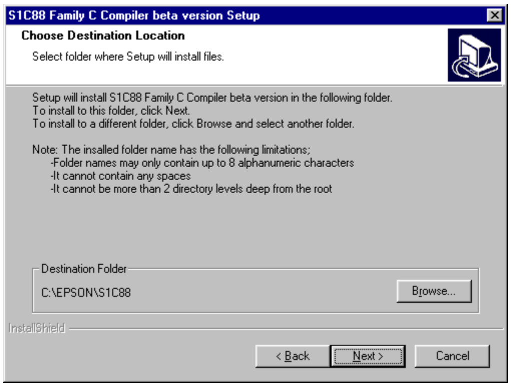


Welcome to the InstallShield Wizard for S1C88 Family C Compiler beta version

The InstallShield(R) Wizard will install S1C88 Family C Compiler beta version on your computer. To continue, click Next.

Welcome to ...
The install wizard starts and displays the welcome dialog box.

(4) Click on the [Next>] button to proceed.



Choose Destination Location

Select folder where Setup will install files.

Setup will install S1C88 Family C Compiler beta version in the following folder.
To install to this folder, click Next.
To install to a different folder, click Browse and select another folder.

Note: The insalled folder name has the following limitations;
-Folder names may only contain up to 8 alphanumeric characters
-It cannot contain any spaces
-It cannot be more than 2 directory levels deep from the root

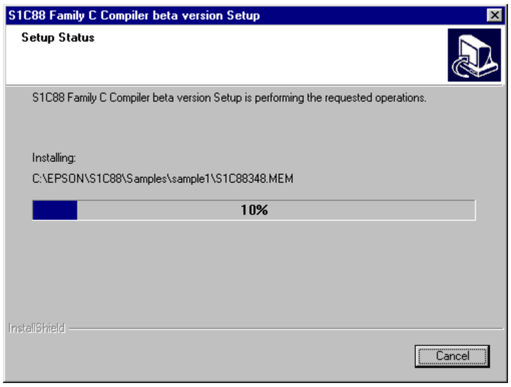
Destination Folder
C:\EPSON\S1C88

Choose Destination Location
A dialog box for specifying the folder in which to install the tools appears.

(5) If you do not wish to change the default settings, simply click the [Next>] button to execute installation.

To install in another folder
Click [Browse...] to bring up the [Choose Folder] dialog box. From this dialog box, enter the path or select the folder in which to install the tools. Click the [OK] button to finish folder selection and then click the [Next>] button.

Note: When installing the tools to a folder other than default, be aware that the folder must satisfy the following requirements:
- *The folder name must be 8 letters or less.*
- *The folder name cannot contain any spaces.*
- *When selecting a sub-directory, it must be located within two levels from the root directory.*



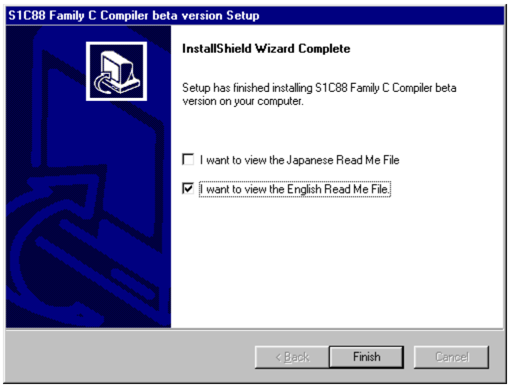
Setup Status

S1C88 Family C Compiler beta version Setup is performing the requested operations.

Installing:
C:\EPSON\S1C88\Samples\sample1\S1C88348.MEM

10%

The installer will start installing the tools.



S1C88 Family C Compiler beta version Setup

InstallShield Wizard Complete

Setup has finished installing S1C88 Family C Compiler beta version on your computer.

□ I want to view the Japanese Read Me File

☑ I want to view the English Read Me File.

InstallShield Wizard Complete

(6) Click [Finish] to terminate the installer.

"401Comupd.exe" may be executed according to the system condition.

To end installation

All dialog boxes that appear during installation have a [Cancel] button. To prematurely terminate installation, click [Cancel] in the dialog box when it is displayed.

To uninstall

To uninstall the installed tools, use "Add / Remove Programs" on the Control Panel.

2.4 Directories and Files after Installation

The installer copies the following files in the specified directory (default is "C:\EPSON\S1C88\"):

```
[EPSON\S1C88]
        README_E.TXT              ... ReadMe text file (English)
        README_J.TXT              ... ReadMe text file (Japanese)
        ADDPATH.BAT               ... Batch file for environment setup

        [\BIN]                    ... S1C88 Family C Compiler Tools
            WB88.EXE              ... Work bench
            C88.EXE               ... C compiler
            AS88.EXE              ... Assembler
            LK88.EXE              ... Linker
            LC88.EXE              ... Locator
            ALC88.EXE             ... Advanced locator
            CC88.EXE              ... Control program
            MK88.EXE              ... Make program
            AR88.EXE              ... Library maintainer
            PR88.EXE              ... Object reader
            SY88.EXE              ... Symbolic table file generator
            ICE88UR.EXE           ... S5U1C88000H control software
            ICE88UR.HLP           ... S5U1C88000H help file
            . . .                 ... Other related files

            [\SAP]                ... S1C88 Family Structured Assembler Tools
                SAP88.EXE         ... Preprocessor
                ASM88.EXE         ... Assembler
                LINK88.EXE        ... Linker
                HEX88.EXE         ... HEX converter
                REL88.EXE         ... Symbol information generator
                SYM88.EXE         ... Symbolic table file generator
        [\DB88]                   ... DB88 debugger directory
            DB88.EXE              ... DB88 debugger
            DEFAULT.PAR           ... Default parameter file
            . . .                 ... Other related files
        [\DEV]
            [\BIN]                ... S1C88 Family Development Tool for Windows
                WINFOG.EXE        ... Function option generator
                WINSOG.EXE        ... Segment option generator
                WINMDC.EXE        ... Mask data checker

            [\88xxx]              ... Model-dependent files
                S1C88xxx.CPU      ... Locator description files
                S1C88xxx.DSC
                S1C88xxx.MEM
                FIL88xxx.EXE      ... Program unused area filling utility
                S1C88xxx.ini      ... Device information definition file
                88xxx.PAR         ... ICE parameter file
                t88xxx.psa        ... ICE self-diagnostic files
                t88xxx.fsa
                t88xxx.fdc
```

```
[\DOC]
    [\ENGLISH]                  ... Document folder (English)
        REL_xxxx_E.TXT          ... Tool release note
        TBD_E.PDF               ... Manual (PDF)
        TBD_E.PDF               ... Quick reference (PDF)
        [\HARD]                 ... Hardware tool document folder (English)
            PRC_COMMON_E.PDF     ... Standard peripheral circuit board manual (PDF)
            ICE88UR_SETUP_E.PDF... ICE manual (PDF)

    [\JAPANESE]                 ... Document folder (Japanese)
        REL_xxxx_J.TXT          ... Tool release note
        TBD_J.PDF               ... Manual (PDF)
        TBD_J.PDF               ... Quick reference (PDF)
        [\HARD]                 ... Hardware tool document folder (Japanese)
            PRC_COMMON_J.PDF     ... Standard peripheral circuit board manual (PDF)
            ICE88UR_SETUP_J.PDF ... ICE manual (PDF)

[\ETC]                          ... Default locator description files
    S1C88.DSC
    MK88.MK
    S1C88.CPU
    S1C88.MEM

[\ICE]
    [\FPGA]
        C88xxx.MOT              ... FPGA data for standard peripheral circuit board

[\INCLUDE]                      ... C header files

[\LIB]                          ... C library files
    [\LIBCC]                    ... Library objects for compact code model
    [\LIBCD]                    ... Library objects for compact data mode
    [\LIBCL]                    ... Library objects for large mode
    [\LIBCS]                    ... Library objects for small mode
    [\SRC]                      ... Library source files

[\SAMPLES]                      ... Sample program sources
                                    Refer to ApplicationNote_J(E).PDF located in this folder for the contents of the sample
                                    programs.

[\WRITER]
    [\8xxxx]  (Flash microcomputer name)
        [\URW2]
            RW8xxxxx.EXE        ... Universal ROM Writer II control software
            8xxxxx.FRM          ... Firmware
        [\OBPW]
            OBW8xxxx.EXE        ... On-board Programming ROM Writer control software
            RW8xxxx.INI         ... Device information setup file
        [\MPRW]
            G8xxxxxx.EXE        ... Multiple-Programming ROM Writer control software
    :                           *  Refer to the technical manual for details of the ROM Writer and PROM programming.
```

Online manual in PDF format

The online manuals are provided in PDF format, so Adobe Acrobat Reader Ver. 4.0 or later is needed to read it.

Files for future release models

The files for future release models will be provided in the Microcomputer User's Site of Seiko Epson. Refer to the Readme file included in the package for installation.

2.5 Environment Settings

The following environment variables must be configured for the tools in this package:

```
SET PATH=C:\EPSON\S1C88\BIN;%PATH%
SET C88INC=C:\EPSON\S1C88\INCLUDE
SET C88LIB=C:\EPSON\S1C88\LIB
```

Run addpath.bat in which the above commands are described before using the tools.
When you select another directory at installation, "EPSON\S1C88\" shown above is changed to that directory name.
However, wb88 automatically configures the environment variables at start-up, so it is not necessary to run addpath.bat when invoking the tools from wb88.

CHAPTER 3 WORK BENCH

This chapter describes the functions and operating method of the Work Bench wb88.

3.1 Features

The Work Bench wb88 provides an integrated operating environment ranging from editing source files to debugging. Its functions and features are summarized below:

- Source edit function that supports tag jump from error messages using a user's editor.
- Allows simple management of all necessary files and information as a project.
- General make process to invoke necessary tools and to update the least necessary files.
- Supports all options of the S1C88 Family C compiler tool chain and invocation of each tool.
- Windows GUI interface for simple operation.

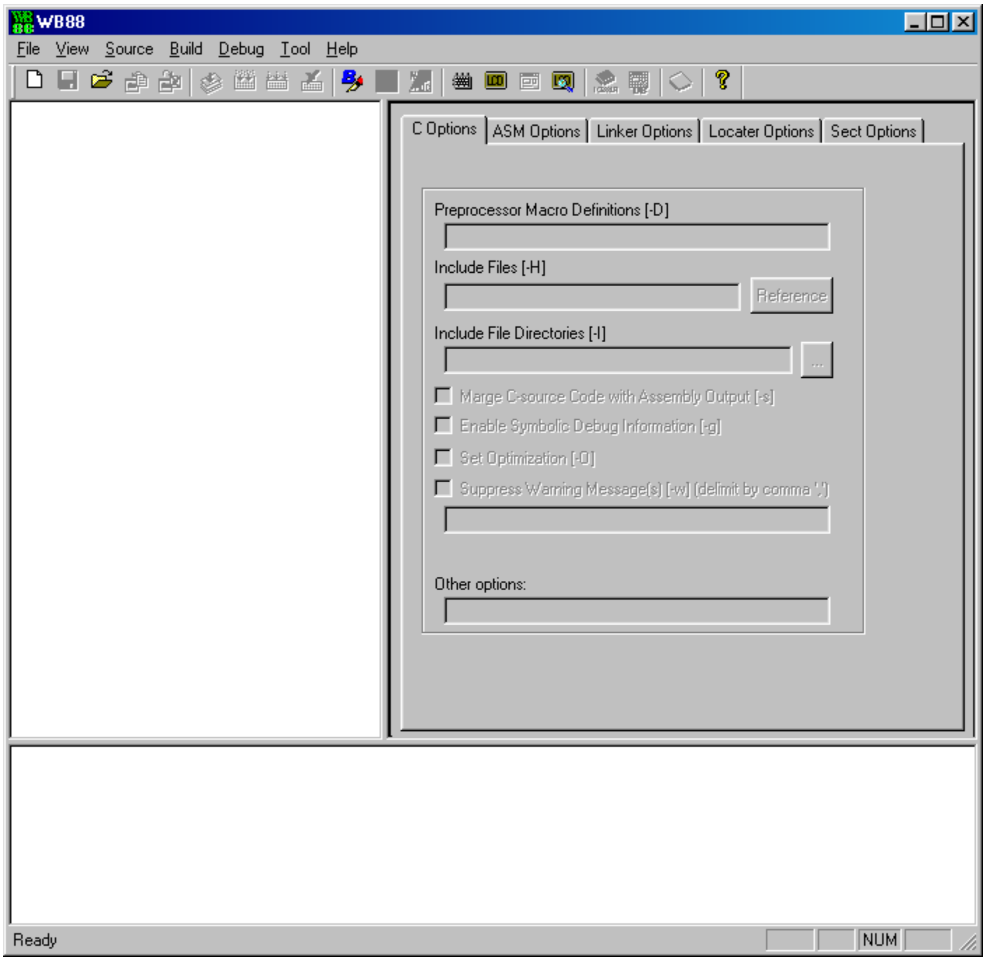
3.2 Starting Up and Terminating the Work Bench

To start up the work bench

wb88.exe

Double-click on the wb88.exe icon.

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



Preprocessor Macro Definitions [-D]

Include Files [-H]

Include File Directories [-I]

□ Marge C-source Code with Assembly Output [-s]

□ Enable Symbolic Debug Information [-g]

□ Set Optimization [-O]

□ Suppress Warning Message(s) [-w] (delimit by comma ',')

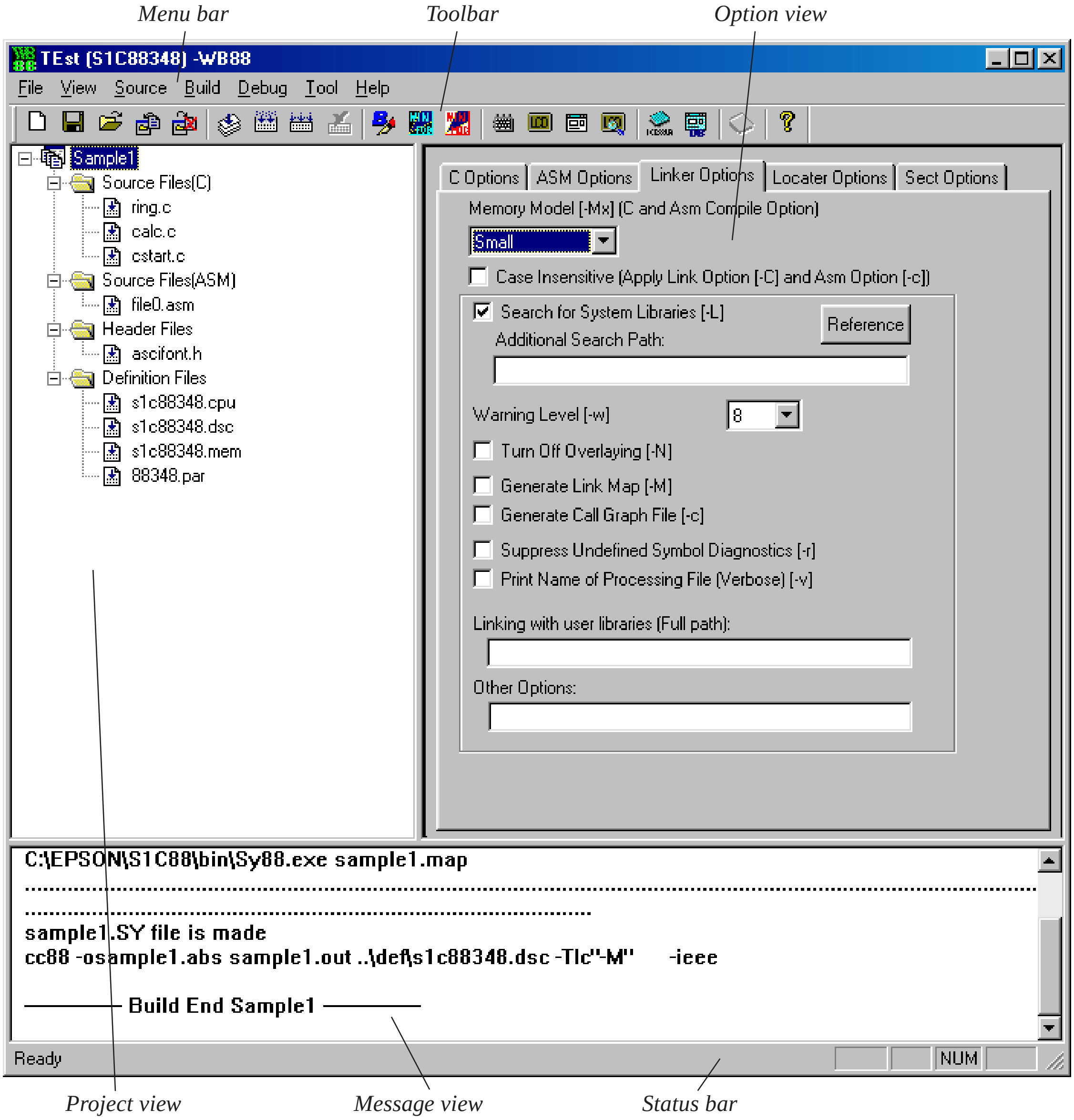
Other options:

To terminate the work bench

Select [Exit] from the [File] menu.

3.3 Work Bench Windows

The work bench window is configured with Project view, Option view and Message view.



Memory Model [-Mx] (C and Asm Compile Option): Small

☐ Case Insensitive (Apply Link Option [-C] and Asm Option [-c])

☑ Search for System Libraries [-L]

Additional Search Path:

Warning Level [-w]: 8

☐ Turn Off Overlaying [-N]

☐ Generate Link Map [-M]

☐ Generate Call Graph File [-c]

☐ Suppress Undefined Symbol Diagnostics [-r]

☐ Print Name of Processing File (Verbose) [-v]

Linking with user libraries (Full path):

Other Options:

```
C:\EPSON\S1C88\bin\Sy88.exe sample1.map
..........................................................................................................................................
...........................................................................
sample1.SY file is made
cc88 -osample1.abs sample1.out ..\def\s1c88348.dsc -Tlc"-M"     -ieee

———— Build End Sample1 ————
```

Each view area can be resized by dragging the boundary. A standard scroll bar appears if the display contents exceed the view area. Use it to scroll the display contents. The arrow keys can also be used.

Project view

This area shows the currently opened work space folder and lists all the files that can be edited by the user in the project, with a structure similar to Windows Explorer.

The file list is classified into five nodes:

• Project	Project name (work space folder name)
• Source Files (C)	C source files (.c)
• Source files (ASM)	Assembly source files (.asm)
• Header Files	Header files (.h / .inc)
• Definition Files	Various device information definition files (.cpu / .dsc / .mem / .par) that allow user to edit

Double-clicking a source file icon invokes the specified editor to open the source file. Definition Files are displayed only when the check box [Disable Making DELFEE] of the section editor is selected.

Option view

This area displays the selected options of the C compiler, assembler, linker, locator and segment editor, and also allows option selection. The option view changes its display contents according to the selection in the project view (whether node or file) as well as clicking a tool name tab. Refer to Section 3.9 for details.

Message view

This area displays the messages delivered from the executed tools in a build or compile process. Double-clicking a syntax error message with a source line number displayed in this window invokes the specified editor. The editor opens the corresponding source and displays the source line in which the error has occurred (available when an editor with the tag jump function that can be specified by wb88 is used).

Menu bar

Refer to Section 3.5.

Tool bar

Refer to Section 3.4.

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

The tool bar can be changed to vertical position by dragging it towards the left edge or right edge of the window. It can also be made a floating window by dragging it outside the tool bar area.

Status bar

Shows help messages when the mouse cursor is placed on a menu item or a button.

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

3.4 Toolbar and Buttons

The toolbar has the following buttons:

[New Project] button
Creates a new project.

[Save Project] button
Saves the project being edited. The file will be overwritten. This button becomes inactive if a project is not opened.

[Insert a file] button
Inserts the specified source / header file into the current opened project. This button becomes inactive if a project is not opened.

[Remove a file] button
Removes the selected file from the project.

[Open] button
Opens a document. A dialog box will appear allowing selection of the file to be opened. When a source or header file is selected, the specified editor activates and opens the file.

[Compile/Assemble] button
Compiles or assembles the source file selected in the option view according to the source format.

[Build] button
Builds the currently opened project using a general make process.

[Rebuild] button
Builds the currently opened project. All the source files will be compiled / assembled regardless of whether they are updated or not.

[Stop Build] button
Stops the build process being executed.

[BMPUtil] button
Invokes the bitmap utility BmpUtil.

[WinFOG] button
Invokes the function option generator winfog.

[WinMDC] button
Invokes the mask data checker winmdc.

[PrtUtil] button
Invokes the port setting utility PrtUtil.

[LCDUtil] button
Invokes the LCD panel customize utility LCDUtil.

[Sim88] button
Invokes the simulator Sim88.

[AutoEva] button
Invokes the auto evaluation system AutoEva.

[ICE88UR] button
Invokes the ice88ur debugger.

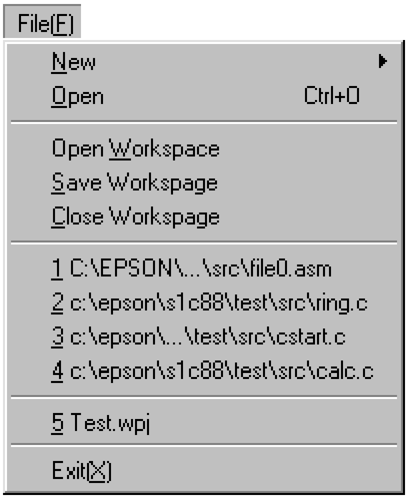
[DB88] button
Invokes the db88 debugger.

[ROM Writer] button
Invokes the on-board ROM writer control software.

[About] button
Displays the version of wb88.

3.5 Menus

3.5.1 [File] Menu


File(F)
New
Open Ctrl+O
Open Workspace
Save Workspage
Close Workspage
1 C:\EPSON\...\src\file0.asm
2 c:\epson\s1c88\test\src\ring.c
3 c:\epson\...\test\src\cstart.c
4 c:\epson\s1c88\test\src\calc.c
5 Test.wpj
Exit(X)

The file names listed in this menu are recently used source and project files. Selecting one opens the file.

[New - C Source File]
Creates a new C source file. When a file name is entered in the displayed dialog box, the specified editor activates and opens a new document. The created source file is inserted into the currently opened project (Source Files (C) node in the project view).

[New - Asm Source File]
Creates a new assembly source file. When a file name is entered in the displayed dialog box, the specified editor activates and opens a new document. The created source file is inserted into the currently opened project (Source Files (ASM) node in the project view).

[New - Header File]
Creates a new header file. When a file name is entered in the displayed dialog box, the specified editor activates and opens a new document. The created source file is inserted into the currently opened project (Header Files node in the project view).

[New - Project]
Creates a new project.

[Open] ([Ctrl]+[O])
Opens a source file, header file or project file. A dialog box will appear allowing selection of the file to be opened. When a source or header file is selected, the specified editor activates and opens the file.

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

[Save Workspace]
Saves the currently opened project.

[Close Workspace]
Closes the currently opened project.

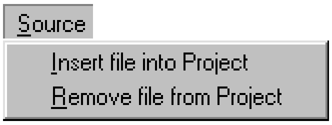
[Exit]
Terminates wb88.

3.5.2 [View] Menu

[Tool Bar]
Shows or hides the tool bar.

[Status Bar]
Shows or hides the status bar.

3.5.3 [Source] Menu



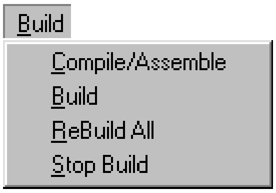
[Insert file into Project]

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

[Remove file from Project]

Removes the source file selected in the Project view from the currently opened project. The actual file is not deleted.

3.5.4 [Build] Menu



[Compile/Assemble]

Compiles or assembles the source file selected in the Project view according to the source format.

[Build]

Builds the currently opened project using a general make process.

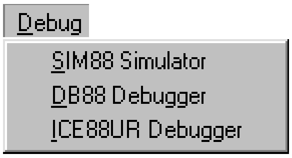
[ReBuild All]

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

[Stop Build]

Stops the build process being executed.

3.5.5 [Debug] Menu



[SIM88 Simulator]

Invokes the Sim88 simulator.

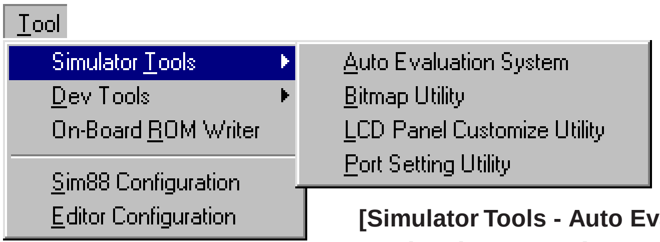
[DB88 Debugger]

Invokes the db88 debugger.

[ICE88UR Debugger]

Invokes the ice88ur debugger.

3.5.6 [Tools] Menu



[Simulator Tools - Auto Evaluation System]

Invokes the auto evaluation system AutoEva.

[Simulator Tools - Bitmap Utility]

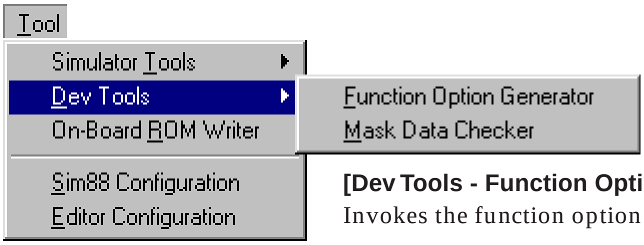
Invokes the bitmap utility BmpUtil.

[Simulator Tools - LCD Panel Customize Utility]

Invokes the LCD panel customize utility LCDUtil.

[Simulator Tools - Port Setting Utility]

Invokes the port setting utility PrtUtil.



[Dev Tools - Function Option Generator]

Invokes the function option generator winfog.

[Dev Tools - Mask Data Checker]

Invokes the mask data checker winmdc.

[On-Board ROM Writer]

Invokes the on-board ROM writer control software.

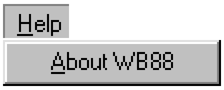
[Sim88 Configuration]

Displays a dialog box for setting the path to the simulator Sim88.exe.

[Editor Configuration]

Displays a dialog box for setting the editor path and the command line options.

3.5.7 [Help] Menu



[About WB88]

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

3.6 Project and Work Space

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

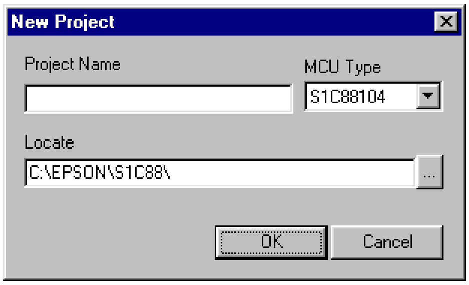
3.6.1 Creating a New Project

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

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

   [New Project] button

   The [New Project] dialog box appears.



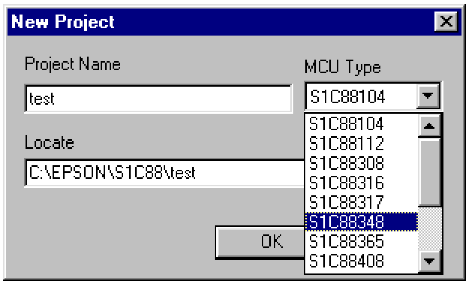
New Project

Project Name:

MCU Type: S1C88104

Locate: C:\EPSON\S1C88\

2. Enter a project name, select a device name and a directory for saving the project, and then click [OK].



New Project

Project Name: test

MCU Type: S1C88104

S1C88104
S1C88112
S1C88308
S1C88316
S1C88317
S1C88348
S1C88365
S1C88408

Locate: C:\EPSON\S1C88\test

∗ The [MCU Type] box lists the device names that exist in the "dev" directory.

The work bench creates the folder (directory) specified in the [Locate] box as a work space, and creates the project file (<project name>.wpj) and the following folders in the folder.
If a folder which has the same name as that of a specified one already exists in the specified location, the work bench uses the folder as the work space.
The specified project name will also be used for the absolute object and other files.

Folders created in the work space

def: Folder in which advanced locator definition files and various other definition files are saved. When a new project is created, a definition file that will be used as a template is copied into this folder. This file can be reused simply by making the necessary changes, if any.

obj: Folder in which intermediate files generated during building are saved.

src: Folder in which source files and header files created from wb88 are saved. (Source files in other folders are not copied to this folder, even when they have been added to a project.)

tmp: Folder in which temporary files created during building or tool execution are saved.

For more information on the file types placed in each folder, refer to Section 3.12, "File List".

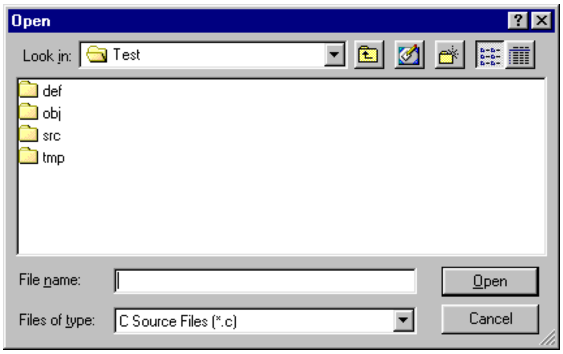
3.6.2 Inserting Sources into a Project

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

1. [Source | Insert file into Project] menu item or [Insert a file] button

   [Insert a file] button

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



Look in: Test

Files of type: C Source Files (*.c)

After specifying the file format (C source, assembly source, or header file), select a file and click the [Open] button. The selected file is added to the project and displayed in project view.

Note: *Reference information on the selected file is registered to the project. Since files are not copied into the work space, do not move a file after adding it to the project. If a file is moved, remove the file from the project (see the section below), then add it back to the project again.*

2. [File | New] menu item

   If a new source file or header file is created with this menu command, the file is automatically added to the project that is currently open. For more information on creating new source and header files, refer to Section 3.7.2.

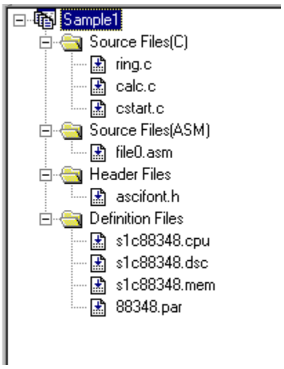
   The newly created files are added to the project and displayed in the Project view.

3.6.3 Removing a Source from the Project

To remove a source or header file from the project, select the file in the Project view and then select [Remove file from Project] from the [Source] menu, click the [Remove a file] button or press the [Delete] key. This removes only the file information, and does not delete the actual file.

[Remove a file] button

3.6.4 Project View



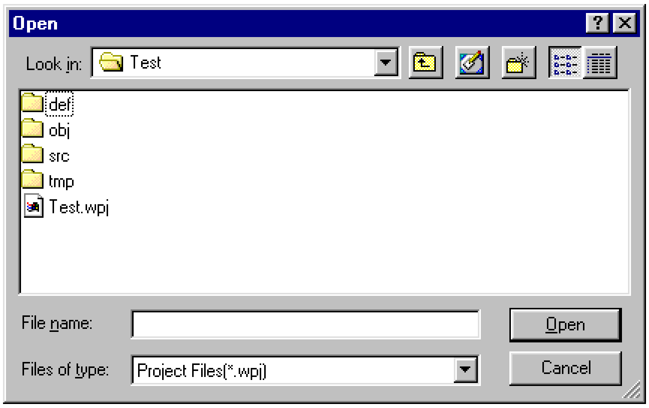
The Project view shows the work space folder and the files that can be edited, such as source, header and definition files, included in the project that has been opened.

When a file icon or file name is double-clicked, the specified editor activates and opens the file. Notepad in Windows is set as the default editor. It can be changed by selecting [Editor Configuration] from the [Tool] menu.

Note: *Note that the list in the [project] window is not the actual directory structure.*

3.6.5 Opening and Closing a Project

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



Look in: Test

File name:

Files of type: Project Files(*.wpj)

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

The project file can also be opened by selecting [Open] from the [File] menu or clicking the [Open] button.

[Open] button

In this case, choose the file type as Project Files (*.wpj) in the file open dialog box.

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

3.6.6 Saving the Project

To save the currently edited project file, select [Save Workspace] from the [File] menu or click the [Save Project] button.

[Save Project] button

In addition, if one of the following operations is performed before the project is saved, a dialog box is displayed to prompt for save confirmation. This allows the project to be saved here.

- Open the project (by selecting the project with [Open Workspace] or [Open] from the [File] menu)
- Close the project ([Close Workspace] on the [File] menu)
- Create a new project ([New | Project] on the [File] menu)
- Compile or assemble ([Compile / Assemble] on the [Build] menu)
- Build ([Build] on the [Build] menu)
- Rebuild ([ReBuild All] on the [Build] menu)

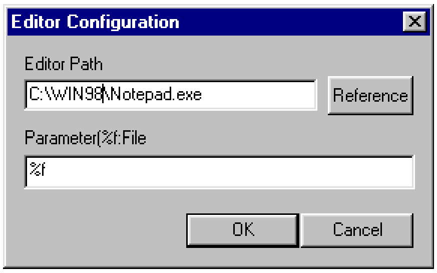
3.7 Creating/Editing Source Files

Although the Work Bench itself does not include a source editor, it can invoke a specified editor and pass file information or line number information to the editor. This function makes it possible to create and edit sources, as well as tag jump from error messages.

3.7.1 Specifying an Editor

When a source / header file is newly created or opened, or when a file name listed in the Project view is double clicked, the Work Bench invokes an editor and passes file information to it. The default editor is the Windows Notepad application. To select another editor:

1. Select [Editor Configuration] from the [Tool] menu. The [Editor Configuration] dialog box shown below is displayed:



Editor Path: C:\WIN98\Notepad.exe

Parameter(%f:File: %f

Enter the following information in this dialog box:

[Editor Path]

Enter the path to the editor used or select an editor from the file select dialog box displayed by clicking the [Reference] button.

[Parameter]

Enter the normal representation of command line options to specify a file name and line number (for tag jump) when invoking the editor. The "%f" and "%l" are replaced with a file name and a line number, respectively, before being sent to the editor. In the case of the default setting, Notepad is invoked using the following command line.

`C:\Win98\Notepad.exe` <specified filename>

For example, if the editor requires specifying a file name in the same way as for Notepad and specifying a "/j <line number>" option for tag jump in front of the file name, set the parameter as follows:

```
/j%1 %f
```

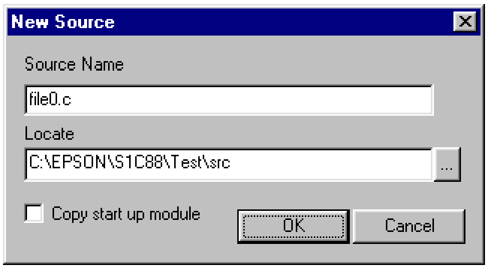
Note: In the default Notepad application, the tag jump function cannot be used.

2. Click the [OK] button. The editor used is changed.

3.7.2 Creating a New Source or Header File

To create a new source or header file:

1. Select [New | C Source File], [New | Asm Source File] or [New | Header File] from the [File] menu. The [New Source] dialog box appears.

   New Source

   Source Name: file0.c

   Locate: C:\EPSON\S1C88\Test\src

   □ Copy start up module

   Example when [C Source File] is selected.

   [Source Name]
   Enter a source file name. Depending on the source type, use one of the following extensions.

.c	C source file
.asm	Assembly source file
.h	Header file
.inc	Include file

   [Locate]
   Enter a directory in which to create the source file. Select directories from the dialog box displayed by clicking the [...] button. The src folder in the work space is displayed as the default location. Use this folder unless you wish to select another folder for a specific reason.

   [Copy start up module]
   This check box is displayed only when C source file is selected. Leave it checked to copy code from the C startup module stationery file into the C source file to be created. The stationery file is cstart.c in the \EPSON\S1C88\LIB\SRC folder.

2. Click the [OK] button.
   This creates a specified source file, and the selected editor starts to open that file. The created file is also added to the project tree displayed in the Project view.

3. In the editor, enter the source codes and save the codes entered.

3.7.3 Editing Files

Correct or print the source file using the selected editor. Use one of the following two methods to open the source file:

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

   [Open] button

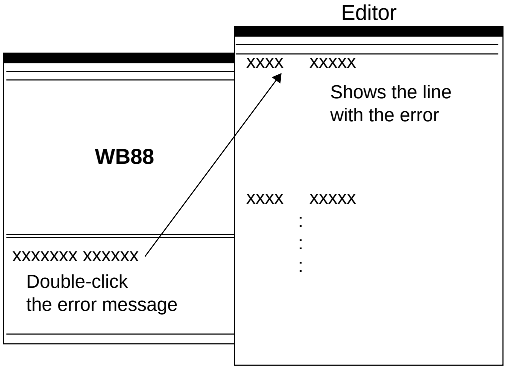
   A [Open] dialog box appears. After specifying the file format (C source, assembly source, or header file), select a file and click the [Open] button.

2. Double-click on the file name displayed in the Project view.
   You can also open a definition file from the [Definition Files] list.

In either case, the selected file is opened in the selected editor. In the editor, perform the necessary work.

3.7.4 Tag Jump Function

If a syntax error occurs during compiling or assembling a source file, an error message is displayed in the Message view. If the error message includes a source line number, double-click the message to open the relevant source file in the editor and to jump to the source line with the error.



Tag jump

Note: Before using the tag jump function, you must ascertain that your editor supports command line-based tag jumps, and that the command line option is correctly set in [Tool | Editor Configuration]. (This function cannot be used with the default Notepad.)

3.8 Build Task

The [Build] menu or with the toolbar button is used to build a project using the C compiler tool chain (i.e., to generate an executable object file from the source file) and to execute compile / assemble operations from the Work Bench. For detailed information on each tool, refer to the "S5U1C88000C Manual I".

3.8.1 Preparing a Build Task

Before starting a build task, necessary source files should be prepared and tool options should be configured.

1. Create a new project. (Refer to Section 3.6.1.)
2. Create source files and add them into the project. (Refer to Sections 3.7 and 3.6.2.)
3. When alc88 is used, edit the advanced locator definition file using the section editor (Refer to Section 3.9.5.)
   When lc88 is used, edit the locator description files (Refer to Section 3.7.3 and "S5U1C88000C Manual I".)
4. Select tool options (Refer to Section 3.9.)

3.8.2 Building an Executable Object

To generate an executable object:

1. Open the project file.
2. Select [Build] from the [Build] menu or click the [Build] button.

   [Build] button

The work bench generates a make file according to the source files in the project and the tool options set by the user. This file is used to control invocation of tools.
First, the make process invokes the C compiler for each source file to be compiled. If the latest assembly source file exists in the work space, the corresponding C source file is not compiled to reduce process time.
Likewise, the assembler is invoked to generate relocatable object files.
Next, the linker is invoked to generate an absolute object file.
Finally, the advanced locator or the locator* is invoked to generate an executable object file.

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

[Rebuild] button

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

[Stop Build] button

* Selecting Advanced Locator alc88 or Locator lc88

Advanced locator alc88 and locator lc88 both have the function to relocate linked relocatable objects to absolute addresses in memory. Either type of locator can be used by selecting or deselecting the check box [Disable branch optimize] (displayed on the [Locator Options] tab screen) for locator options.

When [Disable branch optimize] = OFF (default), alc88 is executed.
When [Disable branch optimize] = ON, lc88 is executed.

The table below summarizes the differences between alc88 and lc88.

Table 3.8.2.1 Differences between alc88 and lc88

Item	Advanced locator alc88	Locator lc88
Definition file	Advanced locator definition file (.inf)	Locator description files (DELFEE) (.dec, .mem, .cpu)
How definition files are created	The section editor of wb88 is used (so there is no need to learn DELFEE).	The section editor of wb88 is used or the user creates files in DELFEE language.
CARL instruction branching optimization function	Available	Not available

Except when necessary to use the existing locator description files, such as when upgrading application versions, we recommend the use of alc88 with a branching optimization function.
See Section 3.9.5, "Section Editor", for details about and how to create definition files.

3.8.3 Running only the Compiler or Assembler

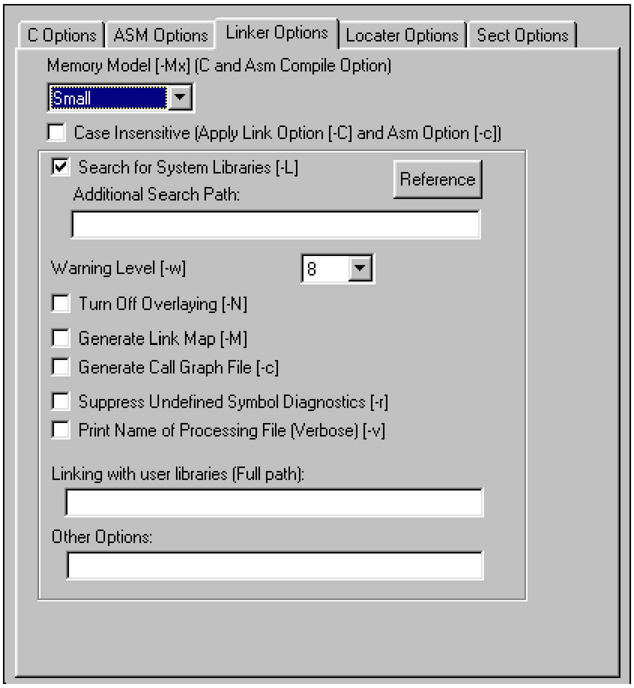
The source files can also be compiled or assembled individually. To invoke only the compiler or assembler, select the source file to compile or assemble from the Project view, then select [Compile / Assemble] from the [Build] menu or click the [Compile / Assemble] button.

[Compile/Assemble] button

Depending on the file type selected, either the compiler or the assembler is launched to process the file.

3.9 Setting Tool Options

Each tool executed in build task has options that can be specified at startup. The Work Bench allows you to select and set these options from the Option view.



Memory Model [-Mx] (C and Asm Compile Option): Small

□ Case Insensitive (Apply Link Option [-C] and Asm Option [-c])

☑ Search for System Libraries [-L]

Additional Search Path:

Warning Level [-w]: 8

□ Turn Off Overlaying [-N]

□ Generate Link Map [-M]

□ Generate Call Graph File [-c]

□ Suppress Undefined Symbol Diagnostics [-r]

□ Print Name of Processing File (Verbose) [-v]

Linking with user libraries (Full path):

Other Options:

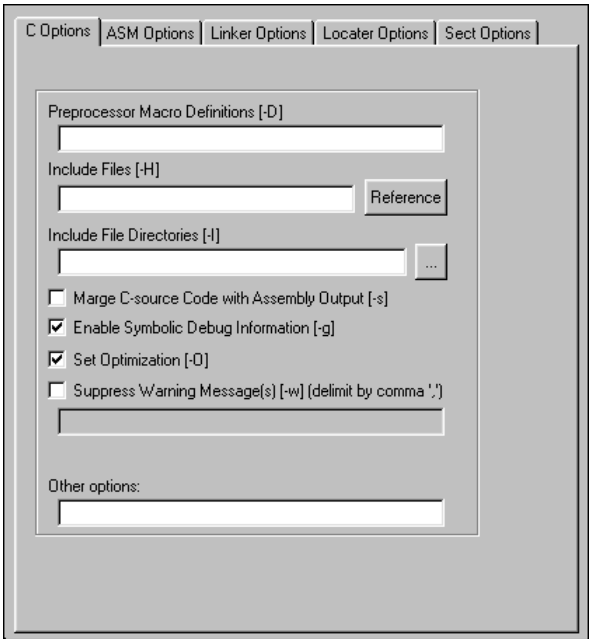
Option view

The options for each tool are displayed by clicking the tab for the intended tool name in the Option view. The tool options displayed vary, depending on the selection made in the Project view, as shown below:

1. Select a project name	Linker options are displayed.
2. Select [Source Files (C)]	Default compile options (which apply to all C sources) are displayed.
3. Select a C source file	Local compile options (which apply only to the selected C source) are displayed.
4. Select [Source Files (ASM)]	Default assemble options (which apply to all assembly sources) are displayed.
5. Select an assembly source file	Local assemble options (which apply only to the selected assembly source) are displayed.

The options for each tool selected in the Option view become effective the next time the tool is run.

3.9.1 Compiler Options



Preprocessor Macro Definitions [-D]

Include Files [-H]

Include File Directories [-I]

□ Merge C-source Code with Assembly Output [-s]

☑ Enable Symbolic Debug Information [-g]

☑ Set Optimization [-O]

□ Suppress Warning Message(s) [-w] (delimit by comma ',')

Other options:

In this screen, you can select the following compiler options:

Preprocessor Macro Definitions "-D*macro*[=*def*]" option of c88

Define the preprocessor macro. Enter in the text box in the following format:

macro name or *macro name = content of definition*

Include Files "-H *file*" option of c88

Specify the file name to be included before compiling. You can also display the files to include from the dialog box displayed by clicking the [Reference] button.

Include File Directories "-I*directory*" option of c88

Specify the directory in which to search for include files that have unspecified path names. You can also select this folder from the dialog box displayed by clicking the [...] button.

Merge C-source Code with Assembly Output "-s" option of c88

If this option is selected, C source codes are merged with the assembler output before being output.

Enable Symbolic Debug Information "-g" option of c88

If this option is selected, symbolic debug information is included in the output file.

Set Optimization "-O" option of c88

Selecting this option specifies "-O1" to optimize the code generated. Unchecking this option specifies "-O0", suppressing optimization of code generation.

Suppress Warning Message(s) "-w[*num*]" option of c88

Selecting this option suppresses compiler warning messages. To suppress all warning messages, leave the text box blank. To specify a specific warning message, enter the message number in the text box. To enter multiple numbers, separate each entry with a comma (,).

Other options

To specify other options (including those listed above), enter the desired option in this text box in command line format.

Notes to be observed when specifying compiler options

If both the -g option (Enable Symbolic Debug Information) and the -O1 option (Set Optimization) are selected, a -W555 warning message is output during compiling.

If the -O1 option is specified, the symbols written in the source may not actually be used to optimize the code. In this case, the debugging information for these symbols will not output to the .abs file, even if the -g option is specified.

Example:
```
int x, y, xy;
x = GLOBAL_X * 100;
y = GLOBAL_Y * 100;
xy = x * y;
```

In this example, since variable xy become nonexistent for optimization, the contents of xy cannot be referenced during debugging.

If the executable file is recreated by specifying the -O1 option (optimization ON) after evaluation of the executable file created with the -O0 option set (optimization OFF), program behavior cannot be assured. Be sure to reverify the executable file whenever it is recreated this way.

About options that are not displayed

The C compiler options not displayed in the Option view are handled as described below:

-e This option is used in internal processing.

-err C compiler messages are displayed in the message window and output to an error log file.

-f *file* This option conflicts with internal processing and cannot be used.

-o *file* The source file name is also used for the output file.

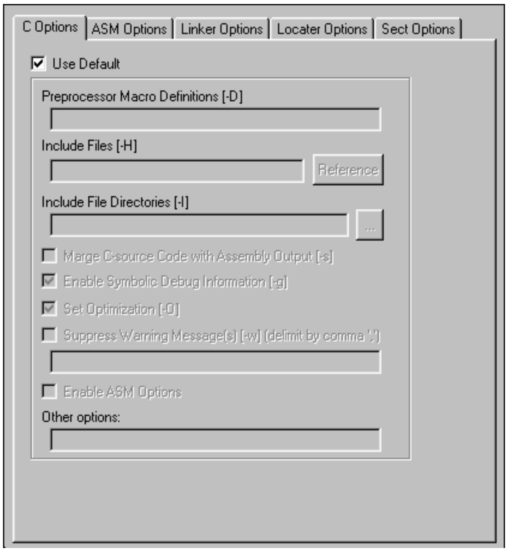
-V This option is not used in wb88.

-M{s | c | d | l} Specify this option in the linker option setup screen.

Default options and local options

If individual C source files are selected in the Project view, the option setup screen shows only the local options that are applied only to the selected C source file. If no specific file is selected in the Project view, or files other than individual C source files are selected, the default options that apply to all C source files are displayed.

If local options are displayed, the option setup screen will also display the [Use Default] button, as in the example shown below, to allow you to specify whether or not to apply the default options to the selected C source file.



☑ Use Default

Preprocessor Macro Definitions [-D]

Include Files [-H]

Include File Directories [-I]

☐ Marge C-source Code with Assembly Output [-s]

☑ Enable Symbolic Debug Information [-g]

☑ Set Optimization [-O]

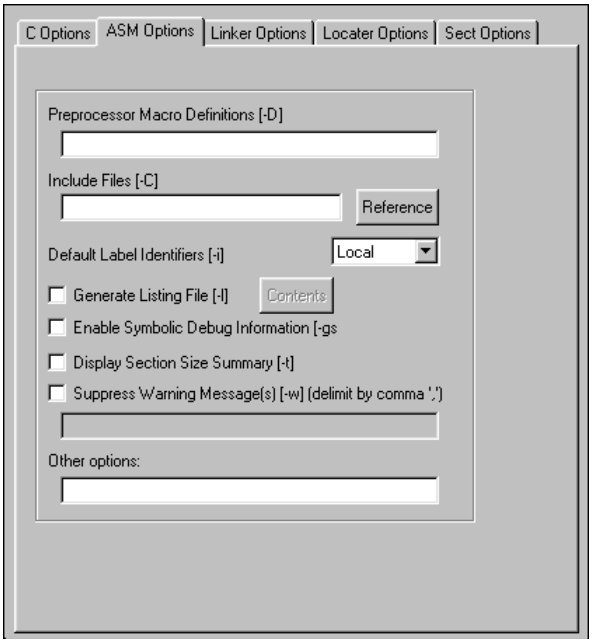
☐ Suppress Warning Message(s) [-w] (delimit by comma ',')

☐ Enable ASM Options

Other options:

To change the compile options for each C source, uncheck the [Use Default] button and set each option individually again.

3.9.2 Assembler Options



Preprocessor Macro Definitions [-D]

Include Files [-C]

Default Label Identifiers [-i] Local

□ Generate Listing File [-l]

□ Enable Symbolic Debug Information [-gs

□ Display Section Size Summary [-t]

□ Suppress Warning Message(s) [-w] (delimit by comma ',')

Other options:

This screen can be used to select the following assembler options:

Preprocessor Macro Definitions "-D*macro*[=*def*]" option of as88

Define the preprocessor macro. Enter in the text box in the following format:

macro name or *macro name = content of definition*

Include Files "-C *file*" option of as88

Specify the file name to be included before assembly. You can also select the files to include from the dialog box displayed by clicking the [Reference] button.

Default Label Identifiers "-i[l | g]" option of as88

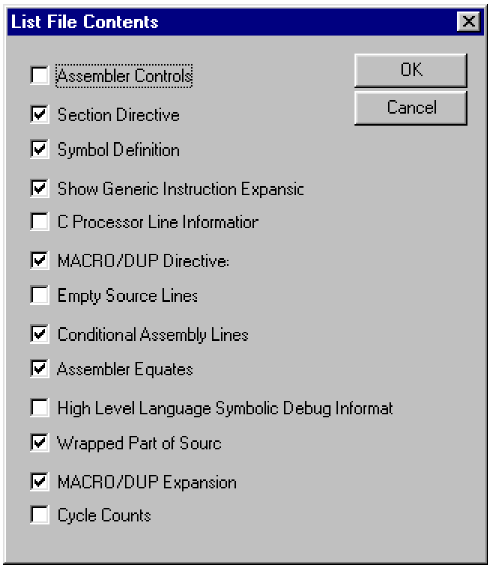
Specify the default label style as local or global. Select from the pull-down list.

Generate Listing File "-l" option of as88

If this option is selected, the assembler generates a list file.

Contents "-L" option of as88

This button is enabled by selecting [Generate Listing File]. Click this button to display the dialog box shown below appears, where you can select the source type line to be removed from the list file. The default option setup content is "-LcDElMnPQsWXy".



□ Assembler Controls

☑ Section Directive

☑ Symbol Definition

☑ Show Generic Instruction Expansic

□ C Processor Line Informatior

☑ MACRO/DUP Directive:

□ Empty Source Lines

☑ Conditional Assembly Lines

☑ Assembler Equates

□ High Level Language Symbolic Debug Informat

☑ Wrapped Part of Sourc

☑ MACRO/DUP Expansion

□ Cycle Counts

Enable Symbolic Debug Information "-gs" option of as88

If this option is selected, symbolic debug information is included in the output file.

Display Section Size Summary "-t" option of as88

If this option is selected, the assembler displays a section summary in message view when assembling.

Suppress Warning Message(s) "-w[*num*]" option of as88

If this option is selected, the assembler suppresses warning messages. To suppress all warning messages, leave the text box blank. To specify a specific warning message to be suppressed, enter the message number in the text box. Separate multiple numbers with a comma (,).

Other options

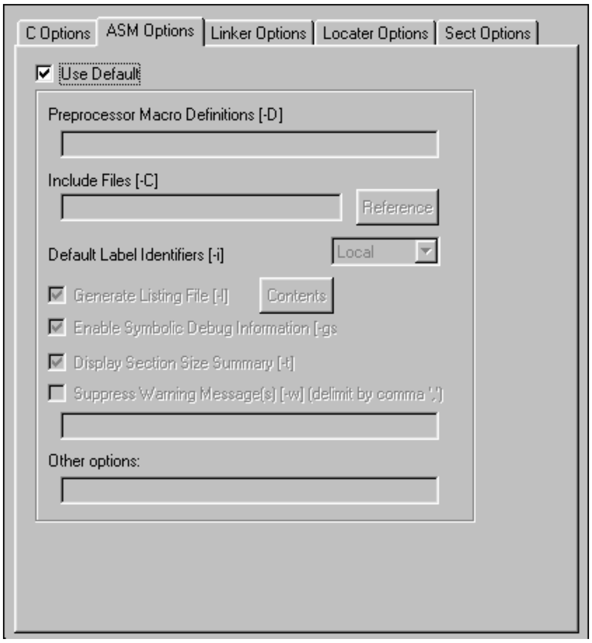
To specify other options (including those listed above), enter the desired option in this text box in command line format.

About options that are not displayed

The assembler options not displayed in the Option view are handled as described below:

-e This option is used in internal processing.
-err Assembler messages are displayed in the Message window and output to an error log file.
-f *file* This option conflicts with internal processing and cannot be used.
-o *file* The source file name is also used for the output file.
-V This option is not used in wb88.
-v This option is not used in wb88.
-c Specify this option in the linker option setup screen.
-M{s | c | d | l} Specify this option in the linker option setup screen.

Default options and local options



☑ Use Default

Preprocessor Macro Definitions [-D]

Include Files [-C]

Default Label Identifiers [-i] Local

☑ Generate Listing File [-l]

☑ Enable Symbolic Debug Information [-gs

☑ Display Section Size Summary [-t]

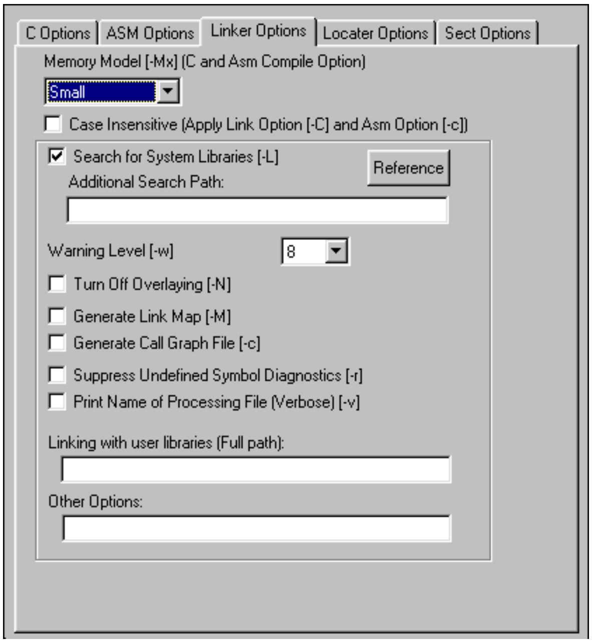
□ Suppress Warning Message(s) [-w] (delimit by comma ',')

Other options:

If individual assembly source files are selected in the Project view, the option setup screen shows the local options that are applied only to the selected assembly source file. If no specific file is selected in the Project view, or files other than individual assembly source files are selected, the default options that apply to all assembly sources are displayed.
If local options are displayed, the option setup screen will display the [Use Default] button, as in the example shown below, allowing you to specify whether or not to apply the default options to the selected assembly source file.

To change the assembler options for each assembly source, uncheck the [Use Default] button and set each option individually again.

3.9.3 Linker Options



Memory Model [-Mx] (C and Asm Compile Option): Small

□ Case Insensitive (Apply Link Option [-C] and Asm Option [-c])

☑ Search for System Libraries [-L]

Additional Search Path:

Warning Level [-w]: 8

□ Turn Off Overlaying [-N]

□ Generate Link Map [-M]

□ Generate Call Graph File [-c]

□ Suppress Undefined Symbol Diagnostics [-r]

□ Print Name of Processing File (Verbose) [-v]

Linking with user libraries (Full path):

Other Options:

This screen can be used to select the following options:

Memory Model "-M{s | c | d | l}" option of c88/as88

Select a memory model from Small, Compact code, Compact data, or Large. This setting is used during compiling and assembly.

Case Insensitive "-c" option of as88 and "-C" option of lk88

If this option is selected, the assembler and linker do not distinguish between uppercase and lowercase characters when assembling and linking.

Search for System Libraries "-L" option of lk88

If this option is selected, the linker searches for system libraries. If this option is unchecked, the linker does not search for system libraries.

If [Additional Search Path] is left blank after selecting this option, only the directory specified in the environment variable C88LIB is searched. To search other directories, enter the appropriate path in [Additional Search Path] or select a directory from the list displayed by clicking the [Reference] button.

Warning Level "-w *n*" option of lk88

Specify the level to which to suppress warning messages. Levels 0 to 9 can be selected from the pulldown list. The default setting is 8. Warning messages whose levels are higher than the selected value are not displayed.

Turn Off Overlaying "-N" option of lk88

Selecting this option disables overlaying.

Generate Link Map "-M" option of lk88

If this option is selected, the linker generates a link map file.

Generate Call Graph File "-c" option of lk88

If this option is selected, the linker generates an independent call graph file.

Suppress Undefined Symbol Diagnostics "-r" option of lk88

If this option is selected, the linker suppresses diagnosis of undefined symbols.

Print Name of Processing File (Verbose) "-v" option of lk88

If this option is selected, the linker displays the currently processed file name when linking.

Linking with user libraries

If there is any user library to link, enter the appropriate file name in this text box. To enter multiple files, separate each entry with a comma (,).

Other Options

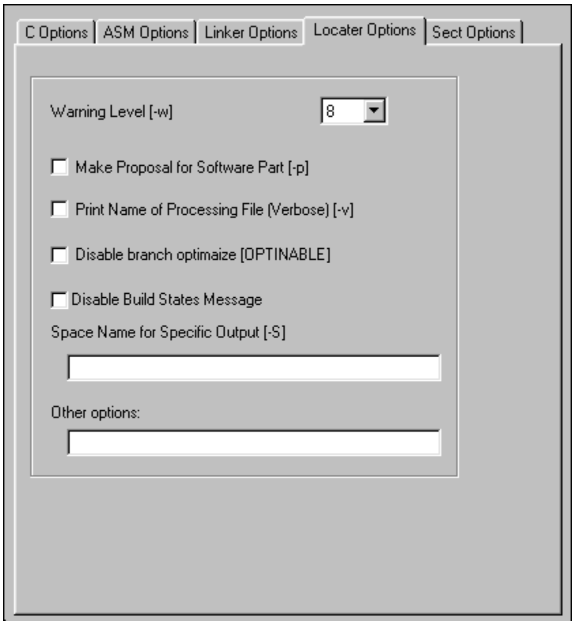
To specify other options (including those listed above), enter the desired option in this text box in command line format.

About options that are not displayed

The linker options not displayed in the Option view are handled as described below:

-e	This option is used in internal processing.
-err	Linker messages are displayed in the message window and output to an error log file.
-f *file*	This option conflicts with internal processing and cannot be used.
-l *x*	This option is automatically processed internally in accordance with memory model settings and system library search settings.
-O *file*	File names are set to the project name.
-o *file*	File names are set to the project name.
-u *symbol*	To specify this option, enter it in [Other Options].
-V	This option is not used in wb88.

3.9.4 Locator Options



Warning Level [-w] 8

☐ Make Proposal for Software Part [-p]

☐ Print Name of Processing File (Verbose) [-v]

☐ Disable branch optimaize [OPTINABLE]

☐ Disable Build States Message

Space Name for Specific Output [-S]

Other options:

This screen can be used to select the following options:

Warning Level "-w *n*" option of lc88

Specify the level to which to suppress warning messages. Levels 0 to 9 can be selected from the pull-down list. The default setting is 8. Warning messages whose levels are higher than the selected value are not displayed.

Make Proposal for Software Part "-p" option of lc88

If this option is selected, the locator displays proposals for the software part of a locator description file.

Print Name of Processing File (Verbose) "-v" option of lc88

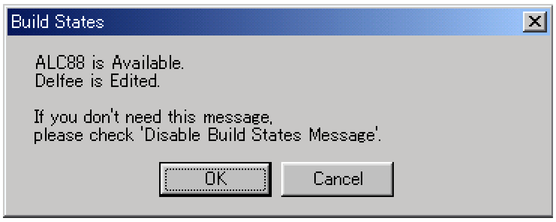
If this option is selected, the locator displays the name of the file currently being processed.

Disable branch optimize

Select this option when using lc88. When the check box is deselected (default), alc88 is used to generate object files in executable format.

Disable Build States Message

When this check box is deselected (default), the dialog box shown below appears when the software starts building or rebuilding a project.



ALC88 is Available.
Delfee is Edited.

If you don't need this message,
please check 'Disable Build States Message'.

This dialog box indicates which locator (alc88 or lc88) is to be used (based on whether the [Disable branch optimize] check box for locator options is selected or deselected), and whether locator description files in DELFEE are to be edited by the section editor (based on whether the [Disable Making DELFEE] check box of the section editor is selected or deselected).
If the wrong locator or edit mode is selected, use the [Cancel] button in this dialog box to stop building (or rebuilding) a project.

If this dialog box need not be displayed, click the [Disable Build States Message] check box.

Space Name for Specific Output "-S *space*" option of lc88

Enter a space name here; the locator then generates a specific output file corresponding to the specified space.

Other options

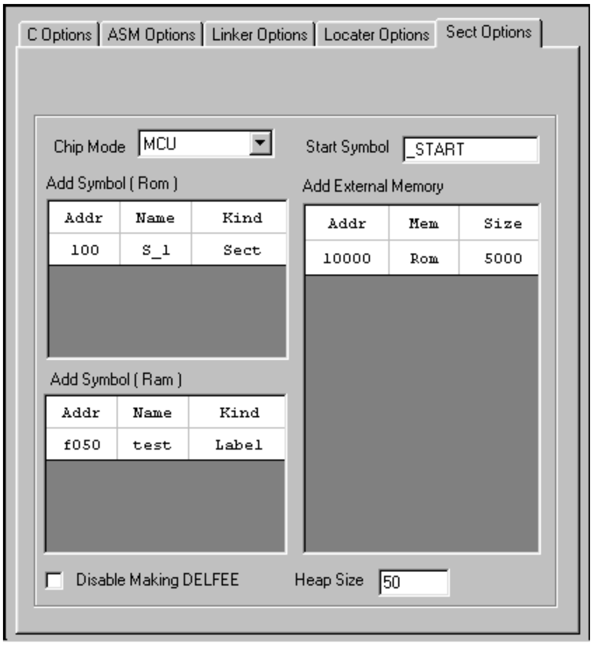
To specify other options (including those listed above), enter the desired option in this text box in command line format.

About options that are not displayed

The locator options not displayed in the Option view are handled as described below:

-d *file*	The dsc file is always specified.
-e	This option is used in internal processing.
-err	Locator messages are displayed in the Message window and output to an error log file.
-f *file*	This option conflicts with internal processing and cannot be used.
-f *format*	This option always generates IEEE 695 standard (.abs) and Motorola S (.s) files.
-M	This option always generates a locate map file.
-o *file*	File names are set to the project name.
-V	This option is not used in wb88.

3.9.5 Section Editor



Chip Mode MCU

Start Symbol _START

Add Symbol (Rom)

Addr	Name	Kind
100	S_1	Sect

Add External Memory

Addr	Mem	Size
10000	Rom	5000

Add Symbol (Ram)

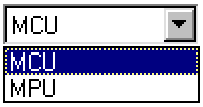
Addr	Name	Kind
f050	test	Label

□ Disable Making DELFEE

Heap Size 50

Use this screen to specify where sections, symbols, and external memory should be located.
The absolute address information specified here is referenced by wb88 as it generates the advanced locator definition file or DELFEE-based locator description files, which are used as input files for advanced locator alc88 or locator lc88 when executing build.

Chip Mode



From the pulldown list, choose which mode to use for the chip, MCU, or MPU mode. Choose MCU mode when using internal ROM. Choose MPU mode when releasing the internal ROM area for external memory (i.e., not using internal ROM).

Start Symbol

Set a start symbol. The contents set here are referenced as the "load_mod start=" parameter in locator description file (.dsc).
The default is _START, which can suffice when starting from cstart.c. When starting from another C routine, set a function name prefixed by "_"; when starting from an assembler routine, set the symbol name of that routine.
Example:

1. Assembler routine

   `GLOBAL _main`

   When starting from `_main:`, set `_main`

2. C routine

   When starting from `void main( )`, set `_main`

Add Symbol (Rom)

Set the name and address of a section, vector table, or label to be located in ROM.
The items to be set in the respective lines are described below.

Addr Enter the start address of a section or vector table, or the address to which to assign a label. When sections are to be located at contiguous addresses, the start address of only the first section is required and the start addresses of the second and subsequent sections may be left blank. When different types of sections generated by the compiler are to be located at contiguous addresses, however, the start address must be specified for each section (as detailed later).

Name Enter the name of a section, vector table, or label (symbol name).

Kind Choose the type of item to be located from the pulldown list:

Vect	Vector table
Label	Label
Sect	Section

Add Symbol (Ram)

Set the name and address of a section or label to be located in RAM.
The items to be set in the respective lines are described below.

Addr Enter the start address of a section or the address to which to assign a label.
When sections are to be located at contiguous addresses, the start address of only the first section is required and the start addresses of the second and subsequent sections may be left blank. When different types of sections generated by the compiler are to be located at contiguous addresses, however, the start address must be specified for each section (as detailed later).

Name Enter the name of a section or label (symbol name).

Kind Choose the type of item to be located from the pulldown list:

Label	Label
Sect	Section

Add External Memory

Set the address and size of memory or a device to be connected to the external bus of the microcomputer.
The items to be set in the respective lines are described below.

Addr Enter the start address of external memory or a device.

Mem Choose the type of external memory from the pulldown list:

Rom	ROM
Ram	RAM
Dev	Any memory-mapped device (e.g., LCD controller)

Size Enter the capacity of external memory or the mapped size of a device in bytes.

Disable Making DELFEE

Choose whether you want locator description files in DELFEE language to be generated by the section editor.

When the check box is deselected (default)

The section editor references the contents set on this screen as it generates the advanced locator definition file for alc88 or locator description files for lc88.

When the check box is selected

The section editor does not generate locator description files for lc88. To use the existing locator description files you created, deselect this check box. In this case, an advanced locator definition file for alc88 is also generated according to the contents set on this screen.

Heap Size

Specify the size of a heap area for which memory is to be allocated by malloc( ), etc. Note that this setting only becomes effective when heap area is required and malloc( ), etc. actually used.

When using Advanced Locator alc88

When using alc88, make the following settings for locator options and in the section editor.

1. Deselect the [Disable branch optimize] check box on the [Locator Options] tab screen.
   The following settings must be made using the section editor:
2. Deselect the [Disable Making DELFEE] check box.
3. Choose the mode to be used for the chip (MCU or MPU mode) from the [Chip Mode] list.
4. Enter the location addresses of sections, etc. in the [Add Symbol (Rom)] and [Add Symbol (Ram)] boxes. (How to enter will be detailed later.)
5. To use external memory or a device, enter the information on it in the [Add External Memory] box. (How to enter will be detailed later.)

Because [Disable branch optimize] for locator options has been deselected, alc88 is invoked when building a project.

When using Locator lc88: Case 1
(Locator description files generated by the section editor are used.)

When you need not use existing locator description files, we recommend using alc88. When necessary to use lc88, make the following settings:

1. Deselect the [Disable Making DELFEE] check box.
   If this check box cannot be deselected, go to the [Locator Options] tab screen and deselect the [Disable branch optimize] check box on it before making this setting.
2. Choose the mode to be used for the chip (MCU or MPU mode) from the [Chip Mode] list.
3. Enter a start symbol name in [Start Symbol] as necessary. (Normally, leave _START intact.)
4. Enter the location addresses of sections, etc. in the [Add Symbol (Rom)] and [Add Symbol (Ram)] boxes. (How to enter will be detailed later.)
5. To use external memory or a device, enter the related information in the [Add External Memory] box. (How to enter will be detailed later.)
6. Select the [Disable branch optimize] check box on the [Locator Options] tab screen.

Because [Disable branch optimize] for locator options has been selected, lc88 is invoked when building a project.

When using Locator lc88: Case 2
(Existing locator description files are used.)

To use existing locator description files as may be needed when upgrading application versions, make the following settings:

1. Select the [Disable Making DELFEE] check box.
   As a result of this setting, the [Disable branch optimize] check box for locator options is automatically selected.
2. The files stored in the [Definition Files] folder will be listed in project view, so correct any locator description file as necessary.

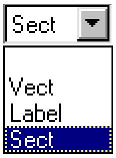
Because [Disable branch optimize] for locator options has been selected, lc88 is invoked when building a project.

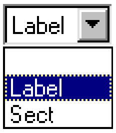
Note: When using existing locator description files you need not enter location addresses, etc. in the section editor. Note, however, that even in this case, an advanced locator definition file even with incomplete content is generated (i.e., contents of locator description files are not reflected). If you want to change for processing by alc88, therefore, be sure to correctly recreate an advanced locator definition file.

[Add Symbol (Rom/Ram)] – Defining and deleting symbols

To define symbols in [Add Symbol (Rom / Ram)], follow the procedure described below.

1. Click the [Addr] cell on a blank line, and enter an address in it.
2. Enter a symbol in [Name].
3. Click the [Kind] cell to display a pulldown list similar to the one shown below. Select the type of item to be located from this list.

(Rom) 

(Ram) 

4. When three cells are filled in, click the [Enter] key and a blank line will be added below.
5. Repeat the above procedure until you enter all sections, etc. to be located. When sections of the same kind are to be located at contiguous addresses, you need only specify [Addr] for the first section and can omit those for the second and subsequent sections. [Name] and [Kind] cannot be omitted. If the kind of section is different from the immediately preceding section that you have set, you must enter [Addr] for that section. Otherwise, the line that you are setting has no effect and you cannot go to the next line. The sections generated by the compiler require special caution with respect to the difference in kind.

The addresses need not be entered in descending or ascending order.
The definition files are updated for what you have entered or selected when you start building (or rebuilding) a project, saving a project, or quitting wb88.

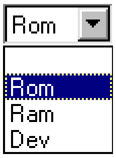
To delete the addresses set in [Add Symbol (Rom / Ram)]:

1. Delete all contents of [Addr], [Name] and [Kind] on the address line you want to delete (by using the [Backspace] or [Delete] key and selecting blank for [Kind]).
2. When three cells have been blanked, click the [Enter] key. The line will be deleted, with subsequent lines moved up.

[Add External Memory] – Defining and deleting external memory

For systems that have ROM or RAM, or such external devices as an LCD controller connected to the external bus, you need to assign addresses and set the size of memory or the device in [Add External Memory].

1. Click the [Addr] cell on a blank line, and enter an address in it.
2. Click the [Mem] cell to display a pulldown list similar to the one shown below. Choose the type of external memory from this list.



3. Enter the size of external memory in [Size].
4. When three cells are filled in, click the [Enter] key and a blank line will be added below.
5. Repeat the above procedure until defining all the external memory and devices required.

The addresses need not be entered in descending or ascending order.
The definition files are updated for what you have entered or selected when you start building (or rebuilding) a project, saving a project, or quitting wb88.

To delete the external memory definitions set in [Add External Memory]:

1. Delete all contents of [Addr], [Mem] and [Size] on the line you want to delete (by using the [Backspace] or [Delete] key and selecting blank for [Mem]).
2. When three cells have been blanked, click the [Enter] key. The line will be deleted, with subsequent lines moved up.

Precautions

1. Limitations on input content

The maximum number of lines and maximum number of characters that can be entered are limited as follows:

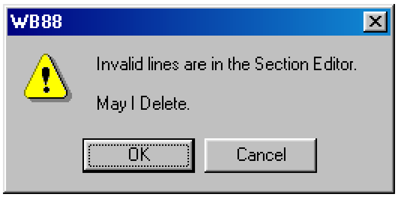
Maximum number of lines entered	[Add Symbol (Rom / Ram)]	100 lines
	[Add External Memory]	20 lines
Maximum number of characters entered	[Addr]	8 digits
	[Name]	30 characters
	[Size]	8 digits

2. Checking the entered data

When you start building (or rebuilding) a project, saving a project, or quitting wb88, the Work Bench checks whether all necessary items of the section editor are filled in.

When no problems are found, wb88 continues or terminates processing.

If a deficiency is found, such as when only two of the three necessary items for symbol or external memory definitions are filled in, the dialog box shown below appears.



Invalid lines are in the Section Editor.

May I Delete.

Click [OK], and wb88 will delete invalid lines before it continues or terminates.

Click [Cancel], and wb88 will stop building (or rebuilding) a project, stop saving a project, or quitting.

Note that wb88 does not check input content for whether the addresses you have entered are within the implemented memory area or whether there are any duplicate symbol names. Such discrepancies or errors are checked by alc88 or lc88.

3. About sections generated by the compiler

When user-defined successive sections are to be specified in [Add Symbol (Rom / Ram)], the address of only the first section need be specified and the addresses of those that follow can be omitted. In addition to these, sections generated by the compiler can also be specified here. In this case, however, care must be taken because the compiler generates different types of sections. Even when sections are to be located at contiguous addresses, the address of a different type of section that follows another section must be specified.

Several types of sections generated by the compiler are listed below.

ROM area

```
code_short
          .comm
          .startup
code
          .text
          .text_xxxxxxxx
          table.......... Address cannot be specified.
data_short
          .nrdata
data
          .frdata
```

RAM area

```
data_short
          .nbss
          .ndata
          .nbssnc
data
          .fdata
          .fbss
          .fbssnc
          stack.......... Address cannot be specified.
          xvwbuffer ..... Address cannot be specified.
```

4. About vectors and labels

Vectors and labels can be defined in [Add Symbol (Rom / Ram)] as matched to the functions of lc88. The user can access the external (extern) vectors or labels named __lc_u_xxxxx, and the addresses of those vectors or labels can be defined in the section editor.
When defining vectors or labels in [Add Symbol (Rom / Ram)], you need only enter the name part "xxxxx".

5. Operations for deselecting the [Disable Making DELFEE] check box while currently selected

Selecting this check box automatically selects the [Disable branch optimize] check box for locator options. While in this state, the [Disable Making DELFEE] check box cannot be deselected again. To deselect this check box while it currently is selected, first deselect the [Disable branch optimize] check box for locator options.

6. About special sections

The following four types of sections cannot be specified in the section editor. If any of these sections are specified, it will be deleted when you save or build a project.
"heap", "stack", "table" and "xvwbuffer"

3.10 Debugging

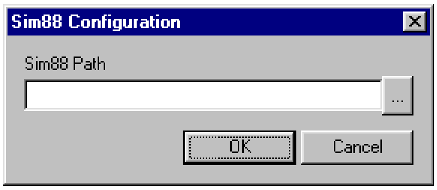
Programs can be debugged by invoking the simulator or in-circuit emulator from the Work Bench.

3.10.1 Simulator

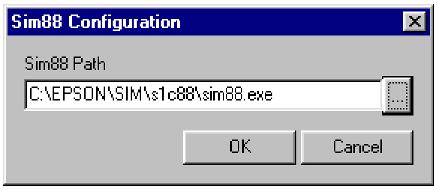
This section describes how to invoke the simulator sim88 from the Work Bench. For detailed information on simulator functions and usage, please refer to the simulator manual.

Setting the path to the simulator

Before simulator sim88 can be invoked, you must set its path. To set the path, select [Sim88 Configuration] from the [Tool] menu to display the dialog box shown below.



Select sim88.exe from the dialog box that is displayed by clicking the [...] button, or enter a path directly into the text box.



Sim88 Path: C:\EPSON\SIM\s1c88\sim88.exe

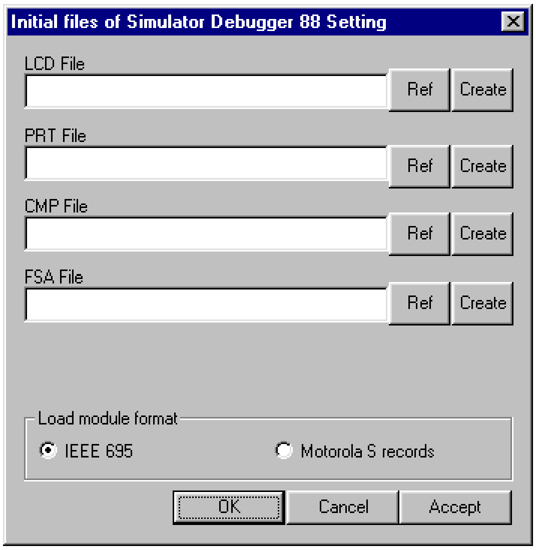
Once a path is set, there is no need to set it again the next time the simulator is run.

Invoking the simulator

To invoke the simulator

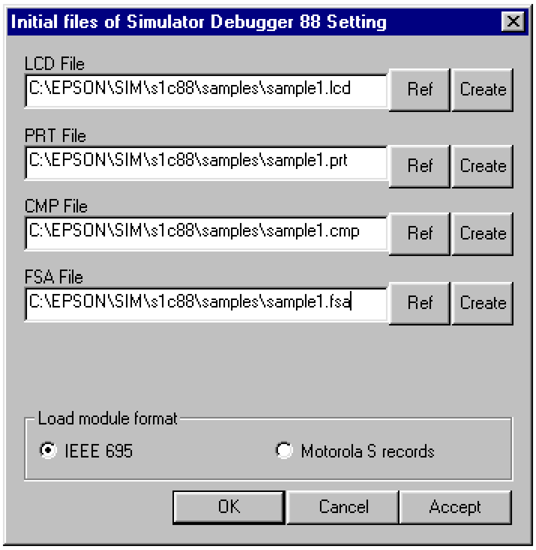
1. Select [Sim 88 Simulator] from the [Debug] menu or click the [Sim88] button. The dialog box shown below is displayed:

   [Sim88] button



Load module format: ● IEEE 695 ○ Motorola S records

2. Specify the following files needed to invoke the simulator. Select each file from the file select dialog box displayed by clicking the [Ref] button, or enter a path for each file directly into the text box.
   LCD File: LCD panel definition file
   PRT File: Port setting file
   CMP File: Component mapping file
   FSA File: Function option HEX file



Initial files of Simulator Debugger 88 Setting

LCD File: C:\EPSON\SIM\s1c88\samples\sample1.lcd

PRT File: C:\EPSON\SIM\s1c88\samples\sample1.prt

CMP File: C:\EPSON\SIM\s1c88\samples\sample1.cmp

FSA File: C:\EPSON\SIM\s1c88\samples\sample1.fsa

Load module format: ● IEEE 695 ○ Motorola S records

   For detailed information on the LCD panel definition file, port setting file, and component mapping file, refer to the simulator manual.

   Click the [Create] button to launch the tool to create each file.
   LCD File: LCD panel customize utility LcdUtil
   PRT File: Port setting utility PrtUtil
   CMP File: Editor (specified with [Tool | Editor Configuration])
   FSA File: Function option generator winfog (see Chapter 8)

   For detailed information on the LCD panel customize utility and port setting utility, refer to the simulator manual. These tools can also be launched from the [Tool] menu or with the toolbar button.

3. Using the [Load module format] radio button, select the object file format (IEEE 695 or Motorola S) to be loaded into the simulator.

4. Click the [OK] button to close the dialog box and start the simulator. From the input file information, the Work Bench generates a simulator project file (.spj) and a command file to load the necessary files, then passes these files to the simulator. The simulator is ready to start debugging as soon as it is started.

   The [Accept] button only generates the above files. It does not close the dialog box or launch the simulator.

3.10.2 In-circuit Emulator (S5U1C88000H5) and Debugger

This section describes how to invoke the debugging system using the ICE (S5U1C88000H5) from the Work Bench. Refer to Chapter 13 for the db88 debugger and the S5U1C88000H5 manual for detailed information on ICE and ice88ur debugger usage and functions.
To invoke the S5U1C88000H5 system

1. Check to see that the ICE is connected to the personal computer on which it is running and that its power is turned on.

2. Start the Work Bench.

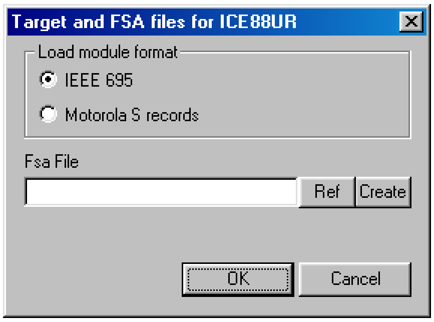
3. To start the db88 debugger, select [DB88 Debugger] from the [Debug] menu or click the [DB88] button.

   [DB88] button

   To start the ice88ur debugger, select [ICE88UR Debugger] from the [Debug] menu or click the [ICE88UR] button.

   [ICE88UR] button

   The dialog box shown below is displayed:

   Target and FSA files for ICE88UR

   Load module format

   ● IEEE 695

   ○ Motorola S records

   Fsa File

4. Using the [Load module format] radio button, select the absolute object file format (IEEE 695 or Motorola S).

5. In [Fsa File], specify a function option HEX file. This is done by selecting a file from the file select dialog box displayed by clicking the [Ref] button, or by entering a path for the file directly into the text box. The [Create] button invokes the function option generator winfog that generates a function option HEX file.

6. Click the [OK] button to close the dialog box and launch the debugger. The Work Bench generates a command file to load the necessary files from the input information and passes it to the debugger. The debugger is ready to start debugging as soon as it is started.

3.11 Executing Other Tools

The following tools can be launched from the [Tool] menu or with the toolbar buttons.

Table 3.11.1 Tools that can be launched from wb88

Tool	Menu item	Button
1. Auto evaluation system	[Tool | Simulator Tools | Auto Evaluation System]	
2. Bitmap utility	[Tool | Simulator Tools | Bitmap Utility]	
3. LCD panel customize utility	[Tool | Simulator Tools | LCD Panel Customize Utility]	LCD
4. Port Setting Utility	[Tool | Simulator Tools | Port Setting Utility]	
5. Function option generator	[Tool | Dev Tools | Function Option Generator]	
6. Mask data checker	[Tool | Dev Tools | Mask Data Checker]	
7. On-board ROM writer control software	[Tool | On-Board ROM Writer]	

For information on how to use each tool, refer to the simulator manual for tools 1 to 4, the corresponding chapters in this manual for tools 5 to 6, and the flash EEPROM-containing microcomputer technical manual for tool 7.

3.12 File List

The table below lists the types of files handled by the Work Bench, and the locations where the files are located.

Table 3.12.1 File list

File type	File name	Extension	Creator/tool	Folder path (default)
C compiler-related files				
C source file	Option	.c	User/text editor	Option (<project>\src)
C header file	Option	.h	User/text editor	Option (<project>\src)
C startup routine	cstartup	.c	wb88	<project>\def\
Assembly source (created by user)	Option	.asm	User/text editor	Option (<project>\src)
Assembly header file	Option	.inc	User/text editor	Option (<project>\src)
Bitmap file	Option	.bmp	User/bitmap editor	Option
Bitmap definition file	Option	.bmu	User/BmpUtil	Option
Data table	Option	.txt	User/BmpUtil	Option
Project management file	Project name	.wpj	wb88	<project>\
Make file	makefile	–	wb88	<project>\tmp\
Error log file	Project name	.err	wb88/cc88	<project>\tmp\
Intermediate assembly source file	[Source name reference]	.src	wb88/c88	<project>\obj\
Assembly list file	[Source name reference]	.lst	wb88/as88	<project>\obj\
Object file	[Source name reference]	.obj	wb88/as88	<project>\obj\
Object library file	Option	.a	User/ar88	Option
Linker object file	Project name	.out	wb88/lk88	<project>\obj\
Link map file	Project name	.lnl	wb88/lk88	<project>\obj\
Call graph file	Project name	.cal	wb88/lk88	<project>\obj\
Advanced locator definition file	Model name	.inf	wb88	EPSON\S1C88\Dev\
Locator definition file	Model name	.dsc	User/text editor	<project>\def\
CPU definition file	Model name	.cpu	User/text editor	<project>\def\
Memory definition file	Model name	.mem	User/text editor	<project>\def\
Locate map file	Project name	.map	wb88/lc88	<project>\obj\
Absolute load module	Project name	.abs	wb88/lc88	<project>\obj\
Motorola S module	Project name	.sa	wb88/lc88	<project>\obj\
Symbolic table file	Project name	.sy	wb88/sy88	<project>\obj\
Program data HEX file	Project name	.psa	wb88/fil88xxx	<project>\obj\
Development tool-related files				
Device information definition file	Model name	.ini	Seiko Epson	EPSON\S1C88\Dev\
Function option HEX file	Option	.fsa	User/WinFOG	Option
Function option document file	Option	.fdc	User/WinFOG	Option
Mask data file	Option	.paN	User/WinMDC	Option
Automatic evaluation system-related files				
Command file	Option	.txt	User	Option
Reference data file	Option	.mXX	User	Option
Result data file	Option	.aXX	User	Option
Check sheet file	Option	.csv	User/AutoEva	Option
Simulator-related files				
LCD panel definition file	Option	.ldc	User/LCDUtil	Option
Port setting file	Option	.prt	User/PrtUtil	Option
Simulator project file	sim88	.spj	wb88	<project>\tmp\
Command file	debug	.cmd	wb88	<project>\tmp\
Component map file	Option	.cmp	User/text editor	Option
ICE-related files				
ICE parameter file	Model name	.par	User/text editor	<project>\def\
FPGA data file for peripheral circuit boards	Model name	.mot	Seiko Epson	
INI file for ICE	ice88ur	.ini	wb88	<project>\tmp\

3.13 Error Messages

The following tables list error messages associated with the Work Bench.

Table 3.13.1 System error messages

Message	Description
not enough memory	There is insufficient memory to run wb88.

Table 3.13.2 Error messages output when generating a project

Message	Description
The file is not a WB88 project file.(<filename>)	The file <filename> is not a wb88 project file.
The version of the project file is not supported.(<filename>)	This version of the project file <filename> is not supported.
Unable to create a project : cannot access. <filename>	Unable to generate a project because the file <filename> could not be accessed correctly.
Unable to create a project : Unable to copy DEF file.(<filename>)	Unable to generate a project because wb88 failed to copy the definition file <filename>.
The project is already existed.(<filename>)	Unable to create a project because the file <filename> already exists. Two or more projects with the same name cannot be created in the same folder.
Unable to create a project : Dev Directory of S1C88 family package does not exist.	Unable to create a project because no DEV directories exist. The DEV directory of the package contains various definition files required for build task. No projects can be built without this directory.

Table 3.13.3 Error messages output when adding files to the project

Message	Description
The file cannot be added to the project. It is not a C file.(<filename>)	The file <filename> cannot be added to the project because it is not a C source file.
The file cannot be added to the project. It is not an ASM file.(<filename>)	The file <filename> cannot be added to the project because it is not an assembly source file.
The file cannot be added to the project. It is not a header file.(<filename>)	The file <filename> cannot be added to the project because it is not a header file.
The file is already existed in the project. It cannot be added in the project.(<filename>)	The file <filename> cannot be added to the project because it already exists.
WB88 does not support such source file type.(<filename>)	This source type file is not supported by wb88.

Table 3.13.4 File error messages

Message	Description
Failed to access the file.(<filename>)	Failed to operate on the file <filename>.
Unable to open the file.(<filename>)	Failed to open the file <filename>.

Table 3.13.5 Error messages output when starting a tool

Message	Description
Unable to execute ICE88UR.exe : Unable to access <filename>.	Cannot start S5U1C88000H5 because wb88 could not access the file <filename>.
Unable to execute Sim88 : Unable to access the DEF file.(<filename>)	Cannot start Sim88 because wb88 could not access the definition file.
Unable to execute <toolname>.	Unable to start <toolname>.

Table 3.13.6 Error messages output when building

Message	Description
Select a C or an ASM file.	Select a C source or assembly source file. Before source files can be compiled, you must select the target file from tree view.
Build Command needs an active project.	The build target must be project.
No target file is found in the project.	No target files to build are found in the project. Source files must be registered to a project before they can be built.

Table 3.13.7 Other error messages

Message	Description
The command needs an active project.	The command requires a project. This error message is displayed if, in the absence of a project, a function is executed for which a project must be present.

CHAPTER 4 OUTLINE OF THE MAIN TOOL CHAIN

The Main tool chain consists of the following tools centered on the C compiler:

1. C compiler <c88.exe>

Compiles C source files to generate assembly source files that can be processed by as88. Note that c88 is an ANSI C-compliant C compiler. Because no special syntax is supported, programs developed for other types of microcomputers can be easily ported to run on the S1C88. Moreover, because the S1C88 architecture can be efficiently used at the C level to generate compact code, c88 is best suitable for the development of embedded applications. With the preprocessor, S1C88 C front-end, and code generator integrated into a single program, c88 operates at high speed as a one-pass compiler without requiring intermediate files.

2. Assembler <as88.exe>

Assembles the assembly source files output by c88 to convert the mnemonics in those files into S1C88 object (machine language) code. The result of this operation is output as relocatable object files in IEEE-695 format that can be linked by lk88.

3. Linker <lk88.exe>

Combines two or more relocatable object files generated by as88 with a library module to generate one new relocatable object file.

4. Locator <lc88.exe>

Relocates the relocatable object created by lk88 to absolute addresses of memory to generate an executable load image file. The relocation information to be referenced at this time must be written in DELFEE language in the locator description files that are loaded on the locator.

Note that lc88 can be used to develop applications using existing locator description files. When you develop new applications, we recommend the use of newly added advanced locator alc88 (beginning with S5U1C88000C Ver. 3) because it has a new branching optimization function in addition to all the functions of lc88. You can select whether to use lc88 or alc88 in wb88.

5. Advanced locator <alc88.exe>

Realizes the relocation functions of lc88 without using description files in DELFEE. For memory models with 64K bytes or more of code area, alc88 should prove especially useful because although extended instructions for bank specification (e.g., LD NB,xxxx) are added immediately before the call instruction (CARL) by the assembler, alc88 has a function to delete unnecessary extended instructions that have been added for intra-bank calls.

Refer to the document titled "S5U1C88000C Manual I" for details about tools 1 to 4. Advanced locator alc88 in 5 is detailed in this manual. Note that because all of the above tools are executed by the functions of wb88, you need not operate any tool individually.

CHAPTER 5 ADVANCED LOCATOR <alc88>

5.1 Functions of alc88

Advanced Locator <alc88> relocates the relocatable object created by linker <lk88> to the absolute addresses of memory to generate an executable load image file. In addition, alc88 has a branching optimization function. This function is effective for memory models with 64K bytes or more of code area (Compact-Data or Large), in which case extended instructions for bank specification (e.g., LD NB,xxxx) are unconditionally added immediately before the call instruction (CARL) by the assembler. However, alc88 deletes such extended instructions whenever found in intra-bank calls.

This function enables alc88 to generate more compact executable object files than those generated by locator <lc88> that has been conventionally used in the Main tool chain.

Moreover, the locator description files in DELFEE language used to provide lc88 with relocation information are not required for alc88. Instead, alc88 uses the advanced locator definition file (.inf) that can be easily generated by the section editor functions of wb88 without any specific concern about details.

Therefore, you have the option of using lc88 when using conventional resources (including locator definition files) to develop applications or alc88 when developing new applications, or not specifically requiring existing locator definition files. You can select which tool to use in wb88.

Note: Branching optimization is only useful for the CARL instruction (in the format below) that causes the CPU to branch off to locations within the same bank (32K-byte area). The extended instructions added before other branch instructions (e.g., jump instruction) are not deleted even if unnecessary. Also note that for extended or branch instructions where the address for an object is already fixed before being entered, the extended instructions are not deleted even if the target of optimization.

```
LD   NB,xxxx
CARL yyyy
```

When yyyy exists in the same bank as the CARL instruction, the immediately preceding "LD NB,xxxx" is deleted.

When yyyy exists in a bank different than that of the CARL instruction, the immediately preceding "LD NB,xxxx" is not deleted.

5.2 Input/output Files

Figure 5.2.1 shows the input/output files of alc88.

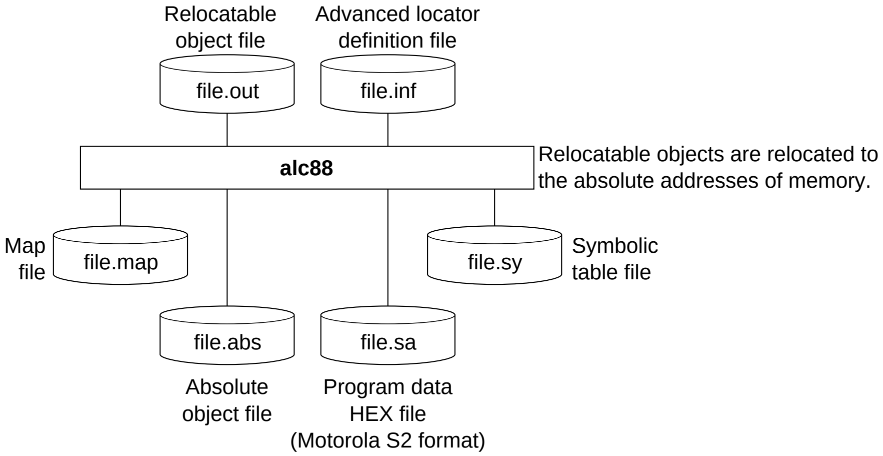

Relocatable object file
Advanced locator definition file
file.out
file.inf
alc88
Relocatable objects are relocated to the absolute addresses of memory.
Map file
file.map
file.sy
Symbolic table file
file.abs
file.sa
Absolute object file
Program data HEX file (Motorola S2 format)

Fig. 5.2.1 Input/output files of alc88

Relocatable object file (file.out)

This is the relocatable object file in IEEE-695 format that has been output by the linker <lk88>.

Advanced locator definition file (file.inf)

This file contains a description of information referenced by alc88 as it relocates relocatable objects to absolute addresses of memory. The section editor of wb88 creates this file.

Absolute object file (file.abs)

This is an executable object file output from the relocatable objects supplied to alc88 by being relocated to the absolute addresses of memory. This file is created in IEEE-695 format and contains debugging information included in the input files.

Program data HEX file (file.sa)

This HEX file is output from absolute objects converted into Motorola S2 format. This file is presented as an input file the program unused area filling utility <fil88xxx>.

Map file (file.map)

A list of absolute addresses to which sections and labels have been allocated is recorded in this file.

Symbolic table file (file.sy)

This file contains symbol information extracted from the debugging information in the input files. This file is required for the symbolic debugging to be performed by the debugger or simulator.

5.3 Using alc88

All operations including the creation of advanced locator definition files are normally handled by wb88. Because alc88 is automatically invoked by wb88 as it executes build processing, the user need not start alc88. The advanced locator definition file is created by using the section editor of wb88. See Chapter 3, "Work Bench", for details on how to build a project or use the section editor.

To run alc88 independently of wb88, execute the following command from the MS-DOS prompt:

```
>alc88  <project_path>  <file.out>  <file.inf> ↵
```

↵	Denotes entering the return key.
<project_path>	Specify the path to the project file (.wpj).
<file.out>	Specify the object file name to be supplied to alc88.
<file.inf>	Specify the advanced locator definition file to be supplied to alc88.

Example: `C:\epson\s1c88\app1 app1.out app1.inf`

When alc88 completes processing, it displays the following message (to stdout) regardless of whether it terminated normally.

```
ALC88 Version x.xx
```

5.4 Error Messages

The error messages of alc88 are listed below.

Table 5.4.1 Error messages

Error message	Description
Illegal Inf File	Advanced locator definition file (.inf) is invalid.
Duplicate Memory -- 0xnnnn ~ 0xnnnn & 0xnnnn ~ 0xnnnn	Memory allocations in 0xnnnn–0xnnnn and 0xnnnn–0xnnnn are duplicated.
No physical memory available for xxxx	No specified addresses exist to which symbol xxxx can be assigned.
Duplicate Symbol Name -- xxxx	There are duplicates of symbol name xxxx.
Cannot find 0xnnnn bytes for xxxx section	No 0xnnnn bytes of memory are available as needed to map section xxxx.
Found unresolved external -- xxxx	No information is available for external symbol (Extern) xxxx.
There is no stack area	No memory can be allocated for the stack because internal RAM lacks sufficient space.
Absolute address 0xnnnn occupied	The absolute address section area beginning with 0xnnnn is already occupied by another area.
Value out of range to label xx at address 0xnn	The branch destination of the short branch instruction (JRS, CARS) is out of the range (-128 to 127).

5.5 Precautions

Note that alc88 is subject to the limitations described below.

(1) Of the effective label descriptions of lc88, alc88 only supports user-defined labels (__lc_cp, __lc_es, __lc_u_xxxx, __lc_b_xxxx, __lc_e_xxxx). The labels __lc_bs, __lc_ub_xxx, __lc_ue_xxx, etc. used in the source have no effect on alc88. Refer to Section 4.9, "Locator Labels", in the "S5U1C88000C Manual I".

(2) Even when branching is optimized by alc88, the results of such optimization are not reflected in the list files created by as88, regardless of whether relocatable or absolute.

CHAPTER 6 OUTLINE OF THE DEVELOPMENT TOOLS

The S1C88 Family Integrated Tool Package contains the tools to create mask option and mask data files, as well as files that contain descriptions of setup information for each type of microcomputer. The tools 1 to 3 below are Windows GUI applications that run under Windows 2000 or Windows XP.

1. Function option generator <winfog.exe>

This tool creates an ICE (S5U1C88000H5) function option setup file after selecting the mask options of the S1C88xxx and the function option document file that is necessary to generate IC mask patterns. You can create function option data by selecting the appropriate item using the check boxes.

2. Segment option generator <winsog.exe>

This tool creates an ICE segment option setup file after selecting the segment options of the S1C88xxx and the segment option document file that is necessary to generate IC mask patterns. You can create segment assignment data by merely clicking on the display memory map and segment decode table shown on the window.

3. Mask data checker <winmdc.exe>

This tool checks the data in development-completed built-in ROM file and option document files to create the mask data file that will be presented to Seiko Epson.

4. Device information definition file <s1c88xxx.ini>

This file is used to set information, such as the configuration of options, on each type of microcomputer for the three tools described above. This file must be available before each tool can be executed.

5. ICE parameter file <88xxx.par>

This file is used to establish correspondence between the ICE and each type of microcomputer. This file is required for starting up the ICE.

6. Program unused area filling utility <fil88xxx.exe>

This tool extracts the built-in ROM area from a program data HEX file and fills unused areas in the built-in ROM with FFH. It also sets a system code to the system-reserved area. This processing must be performed before debugging the program with the ICE as well as before generating a mask data with winmdc. This tool can be executed from the MS-DOS prompt.

7. Self-diagnostic program <t88xxx.psa, t88xxx.fsa, t88xxx.fdc, t88xxx.ssa, t88xxx.sdc, readme.txt>

These are the self-diagnostic program and function option data to check the ICE and S5U1C88xxxP hardware. Download these files to check the ICE. The t88xxx.ssa and t88xxx.sdc files are included only for microcomputers in which segment options are provided.

The readme.txt file contains the description of the S5U1C88xxxP LED illumination status to check the operation with the self-diagnostic program.

Notes:
- *There is no difference between each tool between the different types of microcomputers. Therefore, the explanations in this manual are for all types of microcomputers using "S1C88xxx" as the representative name. The contents of the sample screens shown in this manual vary according to the type of microcomputer. Note that winsog, t88xxx.ssa and t88xxx.sdc are provided only for microcomputers with segment options.*
- *S5U1C88000H3 (previous name: ICE88R) is provided in addition to S5U1C88000H5.*

Differences between new tools (S5U1C88000P-compliant version) and existing tools

The old peripheral boards (S5U1C88316P and S5U1C88348P) have been replaced by a new standard peripheral board (S5U1C88000P). Note that tool action and functionality may differ somewhat given the combination of new and old peripheral boards, and development tools.

For the following types of MPUs, the Integrated Tool Package for the S1C88 Family includes new development tools, which are useful with the standard peripheral board (S5U1C88000P).

S1C88104, S1C88112, S1C88308, S1C88316, S1C88317, S1C88348, S1C88832, S1C88862

Table 6.1 Functional differences depending on combinations of S5U1C88316P and S5U1C88348P peripheral boards and development tools

Functions Combination	S1C88832/862's $\overline{\text{BZ}}$ (R51) and $\overline{\text{TOUT}}$ (R26) outputs	Variation of OSC1/3 oscillator frequencies (OSC1 is for a CR oscillator; OSC3 is for a CR or ceramic oscillator)
Old peripheral board + old development tools	Not available	Not available (Because OSC1 and OSC3 are respectively fixed to 32.768 kHz and 4.9152 MHz, clocks from external sources may be used for other oscillator frequencies as required.)
New peripheral board + new development tools	Available	Available
Old peripheral board + new development tools	Not available Not considered a problem	Not available (Because OSC1 is fixed to 32.768 kHz (with crystal selected) or 32 kHz (with CR selected), and OSC3 is fixed to 8 MHz (with ceramic selected) or approx. 8 MHz (with CR selected), clocks from external sources may be used for other oscillator frequencies as required.) Not considered a problem
New peripheral board + old development tools	Not available Not considered a problem	Not available (Because OSC1 and OSC3 are respectively fixed to 32.768 kHz and 4.9152 MHz, clocks from external sources may be used for other oscillator frequencies as required.) Not considered a problem

CHAPTER 7 PROGRAM UNUSED AREA FILLING UTILITY <fil88xxx>

7.1 Outline of fil88xxx

The Program Unused Area Filling Utility <fil88xxx> loads a Motorola S2 format program data HEX file and generates the built-in ROM data HEX file after filling the unused area of the built-in ROM (000000H–00EFFFH) with FFH. The generated file is used to debug the program with the ICE (S5U1C88000H5). When debugging with the ICE, download this file from the computer.
This file is also used as the program data to generate the mask data for submission to Seiko Epson by the mask data checker <winmdc>.

7.2 Input/output Files

Figure 7.2.1 shows the input/output files of fil88xxx.

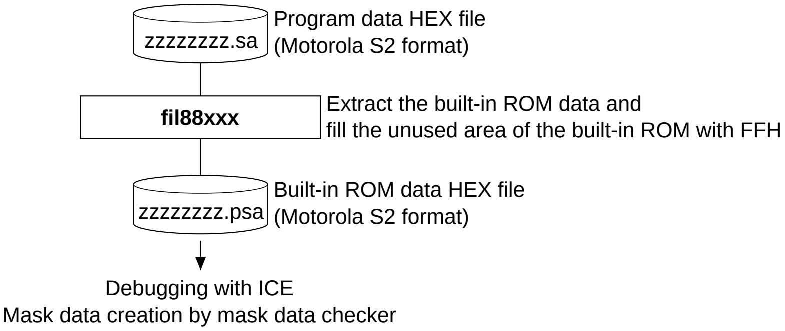

zzzzzzzz.sa
Program data HEX file
(Motorola S2 format)
fil88xxx
Extract the built-in ROM data and
fill the unused area of the built-in ROM with FFH
zzzzzzzz.psa
Built-in ROM data HEX file
(Motorola S2 format)
Debugging with ICE
Mask data creation by mask data checker

Fig. 7.2.1 Input/output files of fil88xxx

Program data HEX file (zzzzzzzz.sa)

This is a Motorola S2 format program data HEX file generated by the HEX converter <hex88> or a third party software tool.

Built-in ROM data HEX file (zzzzzzzz.psa)

This is a Motorola S2 format file that contains the built-in ROM data extracted from the input program data HEX file. The unused areas in the built-in ROM are filled with FFH and a system code is set to the system reserved area (see vector table shown in the Technical Manual). When debugging with the ICE, download this file from the computer. This file is packed along with completed other option files into a single file by the mask data checker <winmdc>, which we would like to have presented to Seiko Epson as the mask data file. From this file, Seiko Epson will create the mask patterns for the IC.

∗1 The "xxx" in the file name denotes the model name of a microcomputer. For the "zzzzzzzz" part, any given file name can be specified.
∗2 For details on how to download the built-in ROM data HEX file into the ICE, refer to the ICE manual.

7.3 Using fil88xxx

(1) Starting up

To start fil88xxx, enter the command shown below from the MS-DOS prompt.

>fil88xxx <file name> ↵

↵ denotes entering the return key.
Specify a Motorola S2 format program data HEX file as the command line parameter. A path can also be specified.
Example: `C:\S1C88\DEV88\DEV88xxx_V1>fil88xxx d:\test\c8xxx0a0.sa`

(2) Start-up message

When fil88xxx is started, the following message is displayed.

```
FIL88xxx Unused Area Filling Utility Version X.XX
Copyright (C) SEIKO EPSON CORP. xxxx
```

(3) End message

When a series of operation are complete, the fil88xxx displays the following message.

When terminated normally

```
...........................................          ... Indicates the proceeding status
Unused Area Filling Completed
System Area Data Set Completed
```

The converted HEX file (.psa) is generated in the same directory as the input file.

When an error has been occurred

```
C8xxx0A0.SA 5:         File Format Error              ... Example of error message
```

If an error is generated during fil88xxx execution, it displays the file name producing the error, the line number and an error message, then terminates the fil88xxx.
Also, when an error has been generated, a post-conversion program data HEX file (.psa) is not generated. In the event of a warning message, a post-conversion program data HEX file is generated.

(4) In the event of forced termination

To forcibly terminate the execution of the fil88xxx, enter "CTRL" + "C".

7.4 Error Messages

The error and warning messages of fil88xxx are listed below.

Table 7.4.1 Error messages

Message	Description
Can't Find File	The specified input file does not exist.
Syntax Error: Input File	An input file name has not been specified.
File Format Error	The input file format is wrong. (∗1)
Can't Open File	The input file cannot be opened.
Not S Record	The input file is not S record format.
Data Length	The data length of 1 line is too short.
Too Many Data In One Line	The data length of 1 line is too long.
Not 3Byte Address	The address length is not 3 bytes (including S1, S3, S7 and S9 record).
Check Sum Error	The check sum does not match.
Duplicate Error	There is a duplicate definition of data in the same address.
Can't Use Vector xxH System Reserve	The physical address 0000xxH cannot be used as a vector because they are reserved as a system area for the S1C88xxx.
Insufficient disk space	There is no disk space.
Write Error	An error has occurred while writing data.

∗1 A file format error will occur under the following conditions:
- Another record has followed the S8 record.
- Something other than a hexadecimal number is included in the file.
- There is a line that consists of less than 12 characters.
- There is an S8 record that has more or less than 12 characters or of which the byte count is not 04.
- There is an S4, S5 or S6 record included in the file.
- There is no S8 record.

Table 7.4.2 Warning message

Message	Description
Warning: No 00H Address	There is no data in the physical address 000000H.

Note: When there is no data in the physical address 000000H, it will output a warning message and filled the data FFH.

7.5 Example of Input/output Files

Input file example

```
S2240000000001235000502350235023502350235023502350235023502350235023502350235040
S2080000202350015013
S224000100CF6E00F6B4FFDD0030DD0100D94004C700F0C40000CFDCC30200D700F8E7F7F262
S2240001209300D94004B0FFB104C543F8C700F8CFEB7093CF3BE7FBC10001C20001CFEED725

:         :         :         :         :         :         :         :         :         :

S224007F001818000055AA000101000001000100010002000000401011121314151617181919
S20E007F201A1B1C1D1E1F2F3F0F3FEB
S804000000FB
```

Output file example

System code (e.g. F1H, FFH) are set in the system reserved area (e.g. addresses 000024H and 000025H) for S1C88xxx.

```
S2240000000001235000502350235023502350235023502350235023502350235023502350235040
S22400002023500150F1FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF21
S224000040FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFBB
S224000060FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF9B
S224000080FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF7B
S2240000A0FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF5B
S2240000C0FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF3B
S2240000E0FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF1B
S224000100CF6E00F6B4FFDD0030DD0100D94004C700F0C40000CFDCC30200D700F8E7F7F262
S2240001209300D94004B0FFB104C543F8C700F8CFEB7093CF3BE7FBC10001C20001CFEED725
S22400014000FEE7EEC500F8C600F8CFEE1255F5DAB000F23A04DD2003D94009DD22019C3F7C
S224000160B001CED400F0D94004F27A00F29000F2A600F2BC00F2DD00F21703F23F03F2BD28
S22400018003F2F203CED084F1803204E703B000CED484F1CED003F1803214E703B000CED462
S2240001A003F1CEAECED006F332FFE7F7B000CED406F3F1B3D97560CED0007F7810CED00143
S2240001C07F7811D97801CED0027F7844CED0037F7845DD62FFDD6000DD63F5DD613FD9768C
S2240001E010DD4008F8A2A0C60E7FB100CED084F1CF40464C02CEB0FC297802A8AAF8CED0CC
S22400020084F13203E608F2E503B000F106F2D403B0FFCED407F4F8A2A0C6127FB100CED0CB

:         :         :         :         :         :         :         :         :         :

S22400EFA0FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF6C
S22400EFC0FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF4C
S22400EFE0FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF2C
S804000000FB
```

CHAPTER 8 FUNCTION OPTION GENERATOR <winfog>

8.1 Outline of winfog

The S1C88 chip allows several hardware specifications such as I/O port functions to be selected as mask options. This helps you to configure the hardware of your product by changing the S1C88 chip's mask patterns according to its specifications.

The Function Option Generator <winfog> is the software tool for creating the files necessary to generate mask patterns. Its graphical user interface (GUI) ensures easy selection mask options. From the files created by winfog, Seiko Epson produces the mask patterns for the S1C88 chip.

In addition, simultaneously with this file, winfog can create mask option setup files (Motorola S2 format data) that are required when debugging programs with the ICE (S5U1C88000H5). When using the ICE to debug a program, you can download this file from the host computer, making it possible to materialize optional functions on the ICE that are equivalent to those on the actual IC.

8.2 Input/output Files

Figure 8.2.1 shows the input/output files of winfog.

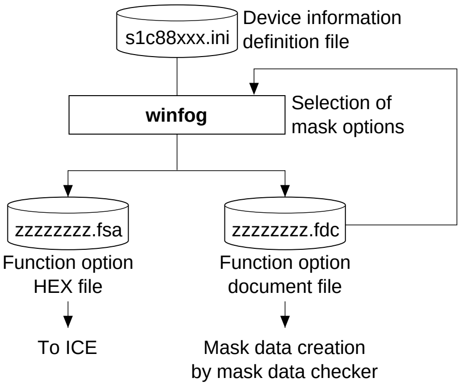

s1c88xxx.ini
Device information definition file
winfog
Selection of mask options
zzzzzzzz.fsa
Function option HEX file
To ICE
zzzzzzzz.fdc
Function option document file
Mask data creation by mask data checker

Fig. 8.2.1 Input/output files of winfog

Device information definition file (s1c88xxx.ini)

This file contains option lists for various types of microcomputers and other information. Always be sure to use the files presented by Seiko Epson. This file is effective for only the type of microcomputer indicated by the file name. Do not modify the contents of the file or use the file in other types of microcomputers.

Function option document file (zzzzzzzz.fdc)

This is a text format file in which the contents of selected mask options are stored. You can read this file into winfog and correct the already selected option settings. This file is packed along with completed other program/data files into a single file by the mask data checker <winmdc>, which we would like to have presented to Seiko Epson as the mask data file. From this file, Seiko Epson will create the mask patterns for the IC.

Function option HEX file (zzzzzzzz.fsa)

This is the Motorola S2 format file necessary to set the selected mask options in the ICE. When you debug programs with the ICE, download this file into the ICE using an ICE command.

∗1 The "xxx" in the file name denotes the model name of a microcomputer. For the "zzzzzzzz" part, any given file name can be specified.

∗2 For details on how to download mask options into the ICE, refer to the ICE manual.

8.3 Using winfog

8.3.1 Starting Up

Startup from Explorer

Double-click on the winfog.exe icon or select winfog from the start menu.
If the device information definition file (s1c88xxx.ini) was loaded into your computer during previous execution, winfog automatically reads the same file as it starts.
Alternatively, drag the Device information definition file icon into the winfog.exe icon to start winfog, which will then read the Device information definition file.

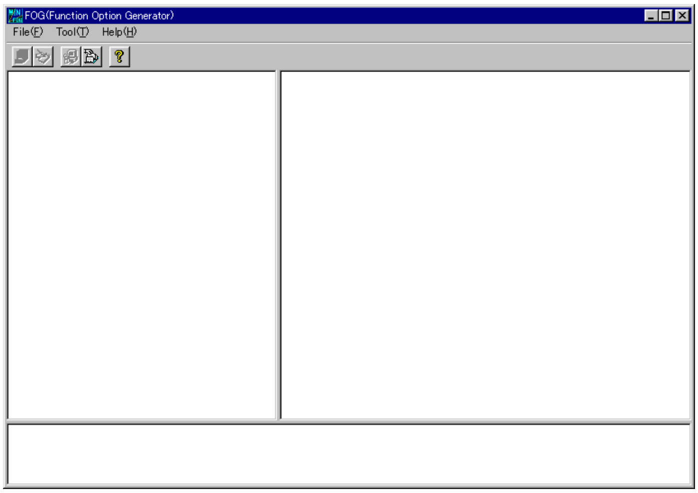
Startup by command input

You can also start winfog from the MS-DOS prompt by entering the command shown below.

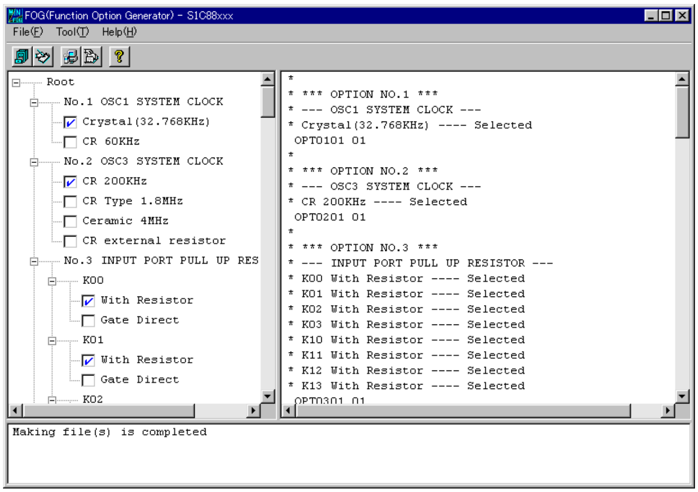
>winfog [s1c88xxx.ini] ↵

↵ denotes entering the return key.
You can specify the device information definition file (s1c88xxx.ini) as a command option. (You can also specify a path.) When you specify the device information definition file here, winfog reads it as it starts. This specification can be omitted.

When winfog starts, it displays the [FOG] window. The following diagrams show a [FOG] window when the device information definition file has been loaded and when it has not.



[FOG] Window (initial screen)



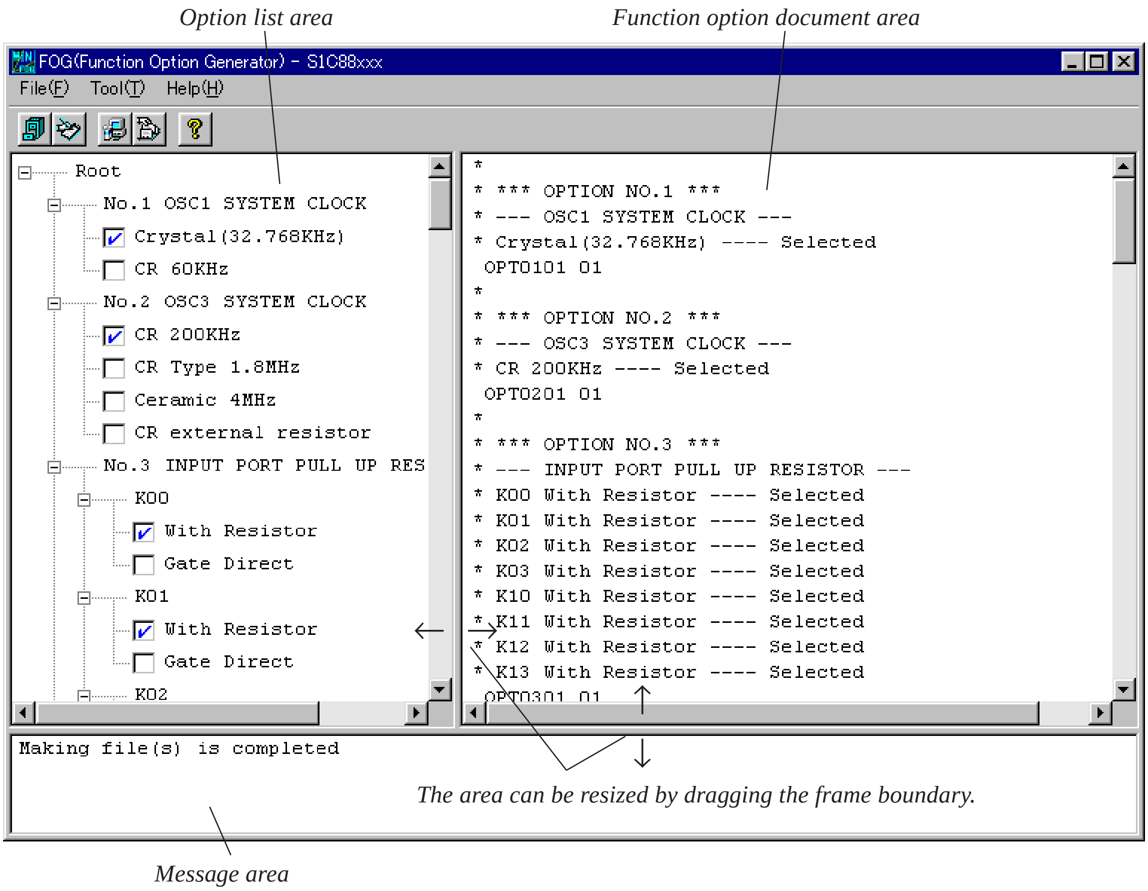
```
Root
  No.1 OSC1 SYSTEM CLOCK
    ☑ Crystal(32.768KHz)
    □ CR 60KHz
  No.2 OSC3 SYSTEM CLOCK
    ☑ CR 200KHz
    □ CR Type 1.8MHz
    □ Ceramic 4MHz
    □ CR external resistor
  No.3 INPUT PORT PULL UP RES
    K00
      ☑ With Resistor
      □ Gate Direct
    K01
      ☑ With Resistor
      □ Gate Direct
    K02
```

```
*
* *** OPTION NO.1 ***
* --- OSC1 SYSTEM CLOCK ---
* Crystal(32.768KHz) ---- Selected
 OPT0101 01
*
* *** OPTION NO.2 ***
* --- OSC3 SYSTEM CLOCK ---
* CR 200KHz ---- Selected
 OPT0201 01
*
* *** OPTION NO.3 ***
* --- INPUT PORT PULL UP RESISTOR ---
* K00 With Resistor ---- Selected
* K01 With Resistor ---- Selected
* K02 With Resistor ---- Selected
* K03 With Resistor ---- Selected
* K10 With Resistor ---- Selected
* K11 With Resistor ---- Selected
* K12 With Resistor ---- Selected
* K13 With Resistor ---- Selected
 OPT0301 01
```

```
Making file(s) is completed
```

[FOG] Window (after reading the device information definition file)

8.3.2 Window


Option list area
Function option document area
FOG(Function Option Generator) - S1C88xxx
File(F) Tool(T) Help(H)
The area can be resized by dragging the frame boundary.
Message area

```
Root
  No.1 OSC1 SYSTEM CLOCK
    ☑ Crystal(32.768KHz)
    □ CR 60KHz
  No.2 OSC3 SYSTEM CLOCK
    ☑ CR 200KHz
    □ CR Type 1.8MHz
    □ Ceramic 4MHz
    □ CR external resistor
  No.3 INPUT PORT PULL UP RES
    K00
      ☑ With Resistor
      □ Gate Direct
    K01
      ☑ With Resistor
      □ Gate Direct
    K02
```

```
*
* *** OPTION NO.1 ***
* --- OSC1 SYSTEM CLOCK ---
* Crystal(32.768KHz) ---- Selected
 OPT0101 01
*
* *** OPTION NO.2 ***
* --- OSC3 SYSTEM CLOCK ---
* CR 200KHz ---- Selected
 OPT0201 01
*
* *** OPTION NO.3 ***
* --- INPUT PORT PULL UP RESISTOR ---
* K00 With Resistor ---- Selected
* K01 With Resistor ---- Selected
* K02 With Resistor ---- Selected
* K03 With Resistor ---- Selected
* K10 With Resistor ---- Selected
* K11 With Resistor ---- Selected
* K12 With Resistor ---- Selected
* K13 With Resistor ---- Selected
 OPT0301 01
```

```
Making file(s) is completed
```

∗ The microcomputer model name on the title bar is the file name (not including the path and extension) of the device information definition file that has been read.

∗ The option list and the function option document vary with each type of microcomputer.

Fig. 8.3.2.1 Window configuration

The [FOG] window is divided into three areas as shown above.

Option list area

Lists mask options set in the device information definition file (s1c88xxx.ini). Use the check boxes in this area to select each option. A selected option has its check box marked by ✓.

Function option document area

Displays the contents of selected options in the function option document format. The contents displayed in this area are output to the function option document file. When you change any selected item in the option list area, the display in this area is immediately updated.

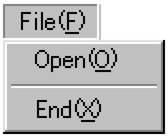
Message area

When you create a file by selecting [Generate] from the [Tool] menu or clicking the [Generate] button, this area displays a message showing the result of the selected operation.

8.3.3 Menus and Toolbar Buttons

This section explains each menu item and toolbar button.

[File] menu



Open

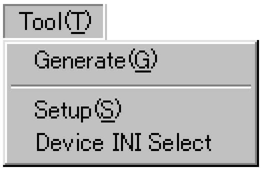
Opens a function option document file. Use this menu command when correcting an existing file. The [Open] button has the same function.

[Open] button

End

Terminates winfog.

[Tool] menu



Generate

Creates a file according to the selected contents of the option list. The [Generate] button has the same function.

[Generate] button

Setup

Sets the date of creation, output file name and a comment included in the function option document file. The [Setup] button has the same function.

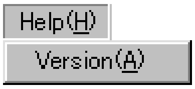
[Setup] button

Device INI Select

Loads the device information definition file <s1c88xxx.ini>. The [Device INI Select] button has the same function. This file must be loaded first before performing any operation with winfog.

[Device INI Select] button

[Help] menu

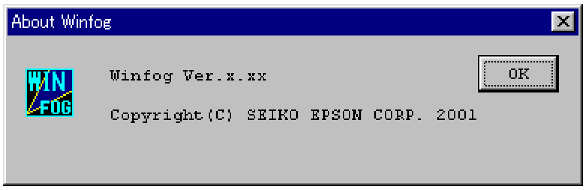


Version

Displays the version of winfog. The [Help] button has the same function.

[Help] button

The dialog box shown below appears. Click [OK] to close this dialog box.



Winfog Ver.x.xx

Copyright(C) SEIKO EPSON CORP. 2001

8.3.4 Operation Procedure

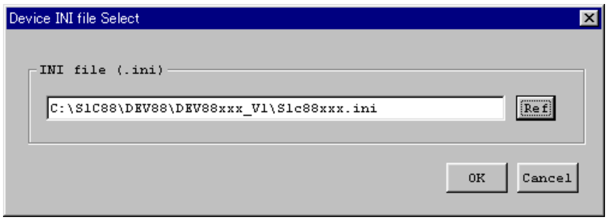
The following shows the basic operation procedure.

(1) Loading the device information definition file

First, select a device information definition file <s1c88xxx.ini> and load it.
Select [Device INI Select] from the [Tool] menu or click the [Device INI Select] button.

[Device INI Select] button

The dialog box shown below appears. Enter a file name including the path in the text box or select a file by clicking the [Ref] button.



Device INI file Select

INI file (.ini)

C:\S1C88\DEV88\DEV88xxx_V1\S1c88xxx.ini

Click [OK], and the file is loaded. If the specified file exists and there is no problem with its contents, the option list and the function option document, which have both been set by default, are displayed in each area.
To stop loading the file, click [Cancel].

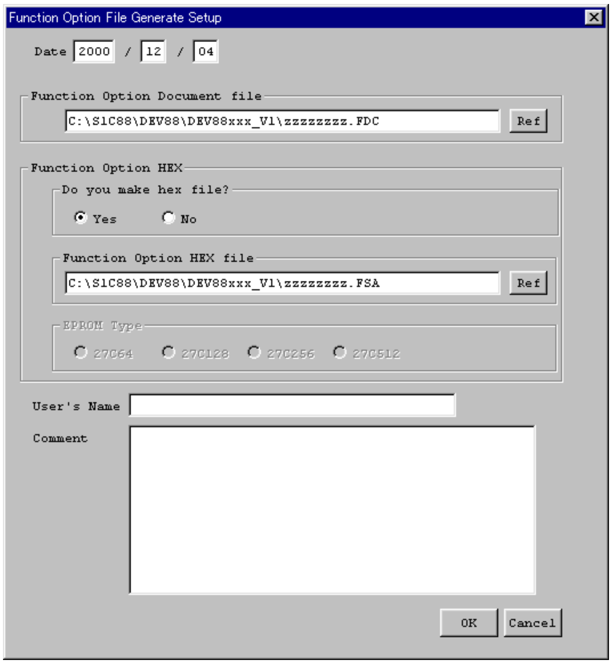
Once a device information definition file is selected, the same file is automatically loaded the next time you start winfog.

Note: When you load a device information definition file after setting up options, all settings are reset to the default state.

(2) Setup

Select [Setup] from the [Tool] menu or click the [Setup] button to bring up the [Setup] dialog box. From this dialog box, select items and enter data.

[Setup] button



Function Option File Generate Setup

Date 2000 / 12 / 04

Function Option Document file

C:\S1C88\DEV88\DEV88xxx_V1\zzzzzzzz.FDC

Function Option HEX

Do you make hex file?

● Yes ○ No

Function Option HEX file

C:\S1C88\DEV88\DEV88xxx_V1\zzzzzzzz.FSA

EPROM Type

○ 27C64 ○ 27C128 ○ 27C256 ○ 27C512

User's Name

Comment

Date
Displays the current date. Change it as necessary.

Function Option Document file
Specify the function option document file name you want to create. The file name displayed by default can be modified. You can use the [Ref] button to look at other folders.

Function Option HEX
Do you make hex file?
Select whether to create a function option HEX file. You need to create one when you use the ICE to debug programs.

Function Option HEX file
When you create a function option HEX file, specify its file name here. The file name displayed by default can be modified. You can use the [Ref] button to look at other folders.

EPROM Type
This option is not available for S1C88 Family microcomputers.

User's Name
Enter your company name. You can enter up to 40 characters. You can use English letters, numbers, symbols, and spaces. The content entered here is recorded in the USER'S NAME field of the function option document file.

Comment
Enter a comment. Up to 50 characters can be entered in one line. You can enter up to 10 lines. You can use English letters, numbers, symbols, and spaces. Use the [Enter] key to create a new line. All comments should include the following information:

- Place of business, your department or section
- Address, telephone number, and facsimile number
- Other: Technical information, etc.

The content entered here is recorded in the COMMENT field of the function option document file.
When you have finished entering the above necessary items, click [OK]. The setup contents are saved, and the dialog box is closed. The setup contents take effect immediately. If you click [Cancel], current settings will not be changed and the dialog box is closed.

Notes: • *File name specification is subject to the following limitations:*

1. *The number of characters that can be used to specify a file name including the path is 2,048.*
2. *The file name itself (not including the extension) can be up to 15 characters, and the extension up to three characters.*
3. *The file name cannot begin with a hyphen (-), nor can the following symbols be used as part of directory names (folder names), file names, and extensions:*
   / : , ; * ? " < > |

- *The symbols shown below cannot be used in the User's Name and Comment:*
  $ \ | `

(3) Selecting options

Select necessary options by clicking the corresponding check boxes in the option list. When you change any selection item in the option list area, the display in the function option document area is updated. Note that when you have loaded the device information definition file, the option list is placed in its default selection state.
For details about option specifications, refer to the Technical Manual available for each type of microcomputer.

(4) Creating files

After selecting options, select [Generate] from the [Tool] menu or click the [Generate] button to create the files.

[Generate] button

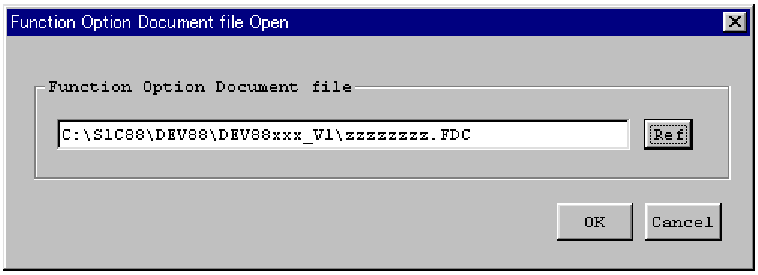
The function option document file you specified from the [Setup] dialog box and the function option HEX file (if specified) are created. When winfog has finished creating the files normally, it displays the message "Making file(s) is completed" in the message area. If an error occurs, an error message is displayed.

(5) Correcting an existing document file

You can read an existing function option document file into winfog and correct it as necessary.
To read a file, select [Open] from the [File] menu or click the [Open] button.

🗐 *[Open] button*

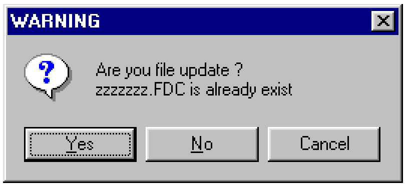
The dialog box shown below appears, so enter a file name including the path in the text box or select a file by clicking the [Ref] button.


Function Option Document file Open
Function Option Document file
Ref
OK
Cancel

C:\S1C88\DEV88\DEV88xxx_V1\zzzzzzzz.FDC

Click [OK], and the file is loaded. If the specified file exists and there is no problem with its contents, the option list and the function option document areas are updated according to the contents of the file. To stop loading the file, click [Cancel].

Perform steps (2) to (4) to update the file.
If you select [Generate] without changing the file name, the message shown below is displayed asking you whether or not to overwrite the file. Click [Yes] to overwrite or [No] or [Cancel] to stop overwriting. Use the [Setup] dialog box to change the file name.



Are you file update ?
zzzzzzz.FDC is already exist

Note: The function option document file can be read only when the device information definition file has been loaded.

(6) Quitting

To terminate winfog, select [End] from the [File] menu.

8.4 Error Messages

The error messages of winfog are listed below. The "Dialog" in the Display column means that messages are displayed in the dialog box, and "Message" means that messages are displayed in the [FOG] window message area.

Table 8.4.1 List of winfog error messages

Message	Description	Display
File name error	Number of characters in the file name or extension exceeds the limit.	Dialog
Illegal character	Prohibited characters have been entered.	Dialog
Please input file name	File name has not been entered.	Dialog
Can't open File : xxxx	File (xxxx) cannot be opened.	Dialog
INI file is not found	Specified device information definition file (.ini) does not exist.	Dialog
INI file does not include FOG information	Specified device information definition file (.ini) does not contain function option information.	Dialog
Function Option document file is not found	Specified function option document file does not exist.	Dialog
Function Option document file does not match INI file	Contents of the specified function option document file do not match device information definition file (.ini).	Dialog
A lot of parameter	Too many command line parameters are specified.	Dialog
Making file(s) is completed [xxxx is no data exist]	Finished creating the file, but the created file (xxxx) does not contain any data.	Message
Can't open File: xxxx Making file(s) is not completed	File (xxxx) cannot be opened when executing Generate.	Message
Can't write File: xxxx Making file(s) is not completed	File (xxxx) cannot be written when executing Generate.	Message

Table 8.4.2 winfog warning messages

Message	Description	Display
Are you file update? xxxx is already exist	Overwrite confirmation message (Specified file already exists.)	Dialog

8.5 Example Output Files

Note: Option and other configurations vary with each type of microcomputer.

Example of a function option document file

```
* S1C88xxx FUNCTION OPTION DOCUMENT Vx.xx    ← Version
*
* FILE NAME    zzzzzzzz.FDC                  ← File name (specified by [Setup])
* USER'S NAME  SEIKO EPSON CORPORATION       ← User name (specified by [Setup])
* INPUT DATE   yyyy/mm/dd                    ← Date of creation (specified by [Setup])
* COMMENT      SAMPLE DATA                   ← Comment (specified by [Setup])
*
* *** OPTION NO.1 ***                        ← Option number
* --- OSC1 SYSTEM CLOCK ---                  ← Option name
* Crystal(32.768KHz) ---- Selected           ← Selected specification
 OPT0101 01                                  ← Mask data
*
* *** OPTION NO.2 ***
* --- OSC3 SYSTEM CLOCK ---
* CR 200KHz ---- Selected
 OPT0201 01
*
* *** OPTION NO.3 ***
* --- INPUT PORT PULL UP RESISTOR ---
* K00 With Resistor ---- Selected
* K01 With Resistor ---- Selected
* K02 With Resistor ---- Selected
* K03 With Resistor ---- Selected
* K10 With Resistor ---- Selected
* K11 With Resistor ---- Selected
* K12 With Resistor ---- Selected
* K13 With Resistor ---- Selected
 OPT0301 01
 OPT0302 01
 OPT0303 01
 OPT0304 01
 OPT0305 01
 OPT0306 01
 OPT0307 01
 OPT0308 01
*
* *** OPTION NO.4 ***
* --- OUTPUT PORT OUTPUT SPECIFICATION ---
* R00 Complementary ---- Selected
* R01 Complementary ---- Selected
* R02 Complementary ---- Selected
* R03 Complementary ---- Selected
 OPT0401 01
 OPT0402 01
 OPT0403 01
 OPT0404 01
*
               :
*
* *** OPTION NO.8 ***
* --- SOUND GENERATOR POLARITY ---
* NEGATIVE ---- Selected
 OPT0801 01
*EOF                                         ← End mark
```

Example of a function option HEX file (Motorola S2 format)

```
S22400000022FF0200FFFFFFFFFFFFFFFFFFFFFFFFF00000000000000FFFFFFFFFFFFFFFFFFCD
S804000000FB
```

For details about the Motorola S2 format, refer to Section A.2.5.3, "Motorola S2 Format".

CHAPTER 9 SEGMENT OPTION GENERATOR <winsog>

9.1 Outline of winsog

Some types of microcomputers in the S1C88 Family allow the LCD output pin output specifications and LCD output pin assignments to be set with hardware options, so that mask patterns for the IC are generated according to option settings. The Segment Option Generator <winsog> is the software tool for creating the files required to generate mask patterns. Its graphical user interface (GUI) ensures simple mask option setting.
In addition, simultaneously with this file, winsog can create mask option setup files (Motorola S2 format data) that are required when debugging programs with the ICE (S5U1C88000H5). When using the ICE to debug a program, you can download this file from the host computer, making it possible to realize optional functions on the ICE that are equivalent to those on the actual IC.

Note: The Segment Option Generator <winsog> is provided for only certain types of microcomputers that have set segment options.

9.2 Input/output Files

Figure 9.2.1 shows the input/output files of winsog.

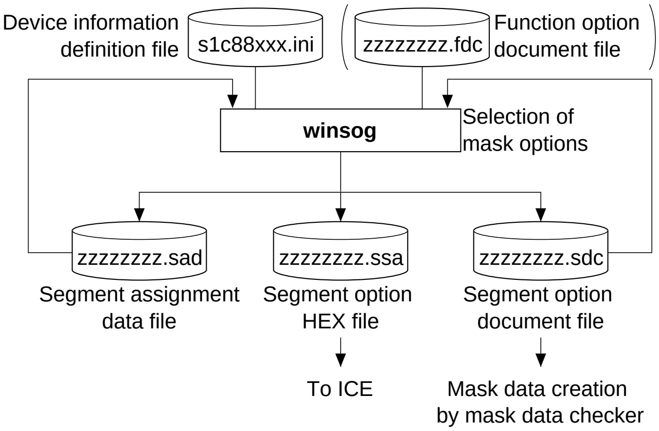

Device information definition file
s1c88xxx.ini
zzzzzzzz.fdc
Function option document file
winsog
Selection of mask options
zzzzzzzz.sad
zzzzzzzz.ssa
zzzzzzzz.sdc
Segment assignment data file
Segment option HEX file
Segment option document file
To ICE
Mask data creation by mask data checker

Fig. 9.2.1 Input/output files of winsog

Device information definition file (s1c88xxx.ini)

This file contains option lists for various types of microcomputers and other information. Always be sure to use the files presented by Seiko Epson. This file is effective for only the type of microcomputer indicated by the file name. Do not modify the contents of the file or use the file in other types of microcomputers.

Function option document file (zzzzzzzz.fdc)

This is the text format file generated by winfog and contains the selected mask options. This file is required only when the segment option setup condition depends on the mask option selected with winfog.

Segment option document file (zzzzzzzz.sdc)

This is a text format file in which setup contents of segment options are stored. You can read this file into winsog and correct the option settings. This file is packed along with completed other program/data files into a single file by the mask data checker <winmdc>, which will be presented to Seiko Epson as the mask data file. From this file, Seiko Epson will create the mask patterns for the IC.

Segment option HEX file (zzzzzzzz.ssa)

This is the Motorola S2 format file necessary to set the selected segment options in the ICE. When you debug programs with the ICE, download this file into the ICE using ICE commands.

Segment assignment data file (zzzzzzzz.sad)

This is a text format file in which segment assignment data is stored. Create this file when terminating winsog before finishing segment assignment. You can continue option setting next time by loading this file to winsog.

∗1 The "xxx" in the file name denotes the model name of a microcomputer. For the "zzzzzzzz" part, any given file name can be specified.
∗2 For details on how to download mask options into the ICE, refer to the ICE manual.

9.3 Using winsog

9.3.1 Starting Up

Startup from Explorer

winsog.exe

Double-click on the winsog.exe icon or select winsog from the start menu.
If the device information definition file (s1c88xxx.ini) was loaded into your computer during previous execution, winsog automatically reads the same file as it starts.
Alternatively, drag the device information definition file icon into the winsog.exe icon to start winsog, which will then read the device information definition file. If a function option document file is required for setting the segment option, a dialog box will appear to allow file selection. In this case enter the file name including the path in the text box or choose the file from the dialog box that appears by clicking on the [Ref] button.

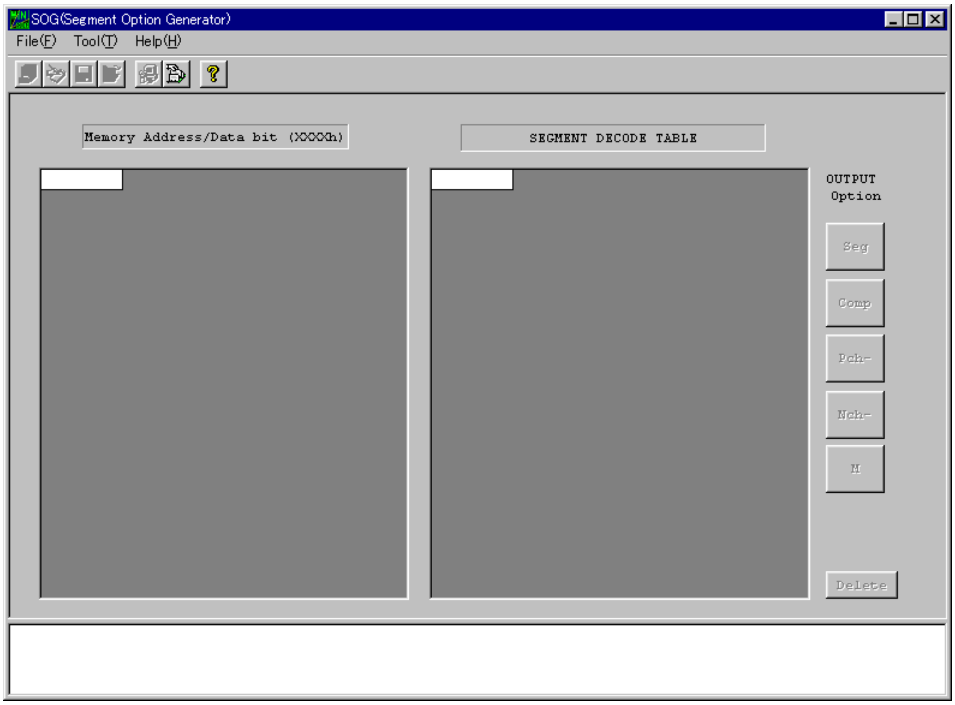
Startup by command input

You can also start winsog from the MS-DOS prompt by entering the command shown below.

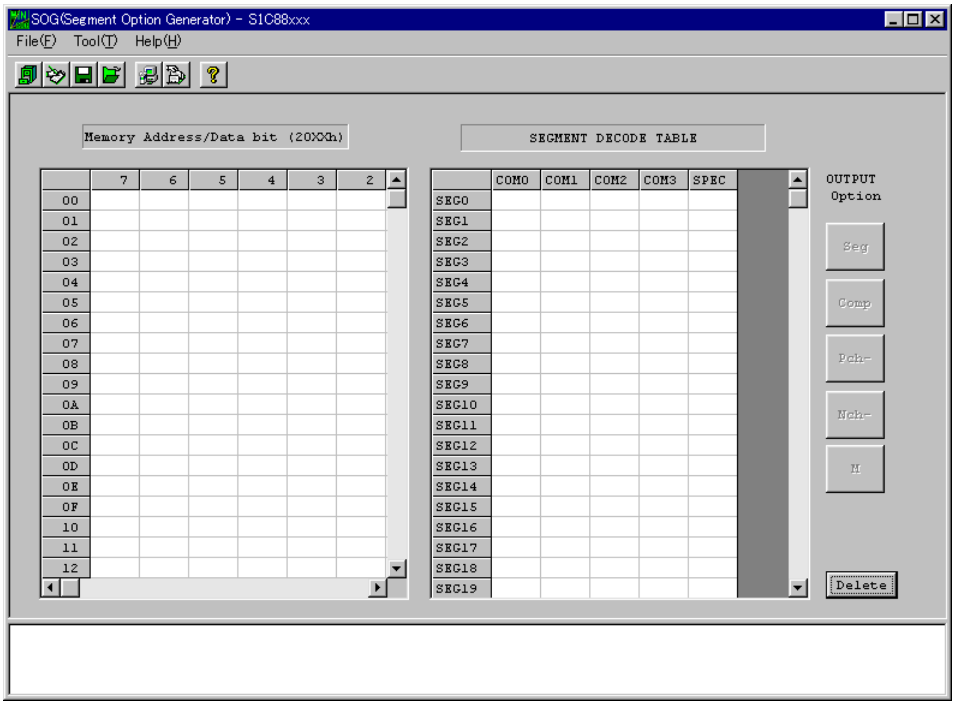
```
>winsog [s1c88xxx.ini] ↵
```

↵ denotes entering the return key.
You can specify the device information definition file (s1c88xxx.ini) as a command option. (You can also specify a path.) When you specify the device information definition file here, winsog reads it as it starts. If a function option document file is required for setting the segment option, the file (zzzzzzzz.fdc) must be prepared in the directory in which s1c88xxx.ini and winsog.exe exist before entering the command. When the command is entered, a dialog box will appear to allow file selection. Enter the file name including the path in the text box or choose the file from the dialog box that appears by clicking on the [Ref] button. This specification can be omitted.

When winsog starts, it displays the [SOG] window. The following diagrams show a [SOG] window when the device information definition file has been loaded and when it has not.



[SOG] Window (initial screen)

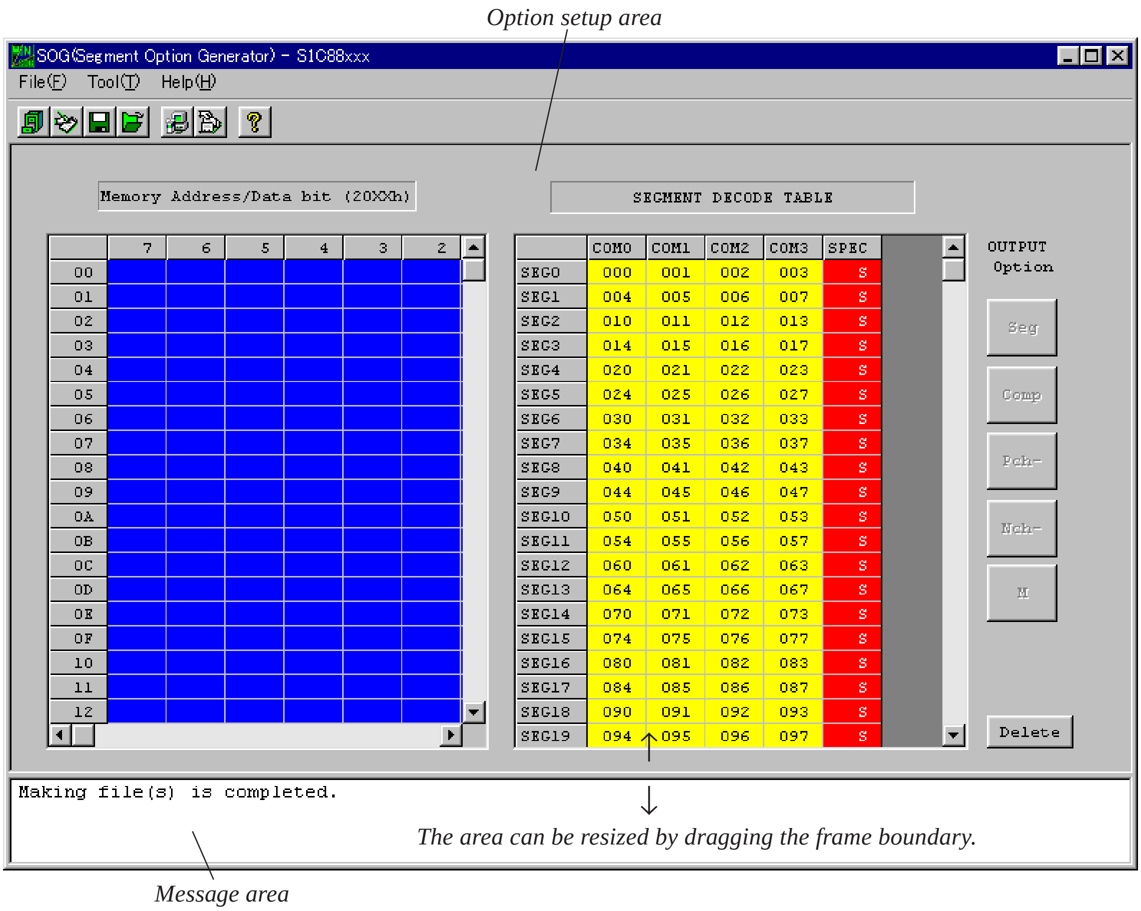


	7	6	5	4	3	2
00						
01						
02						
03						
04						
05						
06						
07						
08						
09						
0A						
0B						
0C						
0D						
0E						
0F						
10						
11						
12						

	COM0	COM1	COM2	COM3	SPEC
SEG0					
SEG1					
SEG2					
SEG3					
SEG4					
SEG5					
SEG6					
SEG7					
SEG8					
SEG9					
SEG10					
SEG11					
SEG12					
SEG13					
SEG14					
SEG15					
SEG16					
SEG17					
SEG18					
SEG19					

[SOG] Window (after reading the device information definition file)

9.3.2 Window


Option setup area
SOG(Segment Option Generator) - S1C88xxx
File(F) Tool(T) Help(H)
Memory Address/Data bit (20XXh)
SEGMENT DECODE TABLE
OUTPUT
Option
Seg
Comp
Pch-
Nch-
M
Delete
The area can be resized by dragging the frame boundary.
Message area

	7	6	5	4	3	2
00						
01						
02						
03						
04						
05						
06						
07						
08						
09						
0A						
0B						
0C						
0D						
0E						
0F						
10						
11						
12						

	COM0	COM1	COM2	COM3	SPEC
SEG0	000	001	002	003	S
SEG1	004	005	006	007	S
SEG2	010	011	012	013	S
SEG3	014	015	016	017	S
SEG4	020	021	022	023	S
SEG5	024	025	026	027	S
SEG6	030	031	032	033	S
SEG7	034	035	036	037	S
SEG8	040	041	042	043	S
SEG9	044	045	046	047	S
SEG10	050	051	052	053	S
SEG11	054	055	056	057	S
SEG12	060	061	062	063	S
SEG13	064	065	066	067	S
SEG14	070	071	072	073	S
SEG15	074	075	076	077	S
SEG16	080	081	082	083	S
SEG17	084	085	086	087	S
SEG18	090	091	092	093	S
SEG19	094	095	096	097	S

```
Making file(s) is completed.
```

* The microcomputer model name on the title bar is the file name (not including the path and extension) of the device information definition file that has been read.

* The display memory addresses and segment configuration vary with each type of microcomputer.

Fig. 9.3.2.1 Window configuration

The [SOG] window is divided into two areas as shown above.

Option setup area

Comprised of a display memory map, a segment decode table, and buttons to select pin specifications. By clicking on cells in the display memory map and segment decode table, you can assign display memory addresses and bits.

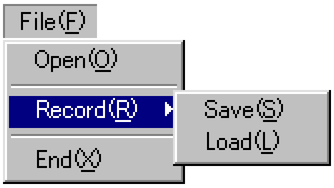
Message area

When you create a file by selecting [Generate] from the [Tool] menu or clicking the [Generate] button, this area displays a message showing the result of the selected operation.

9.3.3 Menus and Toolbar Buttons

This section explains each menu item and toolbar button.

[File] menu



Open

Opens a segment option document file. Use this menu command when correcting an existing file. The [Open] button has the same function.

[Open] button

Record - Save

Saves the current option settings to a file (segment assignment data file). The [Save] button has the same function.

[Save] button

Record - Load

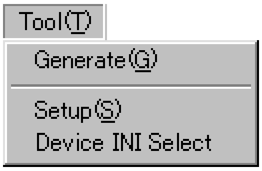
Loads a segment assignment data file. The [Load] button has the same function.

[Load] button

End

Terminates winsog.

[Tool] menu



Generate

Creates a file according to the contents of segment options set. The [Generate] button has the same function.

[Generate] button

Setup

Sets the date of creation or output file name or a comment included in the segment option document file. The [Setup] button has the same function.

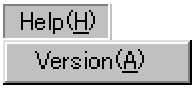
[Setup] button

Device INI Select

Loads the device information definition file <s1c88xxx.ini>. The [Device INI Select] button has the same function. This file must be loaded first before performing any operation with winsog.

[Device INI Select] button

[Help] menu

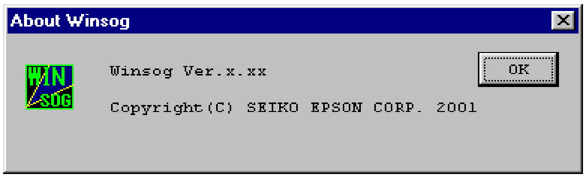


Version

Displays the version of winsog. The [Help] button has the same function.

[Help] button

The dialog box shown below appears. Click [OK] to close this dialog box.



Winsog Ver.x.xx

Copyright(C) SEIKO EPSON CORP. 2001

9.3.4 Option Selection Buttons

The following buttons are available in the option setup area.

OUTPUT Option buttons

These buttons select SEG pin output modes. These buttons are enabled when you click a SPEC cell in [SEGMENT DECODE TABLE].

Seg	Selects LCD segment output.
Comp	Selects DC-complementary output.
Pch-	Selects DC-Pch open-drain output.
Nch-	Selects DC-Nch open-drain output.
M	Selects segment/common shared output.

[Delete] button

Delete	Clears the selected segment assignment. The [Delete] key has the same function.

9.3.5 Operation Procedure

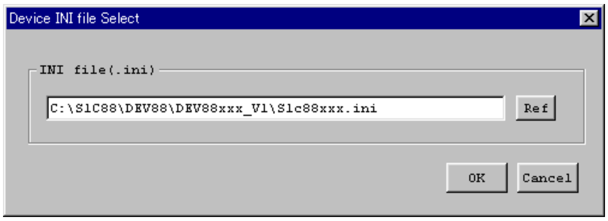
The following shows the basic operation procedure.

(1) Loading the device information definition file

First, select a device information definition file <s1c88xxx.ini> and load it.
Select [Device INI Select] from the [Tool] menu or click the [Device INI Select] button.

[Device INI Select] button

The dialog box shown below appears. Enter a file name including the path in the text box or select a file by clicking the [Ref] button.

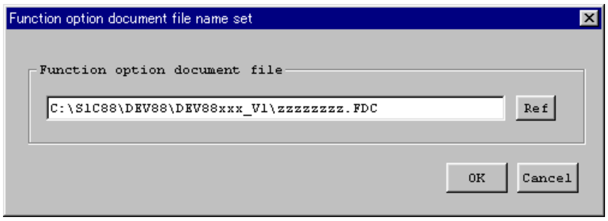


C:\S1C88\DEV88\DEV88xxx_V1\S1c88xxx.ini

Click [OK], and the file is loaded. If the specified file exists and there is no problem with its contents, the set-up items in winsog are initialized with the loaded device information.
To stop loading the file, click [Cancel].

Once a device information definition file is selected, the same file is automatically loaded the next time you start winfog.

If a function option document file is required for setting the segment option, the dialog box shown below will appear to allow file selection. In this case enter the file name including the path in the text box or choose the file from the dialog box that appears by clicking on the [Ref] button.



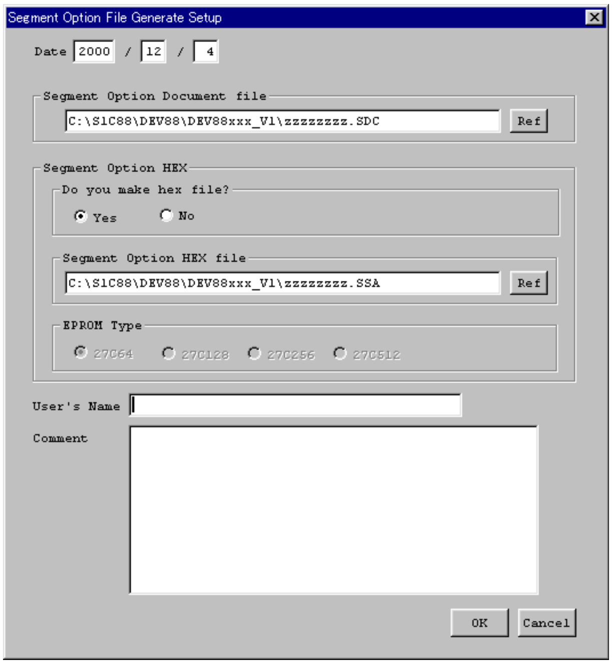
Function option document file: C:\S1C88\DEV88\DEV88xxx_V1\zzzzzzzz.FDC

Note: When you load a device information definition file after setting up options, all settings are reset to the default state.

(2) Setup

Select [Setup] from the [Tool] menu or click the [Setup] button to bring up the [Setup] dialog box. From this dialog box, select items and enter data.

[Setup] button



Date 2000 / 12 / 4

Segment Option Document file: C:\S1C88\DEV88\DEV88xxx_V1\zzzzzzzz.SDC

Segment Option HEX

Do you make hex file? ● Yes ○ No

Segment Option HEX file: C:\S1C88\DEV88\DEV88xxx_V1\zzzzzzzz.SSA

EPROM Type: ● 27C64 ○ 27C128 ○ 27C256 ○ 27C512

User's Name

Comment

Date

Displays the current date. Change it as necessary.

Segment Option Document file

Specify the segment option document file name you want to create. The file name displayed by default can be modified. You can use the [Ref] button to look at other folders.

Segment Option HEX

Do you make hex file?

Select whether to create a segment option HEX file. You need to create one when you use the ICE to debug programs.

Segment Options HEX file

When you create a segment option HEX file, specify its file name here. The file name displayed by default can be modified. You can use the [Ref] button to look at other folders.

EPROM Type

This option is not available for S1C88 Family microcomputers.

User's Name

Enter your company name. Up to 40 characters can be entered. You can use English letters, numbers, symbols, and spaces. The content entered here is recorded in the USER'S NAME field of the segment option document file.

Comment

Enter a comment. Up to 50 characters can be entered in one line. You can enter up to 10 lines. You can use English letters, numbers, symbols, and spaces. Use the [Enter] key to create a new line. All comments should include the following information:

- Place of business, your department or section
- Address, telephone number, and facsimile number
- Other: Technical information, etc.

The content entered here is recorded in the COMMENT field of the segment option document file. When you have finished entering the above necessary items, click [OK]. The setup contents are saved, and the dialog box is closed. The setup contents take effect immediately. If you click [Cancel], current settings will not be changed and the dialog box is closed.

Notes: • *File name specification is subject to the following limitations:*

1. *The number of characters that can be used to specify a file name including the path is 2,048.*
2. *The file name itself (not including the extension) can be up to 15 characters, and the extension up to three characters.*
3. *The file name cannot begin with a hyphen (-), nor can the following symbols be used as part of directory names (folder names), file names, and extensions:*
   / : , ; * ? " < > |

• *The symbols shown below cannot be used in the User's Name and Comment:*
$ \ | `

(3) Setting segment outputs

The LCD drive circuit of a S1C88 Family chip that has had segment options set normally allows selecting the segment output and DC output for every two pins (in certain types of microcomputers, individually for each pin). Segment output should be specified when using the pins for driving an LCD panel.

Segment output ports have a built-in segment decoder allowing any address and data bit in the display memory area to be assigned to any segment. When the segment memory bit is set to 1, the assigned segment lights up; when the bit is set to 0, the segment dims. Segments and display memory bits correspond individually, so that you cannot assign one display memory bit to multiple segments. Therefore, all segments must be assigned different addresses and data bits.

For details about the display memory map and segment assignment, refer to the Technical Manual for each type of microcomputer.

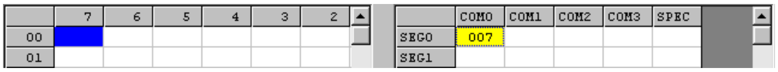
In the explanation below, the chip is assumed to have four common pins, COM0 to COM3.

Follow the procedure below to assign segments:

1. From the [Memory Address / Data bit] table, select the memory address / data bit you want to assign by clicking the appropriate cell. The cell changes color to blue.
   If you select an incorrect cell, select a correct cell.
   The horizontal rows of the table correspond to display memory addresses. The hexadecimal number shown to the right of the "Memory Address / Data bit" title is the base address of display memory, with only the lower byte of address being displayed in each row of the table. The vertical columns of the table correspond to data bits.

2. From [SEGMENT DECODE TABLE], select the SEG pin / COM pin to which you want to assign the memory address / data bit selected in 1 by clicking the appropriate cell. A 3-digit numeric value is displayed in the cell, showing the selected address (2 high-order digits) and data bit (1 low-order digit), and the cell changes color to yellow.

   Selection example:



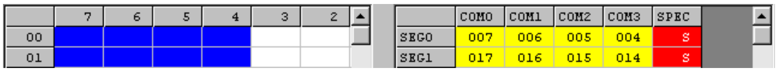
	7	6	5	4	3	2
00	🟦					
01						

	COM0	COM1	COM2	COM3	SPEC
SEG0	007				
SEG1					

   If you select an incorrect cell, click the [Delete] button to clear its assignment and reselect from 1. Two or more cells selected by dragging an area can also be deleted using the [Delete] button. Before selecting a cell in [SEGMENT DECODE TABLE], always select a cell in [Memory Address / Data bit].

3. Click the SPEC cell for the segment selected in 2 and then the [Seg] button. The cell shows the letter S and changes color to red. This means that the segment has been set for a LCD segment output pin.
   If your chip requires selecting segment output and DC output every two pins, the other pin that comprises a pair is set in the same way.

   Selection example:

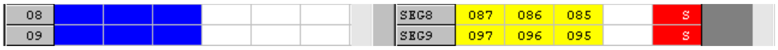


	7	6	5	4	3	2
00	🟦	🟦	🟦	🟦		
01	🟦	🟦	🟦	🟦		

	COM0	COM1	COM2	COM3	SPEC
SEG0	007	006	005	004	S
SEG1	017	016	015	014	S

4. Repeat steps 1 to 2 for all segments used for LCD output. Specification selection in 3 may be performed later.
   If any COM cell in one SEG pin is unused, leave it blank.

   Selection example:



08	🟦	🟦	🟦			
09	🟦	🟦	🟦			

SEG8	087	086	085		S
SEG9	097	096	095		S

(4) Setting DC outputs

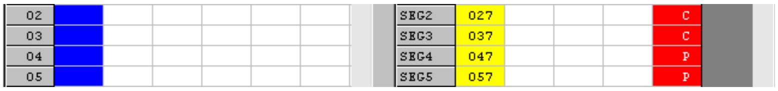
When using SEG pins for general-purpose DC output, assign segments according to steps 1 and 2 described in Item (3), "Setting segment outputs". However, output control works in such a way that the display memory assigned to COM0 is enabled while the display memory assigned to COM1 through COM3 are disabled. Therefore, set a memory address / data bit for only COM0 cell and leave memory address / data bits for COM1 through COM3 cells blank.

For DC output, you may select an output mode between complementary output and Nch (or Pch) open-drain output. Select your desired output in SPEC cell using the buttons listed below:

[Comp] button: Complementary output (C)
[Nch-] button: N-channel open-drain output (N)
[Pch-] button: P-channel open-drain output (P)

If your chip requires selecting an output mode every two pins, the other pin that comprises a pair is set in the same way.

Selection example:



02	🟦					
03	🟦					
04	🟦					
05	🟦					

SEG2	027				C
SEG3	037				C
SEG4	047				P
SEG5	057				P

(5) Setting SEG/COM shared pins

Whether the SEG / COM shared pins output segment signals or common signals is determined by selecting the function option.

When using the shared pins as SEG pins, allocate display memory addresses / bits as shown above and leave unused COM cells blank.

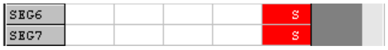
When using the shared pins as COM pins, select segment / common shared output ([M] button) as the output specification and do not allocate memory.

Note: This setting is required only for microcomputers that have SEG/COM shared pins.

(6) Setting unused SEG pins

For SEG pins that are used for neither LCD output nor DC output, leave COM0 through COM3 cells in [SEGMENT DECODE TABLE] blank. However, SPEC cells cannot be left blank, so select segment output (S) for the corresponding SPEC cells.

Selection example:



SEG6					S
SEG7					S

(7) Creating files

After selecting options, select [Generate] from the [Tool] menu or click the [Generate] button to create the files.

[Generate] button

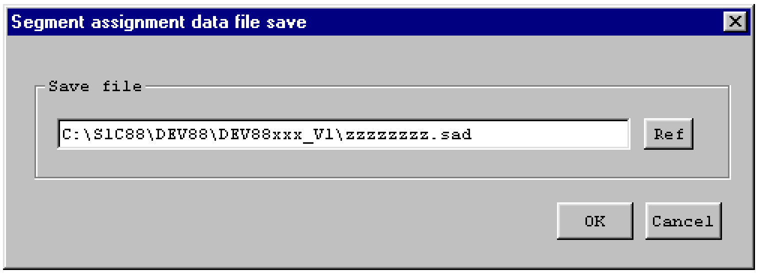
The segment option document file you specified from the [Setup] dialog box and the segment option HEX file (if specified) are created. When winsog has finished creating the files normally, it displays the message "Making file(s) is completed" in the message area. If an error occurs, an error message is displayed.

(8) Saving uncompleted segment option data

You can save the segment option settings that have not been completed as a segment assignment data file. To save data, select [Record - Save] from the [File] menu or click the [Save] button.

[Save] button

The dialog box shown below appears, so enter a file name including the path in the text box or select a file by clicking the [Ref] button.



Segment assignment data file save

Save file

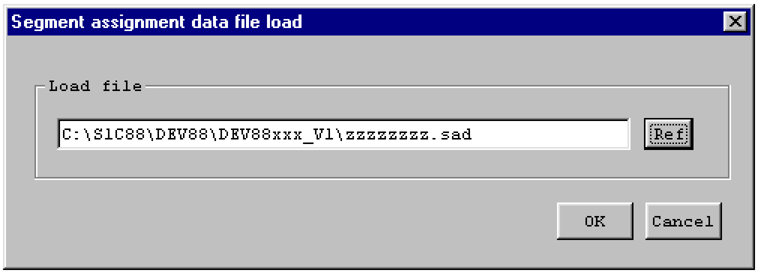
C:\S1C88\DEV88\DEV88xxx_V1\zzzzzzzz.sad

Clicking [OK] saves the current assignment data to the specified file. To stop saving, click [Cancel].

You can read an existing segment option document file into winsog and correct it as necessary.
To load a segment assignment data file, select [Record - Load] from the [File] menu or click the [Load] button.

[Load] button

The dialog box shown below appears, so enter a file name including the path in the text box or select a file by clicking the [Ref] button.



Segment assignment data file load

Load file

C:\S1C88\DEV88\DEV88xxx_V1\zzzzzzzz.sad

Click [OK], and the file is loaded. If the specified file exists and there is no problem with its contents, the option setup area is updated according to the segment assignment data saved in the file. You can continue segment assignment from the previous set state. To stop loading the file, click [Cancel].

Notes: • *The segment assignment data file can be read only when the device information definition file has been loaded.*

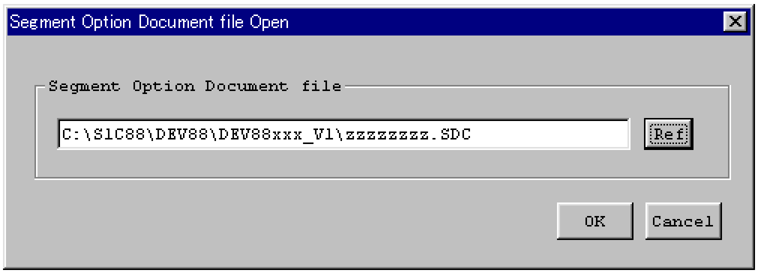
- *Some models need a function option document file to be loaded at the start of winsog, and the contents of the file affect the segment option setup condition. Therefore, the segment assignment data file in which the settings do not match the function option cannot be read.*

(9) Correcting an existing document file

You can read an existing segment option document file into winsog and correct it as necessary.
To read a file, select [Open] from the [File] menu or click the [Open] button.

[Open] button

The dialog box shown below appears, so enter a file name including the path in the text box or select a file by clicking the [Ref] button.



Segment Option Document file Open

Segment Option Document file

C:\S1C88\DEV88\DEV88xxx_V1\zzzzzzzz.SDC

Click [OK], and the file is loaded. If the specified file exists and there is no problem with its contents, [Memory Address / Data bit] and [SEGMENT DECODE TABLE] are updated according to the contents of the file. To stop loading the file, click [Cancel].

If you want to change an assigned address, clear its cell assignment using the [Delete] button first and then reassign a new address. If you want to change a selected output mode too, select the corresponding SPEC cell and clear its selected output mode with the [Delete] button before reselecting a new output mode. Two or more cells selected by dragging an area can also be deleted using the [Delete] button.
If you select [Generate] without changing the file name, the dialog box asking you whether to overwrite the file is displayed. Click [Yes] to overwrite or [No] or [Cancel] to stop overwriting. Use the [Setup] dialog box to change the file name.

Notes: • *The segment option document file can be read only when the device information definition file has been loaded.*

- *Some models need a function option document file to be loaded at the start of winsog, and the contents of the file affect the segment option setup condition. Therefore, the segment option document file in which the settings do not match the function option cannot be read.*

(10) Quitting

To terminate winsog, select [End] from the [File] menu.

9.4 Error Messages

The error messages of winsog are listed below. The "Dialog" in the Display column means that messages are displayed in the dialog box, and "Message" means that messages are displayed in the [SOG] window message area.

Table 9.4.1 List of winsog error messages

Message	Description	Display
File name error	Number of characters in the file name or extension exceeds the limit.	Dialog
Illegal character	Prohibited characters have been entered.	Dialog
Please input file name	File name has not been entered.	Dialog
Can't open File : xxxx	File (xxxx) cannot be opened.	Dialog
INI file is not found	Specified device information definition file (.ini) does not exist.	Dialog
INI file does not include SOG information	Specified device information definition file (.ini) does not contain segment option information.	Dialog
Function Option document file is not found	Specified function option document file does not exist.	Dialog
Function Option document file does not match INI file	Contents of the specified function option document file do not match device information definition file (.ini).	Dialog
Segment Option document file is not found	Specified segment option document file does not exist.	Dialog
Segment Option document file does not match INI file	Contents of the specified segment option document file do not match device information definition file (.ini).	Dialog
Segment assignment data file is not found	Specified segment assignment data file does not exist.	Dialog
Segment assignment data file does not match INI file	Contents of the specified segment assignment data file do not match device information definition file (.ini).	Dialog
Can't open File: xxxx Making file(s) is not completed	File (xxxx) cannot be opened when executing Generate.	Message
Can't write File: xxxx Making file(s) is not completed	File (xxxx) cannot be written when executing Generate.	Message
ERROR: SPEC is not set Making file(s) is not completed	One or more SPEC cells are left blank when executing Generate.	Message

Table 9.4.2 winsog warning messages

Message	Description	Display
Are you file update? xxxx is already exist	Overwrite confirmation message (Specified file already exists.)	Dialog

9.5 Example Output Files

Note: The display memory addresses, the number of SEG/COM pins, and output specification vary with each type of microcomputer.

Example of a segment option document file

```
* S1C88xxx SEGMENT OPTION DOCUMENT Vx.xx     ← Version
*
* FILE NAME    zzzzzzzz.SDC                  ← File name (specified by [Setup])
* USER'S NAME  SEIKO EPSON CORPORATION       ← User name (specified by [Setup])
* INPUT DATE   yyyy/mm/dd                    ← Date of creation (specified by [Setup])
* COMMENT      SAMPLE DATA                   ← Comment (specified by [Setup])
*
*
* OPTION NO.xx                               ← Option number (varies with type of microcomputer)
*
* < LCD SEGMENT DECODE TABLE >
*
* SEG COM0 COM1 COM2 COM3 SPEC
*
   0  163  162  161  1F3  S                  ← Segment decode table
   1  170  172  171  160  S
   2  143  142  141  1E1  S
   3  150  152  151  140  S
                 :
  xx  3B0  3B1  3B2  3B3  S
*EOF                                         ← End mark
```

Example of a segment assignment data file

```
* S1C88xxx SEGMENT OPTION DOCUMENT Vx.xx     ← Version
*
* FILE NAME    zzzzzzzz.SDC                  ← File name (specified by [Setup])
* USER'S NAME                                ← User name (specified by [Setup])
* INPUT DATE   yyyy/mm/dd                    ← Date of creation (specified by [Setup])
* COMMENT                                    ← Comment (specified by [Setup])
*
*
* OPTION NO.xx                               ← Option number (varies with type of microcomputer)
*
* < LCD SEGMENT DECODE TABLE >
*
* SEG COM0 COM1 COM2 COM3 SPEC
*
   0  163  162  161  1F3  S                  ← Segment data has been assigned
   1  170  172  171  160  S
   2  143  142  141  1E1  S
                 :
  mm  FRE  FRE  FRE  FRE  X                  ← FRE: Segment address and data bit have not been assigned.
  nn  FRE  FRE  FRE  FRE  X                  ← X: Output specification has not been set.
  oo  FRE  FRE  FRE  FRE  X
*EOF                                         ← End mark
```

Example of a segment option HEX file (Motorola S2 format)

```
S2240000001603160216011F03FFFFFFFFFFFFFFFF1700170217011600FFFFFFFFFFFFFFFF23
S2240000201403140214011E01FFFFFFFFFFFFFFFF1500150215011400FFFFFFFFFFFFFFFF14
                                     :
S2240010E0FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF0B
S804000000FB
```

For details about the Motorola S2 format, refer to Section A.2.5.3, "Motorola S2 format".

CHAPTER 10 MASK DATA CHECKER <winmdc>

10.1 Outline of winmdc

The Mask Data Checker <winmdc> is the software tool for checking the format of each generated file and creating the files necessary to generate mask patterns. winmdc checks the built-in ROM data HEX file generated by program unused area filling utility <fil88xxx>, the function option document file generated by function option generator <winfog>, and the segment option document file generated by segment option generator <winsog>.
winmdc also has a function for restoring the created mask data file into the original file format.

10.2 Input/output Files

Figure 10.2.1 shows the input/output files of winmdc.

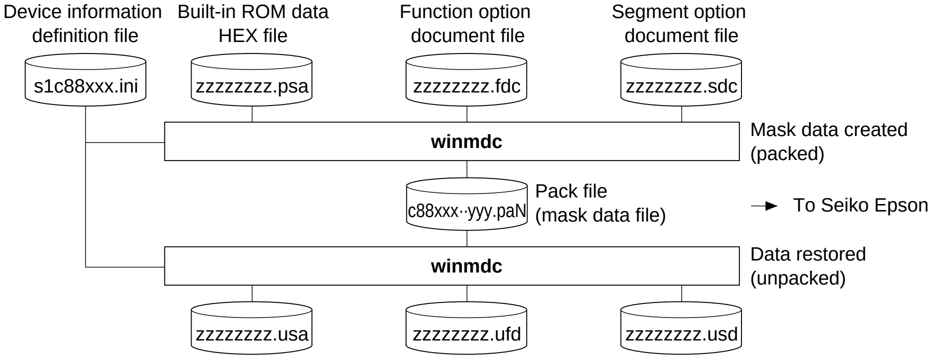

Device information definition file
Built-in ROM data HEX file
Function option document file
Segment option document file
s1c88xxx.ini
zzzzzzzz.psa
zzzzzzzz.fdc
zzzzzzzz.sdc
winmdc
Mask data created (packed)
c88xxx··yyy.paN
Pack file (mask data file)
To Seiko Epson
winmdc
Data restored (unpacked)
zzzzzzzz.usa
zzzzzzzz.ufd
zzzzzzzz.usd

Fig. 10.2.1 Input/output files of winmdc

Device information definition file (s1c88xxx.ini)

This file contains option lists for various types of microcomputers and other information. Always be sure to use the files presented by Seiko Epson. This file is effective for only the type of microcomputer indicated by the file name. Do not modify the contents of the file or use the file in other types of microcomputers.

Built-in ROM data HEX file (zzzzzzzz.psa)

This is the built-in ROM data HEX file in Motorola S2 format. This file is created by program unused area filling utility <fil88xxx>. The unused areas in the built-in ROM are filled with FFH and a system code is set to the system reserved area (see vector table shown in the Technical Manual).

Function option document file (zzzzzzzz.fdc)

This is a text format file in which the contents of selected function options are stored. This file is created by function option generator <winfog>.

Segment option document file (zzzzzzzz.sdc)

This is a text format file in which the contents of segment options set are stored. It is created by segment option generator <winsog>. This file is available for only microcomputers with set segment options.

Pack file (c88xxx··yyy.paN, N = 0 and over)

This is a text format file which contains the above data files combined into one. We would like to have this file presented to Seiko Epson as the mask data file. Seiko Epson will create the mask patterns for the IC from this mask data file.

∗ The "xxx··" in the file name denotes the model name of a microcomputer. The "yyy" part of the file name represents the custom code of each customer. Enter the code from Seiko Epson here. For the "zzzzzzzz" part, any given file name can be specified.

10.3 Using winmdc

10.3.1 Starting Up

Startup from Explorer

Double-click on the winmdc.exe icon or select winmdc from the start menu.

Winmdc.exe

If the device information definition file (s1c88xxx.ini) was loaded into your computer during a previous execution, winmdc automatically reads the same file as it starts. Alternatively, drag the device information definition file icon into the winmdc.exe icon to start winmdc, which will then read the device information definition file.

Startup by command input

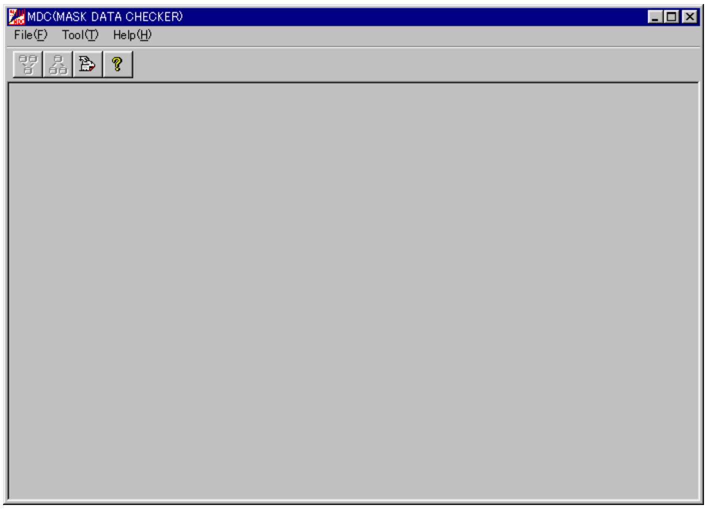
You can also start winmdc from the MS-DOS prompt by entering the command shown below.

>winmdc [s1c88xxx.ini] ↵

↵ denotes entering the return key.

You can specify the device information definition file (s1c88xxx.ini) as a command option. (You can also specify a path.) When you specify the Device information definition file here, winmdc reads it as it starts. This specification can be omitted.

When winmdc starts, it displays the [MDC] window.



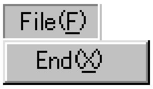
[MDC] Window (initial screen)

- * The microcomputer model name on the title bar is the file name (not including the path and extension) of the device information definition file that has been read.
- * The [Pack] and [Unpack] buttons on the tool bar are enabled when the device information definition file is read.

10.3.2 Menus and Toolbar Buttons

This section explains each menu item and toolbar button.

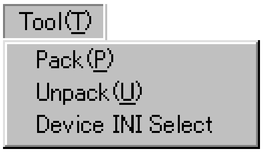
[File] menu



End

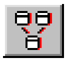
Terminates winmdc.

[Tool] menu



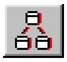
Pack

Packs the ROM data file and option document file to create a mask data file for presentation to Seiko Epson. The [Pack] button has the same function.


[Pack] button

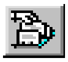
Unpack

Restores files in the original format from a packed file. The [Unpack] button has the same function.

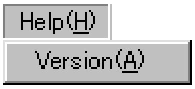

[Unpack] button

Device INI Select

Loads the device information definition file <s1c88xxx.ini>. The [Device INI Select] button has the same function. This file must be loaded first before performing any operation with winmdc.

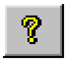

[Device INI Select] button

[Help] menu

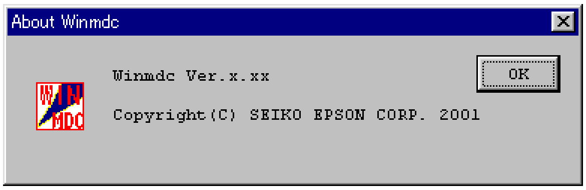


Version

Displays the version of winmdc. The [Help] button has the same function.


[Help] button

The dialog box shown below appears. Click [OK] to close this dialog box.



Winmdc Ver.x.xx

Copyright(C) SEIKO EPSON CORP. 2001

10.3.3 Operation Procedure

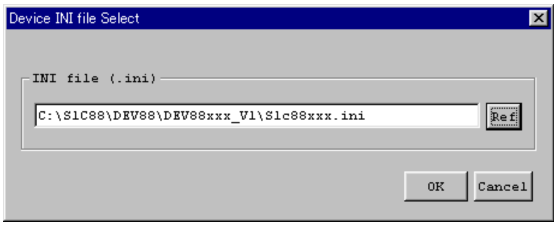
The following shows the basic operation procedure.

(1) Loading the Device information definition file

First, select a device information definition file <s1c88xxx.ini> and load it.
Select [Device INI Select] from the [Tool] menu or click the [Device INI Select] button.

[Device INI Select] button

The dialog box shown below appears. Enter a file name including the path in the text box or select a file by clicking the [Ref] button.


Device INI file Select

INI file (.ini)

C:\S1C88\DEV88\DEV88xxx_V1\S1c88xxx.ini

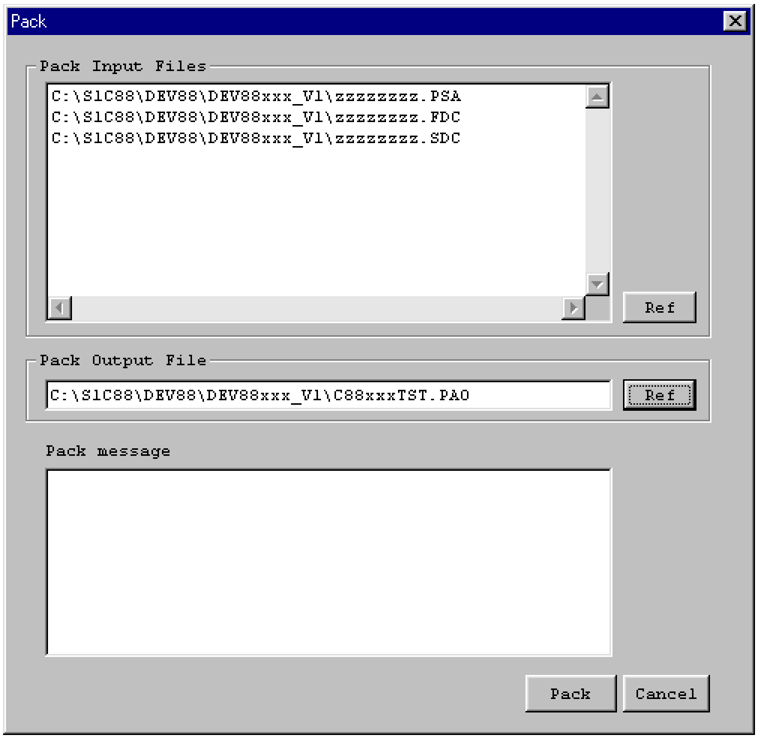
Click [OK], and the file is loaded. If the specified file exists and there is no problem with its contents, the set-up items in winmdc are initialized with the loaded device information.
To stop loading the file, click [Cancel].

Once a device information definition file is selected, the same file is automatically loaded the next time you start winmdc.

(2) Packing

1. Select [Pack] from the [Tool] menu or click the [Pack] button on the tool bar to bring up the [Pack] dialog box.

   [Pack] button


Pack

Pack Input Files

C:\S1C88\DEV88\DEV88xxx_V1\zzzzzzzz.PSA
C:\S1C88\DEV88\DEV88xxx_V1\zzzzzzzz.FDC
C:\S1C88\DEV88\DEV88xxx_V1\zzzzzzzz.SDC

Pack Output File

C:\S1C88\DEV88\DEV88xxx_V1\C88xxxTST.PAO

Pack message

2. Select the files to be entered.
   [Pack Input Files] lists the files of the type specified in the device information definition file by their default file names. If the data files to be entered are represented by different names in this list, replace the file names following the procedure below.
   a. Select a file name to be changed by clicking on it from the list box.
   b. Click the [Ref] button and select the data file to be entered.
   Do this for all files listed.
   When replacing files, take care not to mistake one file type (extension) for another. If the type of input file is erroneous, an error will result during file packing.

3. Setting output file names.
   In the [Pack Output File] text box, specify a pack file name in which you want the mask data to be output. The file name displayed by default can be modified. You can use the [Ref] button to look at other folders.
   Make sure the extension of the output file name is ".pa0". If after presenting data to Seiko Epson, you present new data due to program bugs or any other reason, increase the number in the last digit of the extension in increments of one. For example, the extension of the second file presented should be "c88xxx··yyy.pa1".

Note: File name specification is subject to the following limitations:

1. The number of characters that can be used to specify a file name including the path is 2,048.

2. The file name itself (not including the extension) can be up to 15 characters, and the extension up to three characters.

3. The file name cannot begin with a hyphen (-), nor can the following symbols be used as part of directory names (folder names), file names, and extensions:
/ : , ; * ? " < > |

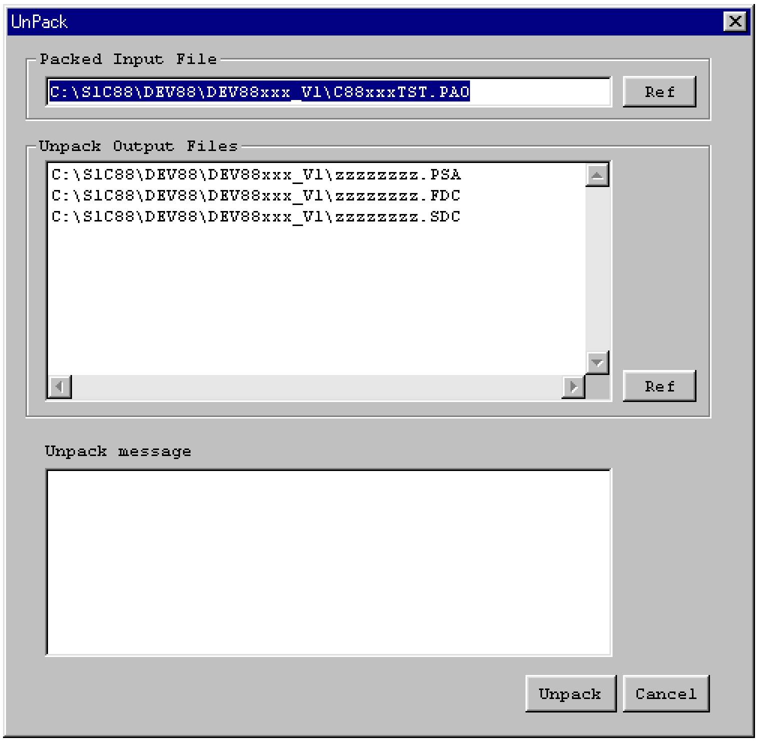
4. Click the [Pack] button to execute packing.
   When winmdc has completed packing, it displays a message "Packing completed!" in the [Pack message] text box. If an error has occurred, an error message is displayed.

5. Click the [Cancel] button to close the dialog box.
   Alternatively, you can click the [Cancel] button to quit winmdc before it executes packing.

(3) Unpacking

1. Select [Unpack] from the [Tool] menu or click the [Unpack] button on the tool bar to bring up the [Unpack] dialog box.

   [Unpack] button



Packed Input File: C:\S1C88\DEV88\DEV88xxx_V1\C88xxxTST.PA0

Unpack Output Files:

```
C:\S1C88\DEV88\DEV88xxx_V1\zzzzzzzz.PSA
C:\S1C88\DEV88\DEV88xxx_V1\zzzzzzzz.FDC
C:\S1C88\DEV88\DEV88xxx_V1\zzzzzzzz.SDC
```

Unpack message

2. Select the file you want to unpack.
   In the [Packed Input File] text box, specify the pack file name you want to enter. Use the names displayed by default to specify this file name after changing one, or select another file using the [Ref] button.

3. Select the output file name.
   [Unpack Output Files] lists the files of the type specified in the device information definition file by their default file names. Modify the file name displayed by the following procedure.
   a. Click in the list box to select the file name to be modified.
   b. Click the [Ref] button to select another folder, and then enter a file name. Modify all the listed file names. The extensions cannot be changed.

4. Click the [Unpack] button to execute unpacking.
   When winmdc has completed unpacking, it displays a message "Unpacking completed!" in the [Unpack message] text box. If an error has occurred, an error message is displayed.

5. Click the [Cancel] button to close the dialog box.
   Alternatively, you can click the [Cancel] button to quit winmdc before it executes unpacking.

(4) Quitting

To terminate winmdc, select [End] from the [File] menu.

10.4 Error Messages

The error messages of winmdc are listed below. The "Dialog" in the Display column means that messages are displayed in the dialog box, and "Message" means that messages are displayed in the message area of the [Pack] or [Unpack] dialog box.

Table 10.4.1 List of I/O error messages

Message	Description	Display
File name error	Number of characters in the file name or extension exceeds the limit.	Dialog
Illegal character	Prohibited characters have been entered.	Dialog
Please input file name	File name has not been entered.	Dialog
INI file is not found	Specified device information definition file (.ini) does not exist.	Dialog
INI file does not include MDC information	Specified device information definition file (.ini) does not contain MDC information.	Dialog
Can't open file : xxxx	File (xxxx) cannot be opened.	Dialog
Can't write file: xxxx	File (xxxx) cannot be written.	Dialog

Table 10.4.2 List of ROM data error messages

Message	Description	Display
Hex data error: Not S record.	Data does not begin with "S."	Message
Hex data error: Data is not sequential.	Data is not listed in ascending order.	Message
Hex data error: Illegal data.	Invalid character is included.	Message
Hex data error: Too many data in one line.	Too many data entries exist in one line.	Message
Hex data error: Check sum error.	Checksum does not match.	Message
Hex data error: ROM capacity over.	Data is large. (Greater than ROM size)	Message
Hex data error: Not enough the ROM data.	Data is small. (Smaller than ROM size)	Message
Hex data error: Illegal start mark.	Start mark is incorrect.	Message
Hex data error: Illegal end mark.	End mark is incorrect.	Message
Hex data error: Illegal comment.	Model name shown at the beginning of data is incorrect.	Message

Table 10.4.3 List of function option data error messages

Message	Description	Display
Option data error : Illegal model name.	Model name is incorrect.	Message
Option data error : Illegal version.	Version is incorrect.	Message
Option data error : Illegal option number.	Option No. is incorrect.	Message
Option data error : Illegal select number.	Selected option number is incorrect.	Message
Option data error : Mask data is not enough.	Mask data is insufficient.	Message
Option data error : Illegal start mark.	Start mark is incorrect.	Message
Option data error : Illegal end mark.	End mark is incorrect.	Message

Table 10.4.4 List of segment option data error messages

Message	Description	Display
LCD segment data error : Illegal model name.	Model name is incorrect.	Message
LCD segment data error : Illegal version.	Version is incorrect.	Message
LCD segment data error : Illegal segment No.	Segment No. is incorrect.	Message
LCD segment data error : Illegal segment area.	Display memory address is out of range.	Message
LCD segment data error : Illegal segment output specification.	Specified output mode is incorrect.	Message
LCD segment data error : Illegal data in this line.	Data written here is not hexadecimal number or output mode.	Message
LCD segment data error : Data is not enough.	Segment data is insufficient.	Message
LCD segment data error : Illegal start mark.	Start mark is incorrect.	Message
LCD segment data error : Illegal end mark.	End mark is incorrect.	Message

10.5 Example Output File

Note: The configuration and contents of data vary with each type of microcomputer.

Example of a pack file (mask data file)

```
*
* S1C88xxx MASK DATA VER x.xx                ← Version
*
\ROM1                                        ← Built-in ROM HEX data start mark
S1C88xxxyyy PROGRAM ROM                      ← Model name
S224000000...............................
   :       :       :       :       :
S804000000FB                                 "zzzzzzzz.psa"
S224000000...............................
   :       :       :       :       :
S804000000FB
\END                                         ← Built-in ROM HEX data end mark
\OPTION1                                     ← Function option start mark
* S1C88xxx FUNCTION OPTION DOCUMENT V x.x    ← Model name/version
*
* FILE NAME    zzzzzzzz.FDC
* USER'S NAME  SEIKO EPSON CORPORATION
* INPUT DATE   yyyy/mm/dd
* COMMENT      SAMPLE DATA
*                                            "zzzzzzzz.fdc"
* *** OPTION NO.1 ***
* --- OSC1 SYSTEM CLOCK ---
* Crystal(32.768KHz) ---- Selected
 OPT0101 01
   :       :       :       :       :
 OPTnn01 01
*EOF
\END                                         ← Function option end mark
\SEGMENT1                                    ← Segment option start mark
* S1C88xxx SEGMENT OPTION DOCUMENT Vx.xx     ← Model name/version
*
* FILE NAME    zzzzzzzz.SDC
* USER'S NAME  SEIKO EPSON CORPORATION
* INPUT DATE   yyyy/mm/dd
* COMMENT      SAMPLE DATA
*
*
* OPTION NO.xx
*                                            "zzzzzzzz.sdc"
* < LCD SEGMENT DECODE TABLE >
*
* SEG COM0 COM1 COM2 COM3 SPEC
*
   0  163  162  161  1F3  S
   1  170  172  171  160  S
               :
  xx  3B0  3B1  3B2  3B3  S
*EOF
\END                                         ← Segment option end mark
```

CHAPTER 11 SELF-DIAGNOSTIC PROGRAM <t88xxx>

11.1 Outline of t88xxx

t88xxx is a self-diagnostic program to check the operation of the hardware tools ICE (S5U1C88000H5) and S5U1C88xxxP that are used for program debugging of the S1C88 Family.
Perform a self-diagnostic of the ICE and S5U1C88xxxP periodically using this program.

11.2 File Configuration

(1) Program data HEX file (t88xxx.psa)

This is the main file of the self-diagnostic program generated by fil88xxx, in which the unused area of the built-in ROM is filled with FFH and the system code is set to the system reserved area of the S1C88xxx.

(2) Function option HEX file (t88xxx.fsa)

This is the file generated by winfog to set the function option into the ICE and S5U1C88xxxP, and is used at self-diagnosis.

(3) Function option document file (t88xxx.fdc)

This is the document file corresponding to the function option HEX file shown above and is generated by winfog.

(4) Segment option HEX file (t88xxx.ssa)

This is the file generated by winsog to set the segment option into the ICE and S5U1C88xxxP, and is used at self-diagnosis.

(5) Segment option document file (t88xxx.sdc)

This is the document file corresponding to the segment option HEX file shown above and is generated by winsog.

Note that the segment option files (4 and 5) are provided for only certain types of microcomputers that have set segment options.

(6) readme.txt

This file contains the description of the S5U1C88xxxP LED illumination status to check the operation with the self-diagnostic program.

11.3 Operation Procedure

After installing S5U1C88xxxP into the ICE, self-diagnosis of the ICE and S5U1C88xxxP can be done by the following operation test.
For the following operation test, the self-diagnostic program (t88xxx.psa) and the function option HEX data (t88xxx.fsa) in this package are used. In addition to these files, the segment option HEX data (t88xxx.ssa) is required for testing the microcomputer model that supports segment option.
Perform the below operations.

(1) Execute the self-diagnostic program (t88xxx.psa), the function option HEX data (t88xxx.fsa) and the segment option HEX data (t88xxx.ssa) after downloading them into the ICE.
Refer to the ICE manual for downloading and executing programs.

(2) Check the LEDs on the S5U1C88xxxP. If the LEDs light in the sequence described in readme.txt after a system reset, it is normal. The "cycle count" described in readme.txt indicates a 1 second interval and the LEDs change their light status every second.

CHAPTER 12 88xxx.par FILE

The 88xxx.par file is a macro file that contains the information for each model. The ICE (S5U1C88000H5) sets its operating environment by loading this parameter file. Therefore, the ICE cannot start up if this parameter file does not exist.

12.1 Contents of 88xxx.par File

The following shows a sample parameter file.

```
[Options]
Prcclksel=0               ...(1)
Vdddown=0                 ...(2)
CC=0                      ...(3)
DIAG=0                    ...(4)

[MAP Config]
;S1C88xxx MAP Configuration Setting
;  000000-00FFFF:Define 1 byte unit
;  010000-FFFFFF:Define 256 bytes unit
;
;syntax:<Start address> <End address> [E][I][U][S][W]
;        E:Emulation memory
;        I:I/O (PRC Board) memory
;        U:User memory
;        S:Stack area
;        W:Write protect (Default does not protect)

;Internal ROM
Map0=000000 00EFFF E W    ...(5)

;Internal RAM
Map1=00F000 00F3FF E

;Stack area
Map2=00F400 00F5FF E S

;Display memory
Map3=00F800 00F828 I
Map4=00F833 00F842 I
Map5=00F900 00F928 I
Map6=00F933 00F942 I
Map7=00FA00 00FA28 I
Map8=00FA33 00FA42 I
Map9=00FB00 00FB28 I
Map10=00FB33 00FB42 I
Map11=00FC00 00FC28 I
Map12=00FC33 00FC42 I
Map13=00FD00 00FD28 I
Map14=00FD33 00FD42 I

;I/O memory
Map15=00FF00 00FF02 I
Map16=00FF10 00FF12 I
Map17=00FF20 00FF25 I
Map18=00FF30 00FF34 I
Map19=00FF35 00FF36 I W
Map20=00FF40 00FF40 I
Map21=00FF41 00FF41 I W
Map22=00FF42 00FF42 I
Map23=00FF43 00FF43 I W
Map24=00FF44 00FF45 I
Map25=00FF48 00FF4A I
Map26=00FF50 00FF53 I
Map27=00FF54 00FF55 I W
Map28=00FF61 00FF61 I
Map29=00FF63 00FF63 I
Map30=00FF70 00FF71 I
Map31=00FF75 00FF75 I
Map32=00FF78 00FF78 I
```

12.2 Description of the Parameters

The parameters (1) to (4) are system reserved items, so do not modify their settings. Parameter (5) and the following parameters are used to set the memory allocations and memory conditions.

General format:

Map<Serial number> = <Start address> <End address> <Switch>

Serial number

The Map parameter must have a serial number within the range from 0 to 1023.
The serial numbers must not be specified in a special order.
If a number is duplicated, the parameter set first is enabled and the others are disabled.

Address settings

Addresses can be set in byte units for the range from 000000 to 00FFFF. Areas exceeding 010000 should be done using 256 byte units. (****00–****FF).

Switch

The following five letters are available for specifying <Switch>: E, I, U, S and W.

- ***Switches for allocating memories (E, I, U switches)***
  The I switch allocates the specified address area to the memory on the S5U1C88xxxP board.
  The E switch allocates the specified area to the emulation memory on the ICE.
  The U switch allocates the specified area to the user’s memory on the target board.
- ***Switch for setting stack area (S switch)***
  The S switch sets the specified area as a stack area.
- ***Write-protect switch/specifying ROM area (W switch)***
  The W switch sets the specified area as a ROM area that cannot be written. When an area is specified without the W switch, the ICE will regard it as a RAM area.

Comments

The ICE identifies a line that begins with a semicolon (;) as a comment line. Comments cannot be placed following parameters.

```
Example: ;Internal ROM                               ... OK
         Map0=000000 00EFFF EW ;internal ROM         ... NG
```

12.3 Emulation Memory

The ICE has built-in a 64KB emulation memory for the memory space from 000000 to 00FFFF and a 512KB emulation memory in S5U1C88000H5 or a 256KB emulation memory in S5U1C88000H3 that can be used as an expanded memory area exceeding address 010000. The emulation memory allows the user to use it as a memory that will be connected externally in the actual product. Thus it is not necessary to mount the external memory on the target board to develop the program. However, prepare the external memory on the target board when developing a product that needs a larger memory than 512KB at a location exceeding address 010000.

Notes

- It is therefore necessary to edit the path description in the ice88*.ini (* = r or ur) file located in the Windows system folder. When the 88xxx.par file exists in the same folder as the ice88*.exe file, only the file name part should be modified.
  Installation of ICE88* for Windows makes the default.par file in the same folder as the ice88*.exe file installed and sets the path information in the ice88*.ini file so that the debugger will refer to the default.par file.
- The parameters (1) to (4) must be described in the part that begins with an [Options] line and the parameters following (5) must be described after the [MAP Config] line. Do not delete [Options] and [MAP Config].

CHAPTER 13 S1C88 FAMILY DEBUGGER

13.1 Overview

The db88 debugger is a development tool for the S1C88 Family of 8-bit single-chip microcomputers. The debugger included in this package allows you to debug software created with the S1C88 integrated tool (C compiler, assembler) using the in-circuit emulator (S5U1C88000H5).

The debugger has the following features and functions:

- Various data can be referenced at the same time using multiple windows.
- Frequently used commands can be executed from tool bars and menus using a mouse.
- Also available are C source, disassembled code and symbol display functions.
- Consecutive program execution and three types of single-stepping are possible.
- Three break functions are supported.
- Trace and coverage functions.
- An automatic command execution function using a command file.

13.2 Input/output Files

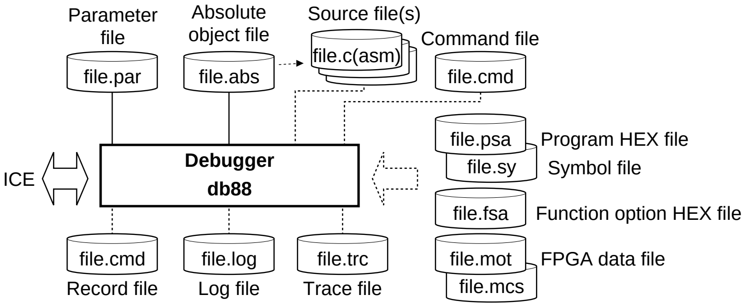


Fig. 13.2.1 Input/output files

Parameter file (file_name.par)

This text file contains memory information on each microcomputer model and is used to set the memory mapping information to the ICE. For the contents of this file, refer to Chapter 12, "88xxx.par File".

Absolute object file (file_name.abs)

This is an IEEE-695 object file generated by the advanced locator or locator. By reading a file in this format that contains debug information, C source display and symbolic debugging can be performed.

Source file (file_name.c, file_name.asm)

This is the source file of the above object file. It is read when the debugger performs source display.

Internal ROM data HEX file (file_name.psa)

This is the program file generated by the fil88xxx unused ROM area FF filling utility in Motorola S2 format file. The unused area of the built-in ROM has been filled with FFH and the system code is set to the system reserved area.

Symbol information file (file_name.sy)

This is the symbol information file generated by the symbol table file generator. By preparing the file with the same name as the internal ROM data HEX file in the same directory, it will be automatically loaded at the same time the internal ROM data HEX file is loaded. This file allows the debugger to display the symbols defined in the source.

Function option HEX file (file_name.fsa)

This is the mask option setup file in Motorola S2 format that is generated by the function option generator.

FPGA data file (file_name.mot, file_name.mcs)

This data file is used to configure the FPGA on the peripheral board S5U1C88000P for a S1C88 Family model. ".mot" is a Motorola S2 format file and ".mcs" is an Intel HEX format file.

Command file (file_name.cmd)

This text file contains a description of debug commands to be executed successively. By writing a series of frequently used commands in this file, the time and labor required for entering commands from the keyboard can be saved. The command described in the file are read and executed using the com command.

Log file (file_name.log)

This text file contains the executed commands and execution results. Output of this file can be controlled by the log command.

Record file (file_name.cmd)

This text file contains the executed commands. Output of this file can be controlled by the rec command. This command can be used as a command file.

Trace file (file_name.trc)

This text file contains the specified range of trace information. Output of this file can be controlled by the tf command.

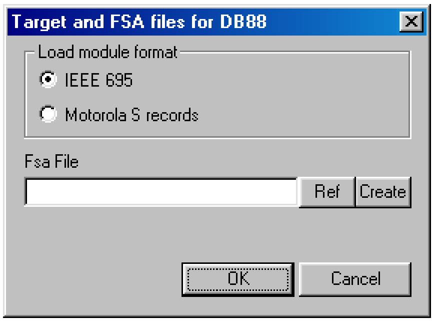
13.3 Starting and Terminating the Debugger

13.3.1 Starting the Debugger

Connect the ICE (S5U1C88000H5) to a personal computer and turn the power on before starting up the debugger.
The debugger can be started up using one of the following methods:

Starting from Work Bench

After the build process of the project has completed, select [DB88 Debugger] from the [Debug] menu or click the [DB88] button. The dialog box shown below appears.



Target and FSA files for DB88

Load module format
● IEEE 695
○ Motorola S records

Fsa File

Select the absolute object file format (IEEE 695 or Motorola S) using the radio button.
Select a function option HEX file from the dialog box displayed by clicking the [Ref] button, or enter a function option HEX file name directly into the [Fsa File] text box.
The [Create] button invokes the function option generator winfog to generate a new function option HEX file.
After these items have been selected / entered, click the [OK] button to launch the debugger.

Starting from Windows Explorer

DB88.exe

Double-click this icon to start the debugger.

Starting from MS-DOS prompt

Enter the command shown below to start the debugger.

db88 ∧ [<parameter file name>] ∧ [<command file name>]

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

Example: `C:\epson\s1c88\\db88\db88 par88xxx.par startup.cmd`

Note: The parameter file and command file will be recognized by their extensions ".par" and ".com", so the extension must be included in the file name to be specified.

When the debugger starts up, it outputs the following message in the [Command] window.

```
DB88 Ver x.xx
Copyright SEIKO EPSON CORP. 2001

Parameter file: xxxxxxxx.par
Initialize........................ OK
>
```

When the tests and initialization of the ICE have been finished, the debugger displays "OK" and is ready to execute a debugger command. When the debugger is invoked from the Work Bench, the specified object file is loaded after the tests have been finished. The state of the screen including the position and size of the windows will return the same as the last time the debugger was terminated.

Note: If the ICE is in self-diagnosis state (when the ICE is turned on with the DIAG switch set to on position), the debugger does not display "OK" until the diagnosis is finished. The self-diagnosis takes about 40 seconds for the process.

If "NG" is displayed, restart the debugger after checking the following conditions:

- The USB cable is connected properly
- The USB driver for the ICE is installed
- The peripheral board is correctly fitted in place
- The ICE's power is turned on
- The ICE remains reset

13.3.2 Terminating the Debugger

To terminate the debugger, select [Exit] from the [File] menu.

You can also input the q command in the [Command] window to terminate the debugger.

```
>q
```

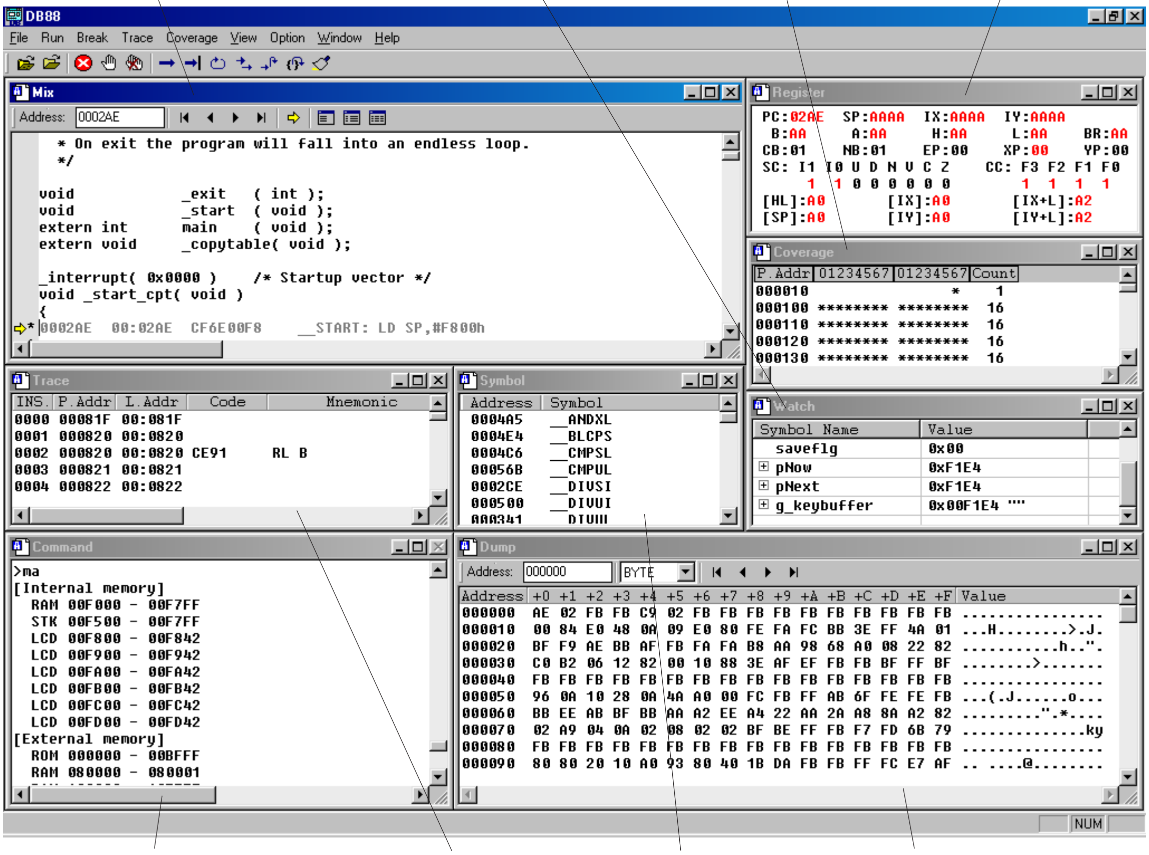
13.4 Windows

This section describes the types of windows used by the debugger.

13.4.1 Basic Structure of Window

The diagram below shows the window structure of the debugger.

[Source] window *[Watch] window* *[Coverage] window* *[Register] window*



```
 * On exit the program will fall into an endless loop.
 */

void            _exit   ( int );
void            _start  ( void );
extern int      main    ( void );
extern void     _copytable( void );

_interrupt( 0x0000 )    /* Startup vector */
void _start_cpt( void )
{
*0002AE  00:02AE  CF6E00F8    __START: LD SP,#F800h
```

```
PC:02AE   SP:AAAA   IX:AAAA   IY:AAAA
 B:AA      A:AA      H:AA      L:AA      BR:AA
CB:01     NB:01     EP:00     XP:00     YP:00
SC: I1 I0 U D N V C Z       CC: F3 F2 F1 F0
     1  1 0 0 0 0 0 0            1  1  1  1
[HL]:A0        [IX]:A0        [IX+L]:A2
[SP]:A0        [IY]:A0        [IY+L]:A2
```

```
P.Addr 01234567 01234567 Count
000010                 *     1
000100 ******** ********    16
000110 ******** ********    16
000120 ******** ********    16
000130 ******** ********    16
```

```
Symbol Name     Value
  saveflg       0x00
+ pNow          0xF1E4
+ pNext         0xF1E4
+ g_keybuffer   0x00F1E4 ""
```

```
INS. P.Addr L.Addr  Code     Mnemonic
0000 00081F 00:081F
0001 000820 00:0820
0002 000820 00:0820 CE91     RL B
0003 000821 00:0821
0004 000822 00:0822
```

```
Address   Symbol
0004A5    __ANDXL
0004E4    __BLCPS
0004C6    __CMPSL
00056B    __CMPUL
0002CE    __DIVSI
000500    __DIVUI
```

```
>ma
[Internal memory]
  RAM 00F000 - 00F7FF
  STK 00F500 - 00F7FF
  LCD 00F800 - 00F842
  LCD 00F900 - 00F942
  LCD 00FA00 - 00FA42
  LCD 00FB00 - 00FB42
  LCD 00FC00 - 00FC42
  LCD 00FD00 - 00FD42
[External memory]
  ROM 000000 - 00BFFF
  RAM 080000 - 080001
```

```
Address +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +A +B +C +D +E +F Value
000000  AE 02 FB FB C9 02 FB FB FB FB FB FB FB FB FB FB ................
000010  00 84 E0 48 0A 09 E0 80 FE FA FC BB 3E FF 4A 01 ...H........>.J.
000020  BF F9 AE BB AF FB FA FA B8 AA 98 68 A0 08 22 82 ...........h..".
000030  C0 B2 06 12 82 00 10 88 3E AF EF FB FB BF FF BF ........>.......
000040  FB FB FB FB FB FB FB FB FB FB FB FB FB FB FB FB ................
000050  96 0A 10 28 0A 4A A0 00 FC FB FF AB 6F FE FE FB ...(.J......o...
000060  BB EE AB BF BB AA A2 EE A4 22 AA 2A A8 8A A2 82 ........".*.....
000070  02 A9 04 0A 02 08 02 02 BF BE FF FB F7 FD 6B 79 ..............ky
000080  FB FB FB FB FB FB FB FB FB FB FB FB FB FB FB FB ................
000090  80 80 20 10 A0 93 80 40 1B DA FB FB FF FC E7 AF ..  ....@........
```

[Command] window *[Trace] window* *[Symbol] window* *[Dump] window*

Features common to all windows

(1) Open/close and activating a window

All windows except [Command] can be closed or opened.

To open a window, select the window name from the [View] menu. When a command is executed, the corresponding window opens if the command uses the window for displaying the executed results.

To close a window, click the [Close] box on the window.

The opened windows are listed in the [Window] menu. Selecting one from the list activates the selected window. It can also be done by simply clicking on an inactive window. Furthermore, pressing [Ctrl]+[Tab] switches the active window to the next open window.

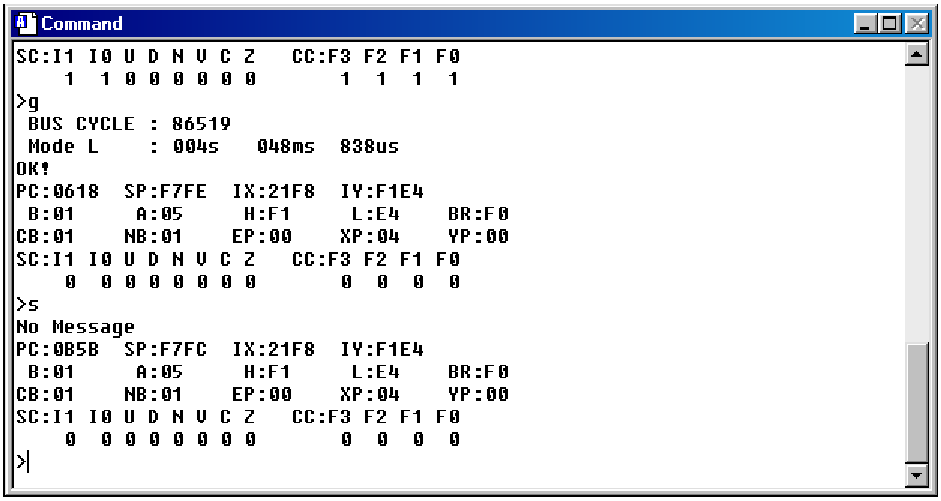
(2) Resizing and moving a window

Each window can be resized as needed by dragging the boundary of the window with the mouse. The [Minimize] and [Maximize] buttons work in the same way as in general Windows applications. Each window can be moved to the desired display position by dragging the window's title bar with the mouse. However, windows can only be resized and moved within the range of the application window.

(3) Other

The opened windows can be cascaded or tiled using the [Window] menu.

13.4.2 [Command] Window



```
SC:I1 I0 U D N V C Z   CC:F3 F2 F1 F0
    1  1 0 0 0 0 0 0       1  1  1  1
>g
 BUS CYCLE : 86519
 Mode L    : 004s   048ms  838us
OK!
PC:0618  SP:F7FE  IX:21F8  IY:F1E4
 B:01     A:05     H:F1     L:E4    BR:F0
CB:01    NB:01    EP:00    XP:04    YP:00
SC:I1 I0 U D N V C Z   CC:F3 F2 F1 F0
    0  0 0 0 0 0 0 0       0  0  0  0
>s
No Message
PC:0B5B  SP:F7FC  IX:21F8  IY:F1E4
 B:01     A:05     H:F1     L:E4    BR:F0
CB:01    NB:01    EP:00    XP:04    YP:00
SC:I1 I0 U D N V C Z   CC:F3 F2 F1 F0
    0  0 0 0 0 0 0 0       0  0  0  0
>
```

The [Command] window is used to do the following:

(1) Entering debug commands

When the prompt ">" appears in the [Command] window, the system will accept a command entered from the keyboard.

(2) Displaying debug commands selected from menus or tool bar

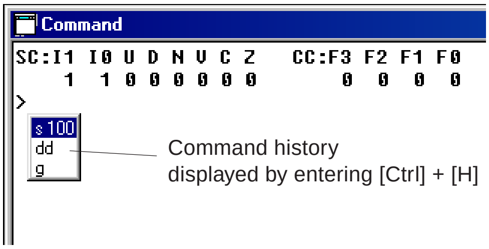
When a command is executed by selecting the menu item or tool bar button, the executed command line is displayed in the [Command] window.

(3) Displaying command execution results

The [Command] window displays command execution results. However, some command execution results are displayed in other windows. The contents of these execution results are displayed when their corresponding windows are open. If the corresponding window is closed, the execution result is displayed in the [Command] window.
When writing to a log file, the content of the write data is displayed in the window. (Refer to the description for log command.)

(4) Displaying the command history

db88 stores up to the 32 most recent commands executed since startup in memory. (If any command has been executed twice or more, it is registered only once.) The commands stored in memory can be recalled by entering the [Ctrl] + [H] keys when the [Command] window is active.



```
SC:I1 I0 U D N V C Z   CC:F3 F2 F1 F0
    1  1 0 0 0 0 0 0       0  0  0  0
>
  s 100
  dd
  g
```

Command history
displayed by entering [Ctrl] + [H]

- Simply enter [Ctrl] + [H] to display a command history in popup list form. Double-click a command to repeat, or select a command with the up or down arrow keys and press [Enter]. The command is pasted into the prompt position. It can then be executed by pressing the [Enter] key. If the command history has only one previous command registered, the command is pasted directly into the prompt position without being displayed in a popup list.
- Enter [Ctrl] + [H] after entering any character to display a command history in one of the following ways:
  - If the command history has several commands registered that begin with that character (string), those commands are listed. Then, when another character (string) is entered, one of the recently executed commands among those listed is selected (highlighted) that includes the character (string).
  - If the command history contains only a single command registered that begins with the character (string), the command is pasted directly into the prompt position.
  - If the command history does not contain any commands registered that begin with the character (string), no operation is performed.

  For example, if the command history contains the three commands dd, sy, and s:
  - Enter [Ctrl] + [H] after entering the character 's'. The commands s and sy are listed. Here, the recently executed command s is displayed above the other commands and highlighted.
  - If you follow by entering a 'y', command sy is highlighted.
  - Enter [Ctrl] + [H] after entering the character 'd' to paste the command dd into the prompt position.

Note: The [Command] window cannot be closed.

13.4.3 [Source] Window

The [Source] window displays the program code. The following three display modes are supported:

1. Disassemble display mode

After disassembling the loaded object, the debugger displays the addresses, codes, and mnemonics in it. To open the [Source] window in this mode, select [Source | Disassemble] from the [View] menu. To go to disassemble display mode while in another mode, select [Source | Disassemble] as described above, or click the [Disassemble] button on the [Source] window, or run the u command. When the [Source] window is in this display mode, the word "Disassemble" is displayed on the title bar. This display mode can be selected regardless of the type of object file loaded.

[Disassemble] button

2. Source display mode

In this mode, the debugger displays the corresponding source for an object that includes the current program counter address. However, this mode can be selected only when an absolute object file (.abs) in IEEE-695 format containing debug information for source display purpose is loaded. To open the [Source] window in this mode, select [Source | Source] from the [View] menu. To go to source display mode while in another mode, select [Source | Source] as described above, or click the [Source] button on the [Source] window, or run the sc command. When an absolute object file (.abs) that contains C source debug information is loaded while the [Source] window is open, the [Source] window automatically enters this mode. In this display mode, the source file name is displayed on the title bar.

[Source] button

3. Mix display mode

In this mode, the debugger displays the source and its disassembled contents (address, code, and mnemonic) separately in the upper and lower rows. However, this mode can be selected only when an absolute object file (.abs) in IEEE-695 format containing debug information for source display purpose is loaded. To open the [Source] window in this mode, select [Source | Mix] from the [View] menu. To go to mix display mode while in another mode, select [Source | Mix] as described above, or click the [Mix] button on the [Source] window, or run the m command. When the [Source] window is in this display mode, the word "Mix" is displayed on the title bar.

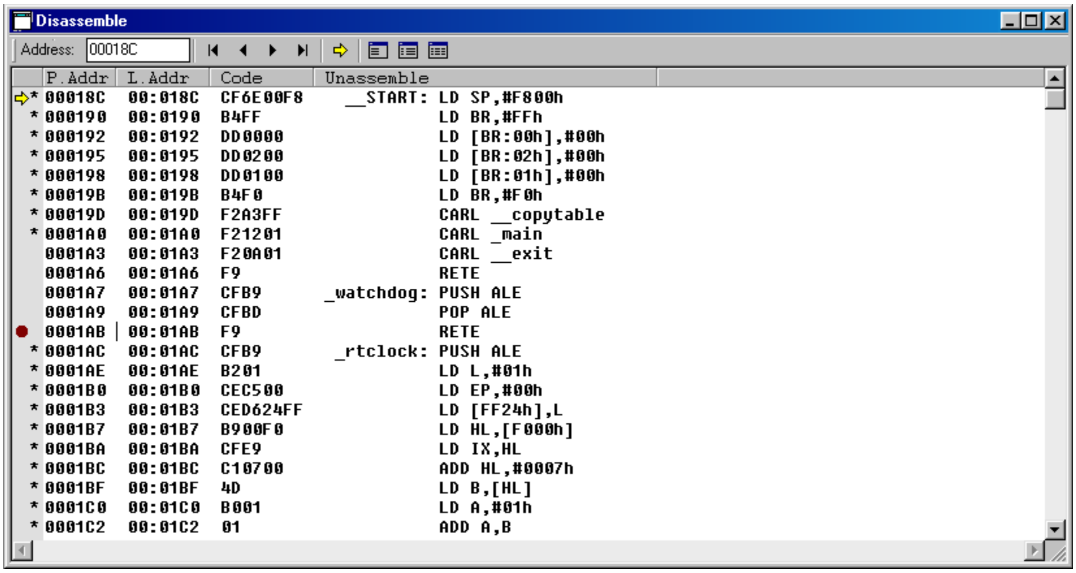
[Mix] button

* **Source display**

The source of any object can be displayed only when an absolute object file in IEEE-695 format that contains debug information for source display purpose is loaded.

Furthermore, because the source file is loaded after locating it from the object file's debug information (relative path information for the source file), if the source file is removed or relocated (i.e., its relative position from the object file has changed), the source is not displayed. In this case, the window in source display mode is left blank, and in mix display mode, the window shows only the disassembled contents.

Disassemble display mode



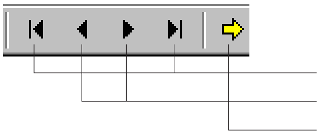
```
Address: 00018C

   P.Addr   L.Addr   Code         Unassemble
➪* 00018C   00:018C  CF6E00F8       __START: LD SP,#F800h
 * 000190   00:0190  B4FF                    LD BR,#FFh
 * 000192   00:0192  DD0000                  LD [BR:00h],#00h
 * 000195   00:0195  DD0200                  LD [BR:02h],#00h
 * 000198   00:0198  DD0100                  LD [BR:01h],#00h
 * 00019B   00:019B  B4F0                    LD BR,#F0h
 * 00019D   00:019D  F2A3FF                  CARL __copytable
 * 0001A0   00:01A0  F21201                  CARL _main
   0001A3   00:01A3  F20A01                  CARL __exit
   0001A6   00:01A6  F9                      RETE
   0001A7   00:01A7  CFB9        _watchdog: PUSH ALE
   0001A9   00:01A9  CFBD                    POP ALE
●  0001AB | 00:01AB  F9                      RETE
 * 0001AC   00:01AC  CFB9         _rtclock: PUSH ALE
 * 0001AE   00:01AE  B201                    LD L,#01h
 * 0001B0   00:01B0  CEC500                  LD EP,#00h
 * 0001B3   00:01B3  CED624FF                LD [FF24h],L
 * 0001B7   00:01B7  B900F0                  LD HL,[F000h]
 * 0001BA   00:01BA  CFE9                    LD IX,HL
 * 0001BC   00:01BC  C10700                  ADD HL,#0007h
 * 0001BF   00:01BF  4D                      LD B,[HL]
 * 0001C0   00:01C0  B001                    LD A,#01h
 * 0001C2   00:01C2  01                      ADD A,B
```

Described below are the functions of the [Source] window in disassemble display mode:

(1) Displaying program code

The window displays the physical/logical addresses, codes, and disassembled contents. Program display location can be changed by the following method as well as scrolling.

- Enter an address in the [Address] text box. Or specify an address using the u command. The program is displayed from the selected address.
-



Displays the beginning or end area of the memory.
Displays one page before or after in the current window size.
Displays the program from the current PC address.

Note: The S1C88 Family processors use variable length mnemonics, so that when the window is scrolled upward, the disassembled contents shown on the window may differ from the actual code.

∗ Updating of display

When a program is loaded and executed (g, gr, s, n, se, or rst command), or the memory contents are changed (de, df, or dm command), the display contents are updated. In this case the [Disassemble] window updates its display contents so that the current PC address can always be displayed.

(2) Displaying the current PC

The current PC (program counter) address is indicated by a yellow arrow at the beginning of the line.

(3) Displaying PC breakpoints

The address line where a breakpoint is set is indicated by a red ● mark at the beginning of the line.

(4) Coverage information

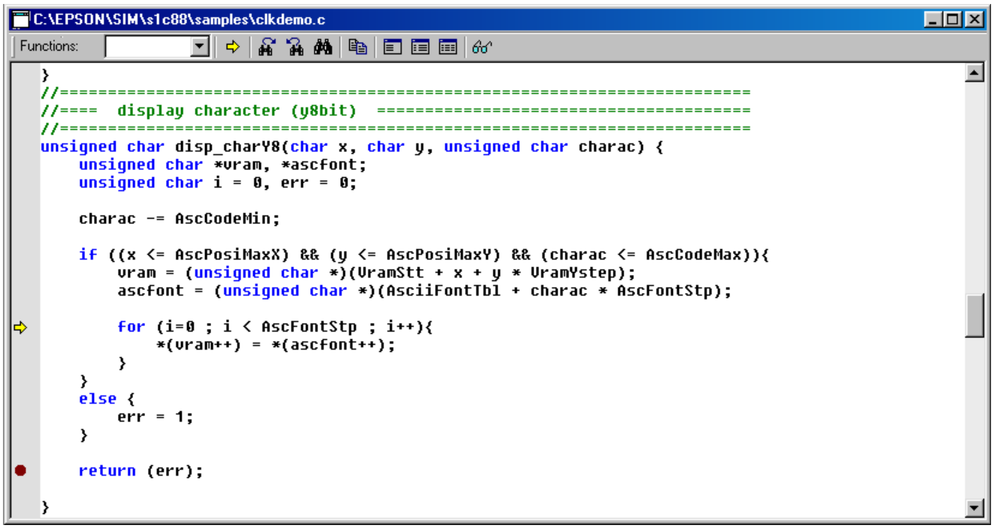
The coverage function places an ∗ at the beginning of the executed address line.

(5) Setting a break at the cursor position

Place the cursor at an address line where a breakpoint is to be set. Then click on the [Break] button. A PC breakpoint will be set at that address. If the same is done at the address line where a PC breakpoint has been set, the breakpoint will be cleared. This function allows setting of two or more breakpoints.

If the [Go to Cursor] button is clicked, the program will execute beginning with the current PC position, and program execution breaks at the line where the cursor is located.

Source display mode



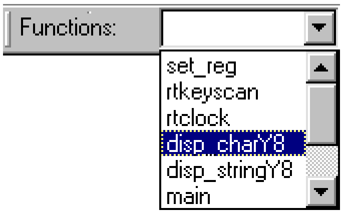
```
}
//========================================================================
//====  display character (y8bit)  =======================================
//========================================================================
unsigned char disp_charY8(char x, char y, unsigned char charac) {
    unsigned char *vram, *ascfont;
    unsigned char i = 0, err = 0;

    charac -= AscCodeMin;

    if ((x <= AscPosiMaxX) && (y <= AscPosiMaxY) && (charac <= AscCodeMax)){
        vram = (unsigned char *)(VramStt + x + y * VramYstep);
        ascfont = (unsigned char *)(AsciiFontTbl + charac * AscFontStp);

        for (i=0 ; i < AscFontStp ; i++){
            *(vram++) = *(ascfont++);
        }
    }
    else {
        err = 1;
    }

    return (err);

}
```

Described below are the functions of the [Source] window in source display mode:

(1) Displaying program code

The window displays the source of the loaded object. The source automatically displayed here includes the address indicated by the current PC (program counter).

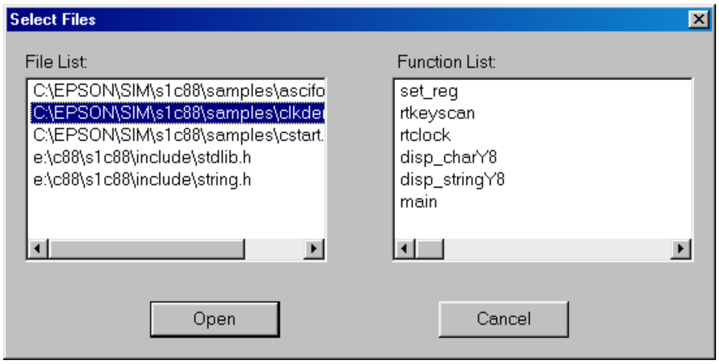
The comment lines, reserved words, and any text other than these two types are displayed in green, blue, and black, respectively. The tab width is set to a length of four characters. The program display position can be changed in the following manner, as well as by scrolling:



- Select a function name from the [Functions] pulldown list. The source is displayed from the beginning of that function.

- Click the [Current PC] button. The source is displayed from the current PC address.

- To display another source file, click the [Source Files] button to bring up the dialog box shown below and to select the desired source file from the list of sources.



File List:
C:\EPSON\SIM\s1c88\samples\ascifo
C:\EPSON\SIM\s1c88\samples\clkde
C:\EPSON\SIM\s1c88\samples\cstart.
e:\c88\s1c88\include\stdlib.h
e:\c88\s1c88\include\string.h

Function List:
set_reg
rtkeyscan
rtclock
disp_charY8
disp_stringY8
main

* Updating of display

When a program is loaded and executed (g, gr, s, n, se, or rst command) and program execution is halted midway, the display contents are updated. In this case, the source that includes the current PC address is displayed in the window. If the corresponding source cannot be found, the [Select Files] dialog box shown above appears, prompting for selection of the source to be displayed.

(2) Displaying the current PC

The source lines that include the address indicated by the current PC (program counter) are marked with a yellow arrow at the beginning of the line.

(3) Displaying PC breakpoints

The source lines that include any address that has been set as a breakpoint are marked with a red ● mark at the beginning of the line.

(4) Setting a break at the cursor position

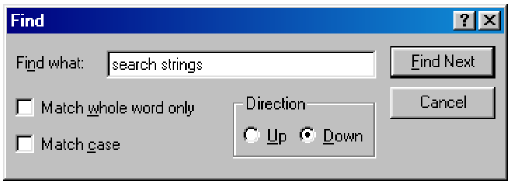
Place the cursor at the source line at which a breakpoint is to be set. Then click the [Break] button. This sets the source line (the start address of the effective object code corresponding to the source) as a breakpoint. (A breakpoint can also be set by double-clicking anywhere in the line.) If the same action is performed at the source line in which a PC breakpoint has been set, the breakpoint is cleared. Multiple breakpoints can be set, one breakpoint per source line. However, no breakpoints can be set in source lines that do not have actual code. Note that due to optimization by the C compiler, no code can be generated for some C statements that would otherwise have code generated. For source lines at which breakpoints cannot be set, change to mix display mode and check.

Click the [Go to Cursor] button. The program starts running from the current PC and breaks at the line at which the cursor is positioned. In this case, the cursor must also be located at the source lines that have the actual code. Clicking the [Go to Cursor] button has no effect unless the source has the actual code.

(5) Searching for a character string

In source display mode, the [Source] window displays the following find buttons, permitting a search for a character string.

Click the [Find] button to display the dialog box shown below, allowing you to specify a search string.



Find

Find what: search strings

☐ Match whole word only

☐ Match case

Direction: ○ Up ● Down

Enter a search string in the [Find what] edit box and click the [Find Next] button. The string search proceeds in the downward direction of the [Source] window (toward the end of the program) from the current cursor position. If an instance of the specified string is found in the [Source] window, it is placed in a selected state.

When the [Find Next] button is clicked again, the next instance of the specified string is sought from that position forward. To search up (toward the beginning of the program), select the [Up] button for [Direction]. To search for instances that completely match the specified string, check the [Match whole word only] check box. Or to discriminate between uppercase and lowercase letters when searching, check the [Match case] check box, before clicking the [Find Next] button.

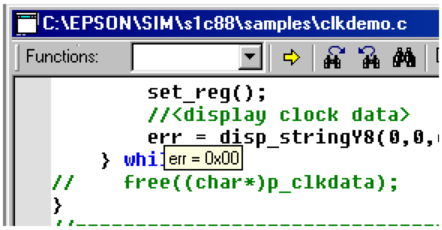
Select a string by dragging the mouse in the [Source] window and clicking the [Find Next] button on the [Source] window. The string search proceeds in the downward direction of the [Source] window (toward the end of the program) from that selected position. If an instance of the string is found, the newly found string is placed in a selected state. When the [Find Next] button is clicked again, the next instance of the string is sought from that position forward. This search is case-insensitive, and instances that do not completely match the string will also be found.

The [Find Previous] button functions in the same way as the [Find Next] button described above, except that string searches proceed up (toward the beginning of the program).

(6) Registering symbols in the [Watch] window

Select a symbol name in the window by dragging with the mouse (displayed in reverse video when selected) and click the [Watch] button. The symbol is registered to the symbol list of the [Watch] window. Once registered this way, the value of that symbol can be verified in the [Watch] window.

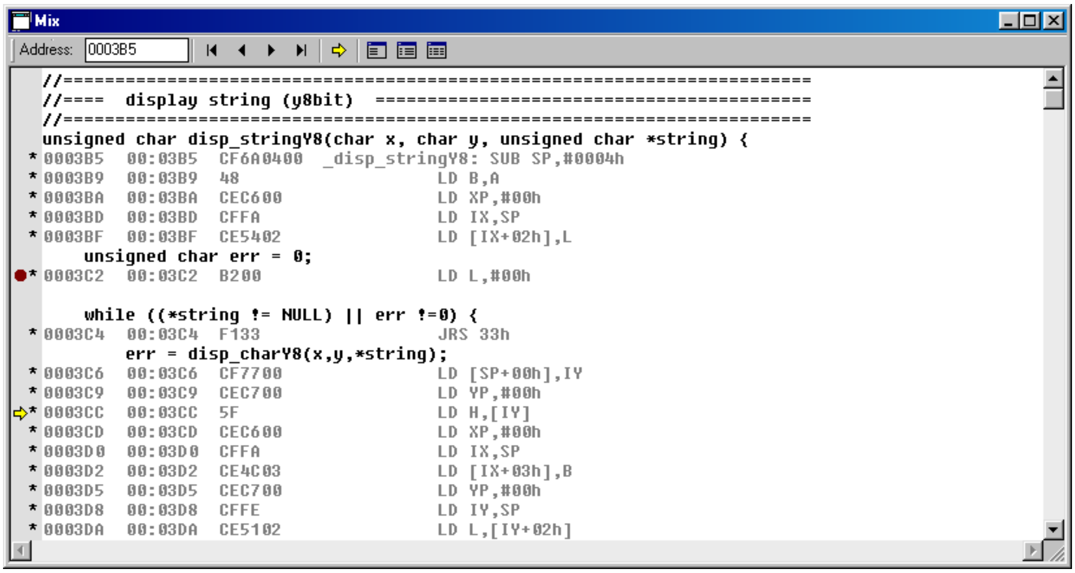
(7) Displaying variable values



```
C:\EPSON\SIM\s1c88\samples\clkdemo.c
Functions:
          set_reg();
          //<display clock data>
          err = disp_stringY8(0,0,
      } whi err = 0x00
//    free((char*)p_clkdata);
}
//-------------------------------
```

Place the mouse cursor at a variable name in the displayed source (need not to click), and the value of that variable (or address for a pointer variable) is displayed. The variable types (signed / unsigned) int, long, and short are displayed in decimal notation, while addresses, structures, and unions are displayed in hexadecimal notation. To display the values of structure members, the member's variable name needs to be selected with the mouse. For array elements, variable names must be selected with the mouse. Out-of-scope variables are not displayed.

Mix display mode



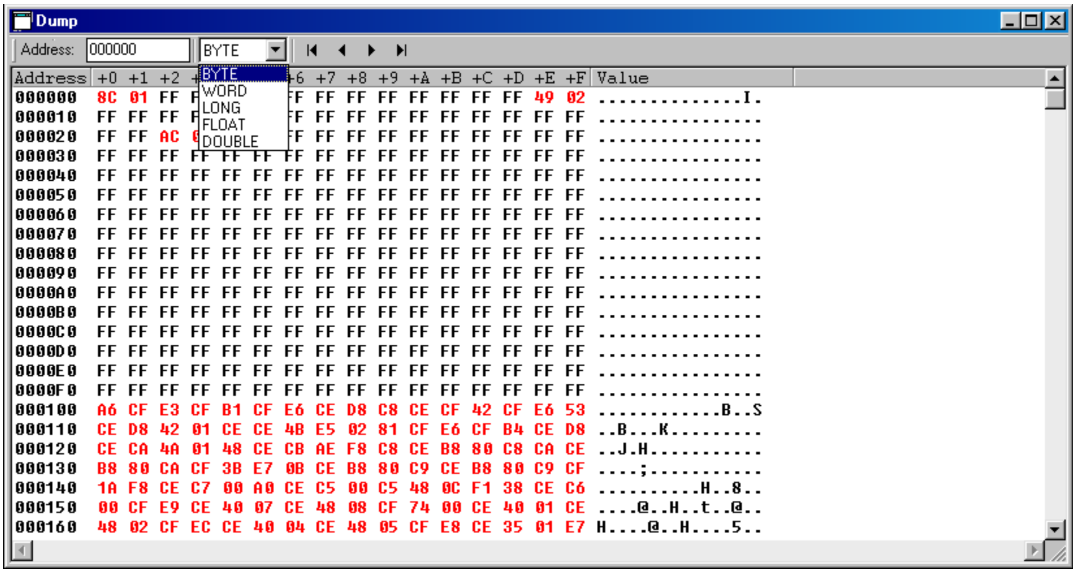
```
Mix
Address: 0003B5

 //======================================================================
 //====   display string (y8bit)  =======================================
 //======================================================================
 unsigned char disp_stringY8(char x, char y, unsigned char *string) {
*0003B5   00:03B5  CF6A0400  _disp_stringY8: SUB SP,#0004h
*0003B9   00:03B9  48                       LD B,A
*0003BA   00:03BA  CEC600                   LD XP,#00h
*0003BD   00:03BD  CFFA                     LD IX,SP
*0003BF   00:03BF  CE5402                   LD [IX+02h],L
     unsigned char err = 0;
*0003C2   00:03C2  B200                     LD L,#00h

     while ((*string != NULL) || err !=0) {
*0003C4   00:03C4  F133                     JRS 33h
         err = disp_charY8(x,y,*string);
*0003C6   00:03C6  CF7700                   LD [SP+00h],IY
*0003C9   00:03C9  CEC700                   LD YP,#00h
*0003CC   00:03CC  5F                       LD H,[IY]
*0003CD   00:03CD  CEC600                   LD XP,#00h
*0003D0   00:03D0  CFFA                     LD IX,SP
*0003D2   00:03D2  CE4C03                   LD [IX+03h],B
*0003D5   00:03D5  CEC700                   LD YP,#00h
*0003D8   00:03D8  CFFE                     LD IY,SP
*0003DA   00:03DA  CE5102                   LD L,[IY+02h]
```

The mix display mode is functionally the same as disassemble display mode. The difference is that each source line and the disassembled contents of the corresponding object code (physical / logical address, object code, and mnemonic) are displayed one for one in the upper and lower rows. However, mix display mode can only be selected when an absolute object file (.abs) in IEEE-695 format containing debug information is loaded.

The displayed source lines cannot be operated on - for example, by setting a break. Various display manipulating and break setting operations can only be performed on the disassembled display contents. For [Source] window functions that can be used in mix display mode, refer to the description of disassemble display mode.

The source lines and the disassembled contents are displayed in black and gray, respectively.

13.4.4 [Dump] Window



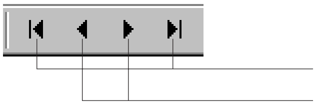
```
Address: 000000   BYTE

Address  +0 +1 +2 +           +6 +7 +8 +9 +A +B +C +D +E +F  Value
000000   8C 01 FF F           F  FF FF FF FF FF FF FF 49 02  ..............I.
000010   FF FF FF F           F  FF FF FF FF FF FF FF FF FF  ................
000020   FF FF AC 0           F  FF FF FF FF FF FF FF FF FF  ................
000030   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ................
000040   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ................
000050   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ................
000060   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ................
000070   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ................
000080   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ................
000090   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ................
0000A0   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ................
0000B0   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ................
0000C0   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ................
0000D0   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ................
0000E0   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ................
0000F0   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ................
000100   A6 CF E3 CF B1 CF E6 CE D8 C8 CE CF 42 CF E6 53  ............B..S
000110   CE D8 42 01 CE CE 4B E5 02 81 CF E6 CF B4 CE D8  ..B...K.........
000120   CE CA 4A 01 48 CE CB AE F8 C8 CE B8 80 C8 CA CE  ..J.H...........
000130   B8 80 CA CF 3B E7 0B CE B8 80 C9 CE B8 80 C9 CF  ....;...........
000140   1A F8 CE C7 00 A0 CE C5 00 C5 48 0C F1 38 CE C6  ..........H..8..
000150   00 CF E9 CE 40 07 CE 48 08 CF 74 00 CE 40 01 CE  ....@..H..t..@..
000160   48 02 CF EC CE 40 04 CE 48 05 CF E8 CE 35 01 E7  H....@..H....5..
```

(1) Displaying data memory contents

The [Dump] window displays the memory dump results in hexadecimal numbers.

Data is displayed in byte units by default. It can be changed to another size using the pull-down box.

Memory display location can be changed by the following method as well as scrolling.

- Enter an address in the [Address] text box. Or specify an address using the dd command.
  Data is displayed from the selected address.
- 

Displays the beginning or end area of the memory.
Displays one page before or after in the current window size.

＊ Updating of display

The display contents of the [Dump] window are updated automatically when memory contents are modified with a command (de, df, or dm command), or by direct modification. After executing the program (g, gr, s, n, se, or rst command), the display contents are also updated. To refresh the [Dump] window manually, execute the dd command or click the vertical scroll bar.

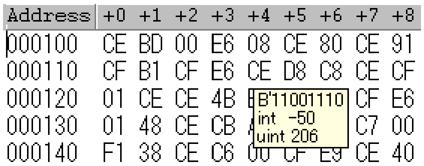
After program execution is completed, the value changed during execution is displayed in red.

(2) Direct modification of data memory contents

The [Dump] window allows direct modification of data memory contents. To modify data on the [Dump] window, place the cursor at the front of the data to be modified or double click the data, and then type a hexadecimal character (0–9, a–f). Data in the address will be modified with the entered number and the cursor will move to the next address. This allows successive modification of a series of addresses.

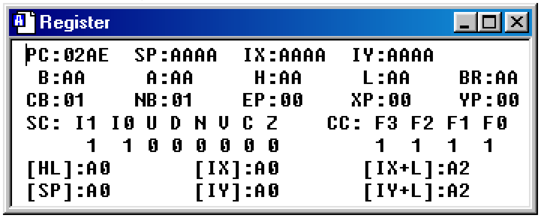
(3) Displaying decimal data

Hover the mouse cursor over data (need not to click) during [BYTE], [WORD], or [LONG] display, and the data is displayed in decimal notation (signed int or unsigned int). For [BYTE], the data is also displayed in bits.



```
Address  +0 +1 +2 +3 +4 +5 +6 +7 +8
000100   CE BD 00 E6 08 CE 80 CE 91
000110   CF B1 CF E6 CE D8 C8 CE CF
000120   01 CE CE 4B E B'11001110 CF E6
000130   01 48 CE CB A int  -50   C7 00
000140   F1 38 CE C6 00 uint 206  CE 40
                    CF E9
```

13.4.5 [Register] Window



```
PC:02AE  SP:AAAA  IX:AAAA  IY:AAAA
 B:AA     A:AA     H:AA     L:AA     BR:AA
CB:01    NB:01    EP:00    XP:00    YP:00
SC: I1 I0 U D N V C Z    CC: F3 F2 F1 F0
     1  1 0 0 0 0 0 0        1  1  1  1
[HL]:A0       [IX]:A0       [IX+L]:A2
[SP]:A0       [IY]:A0       [IY+L]:A2
```

(1) Displaying register contents

The [Register] window displays the contents of the S1C88 CPU registers, condition flags and the memory pointed by the [HL], [SP], [IX], [IY], [IX+L] and [IY+L] registers.

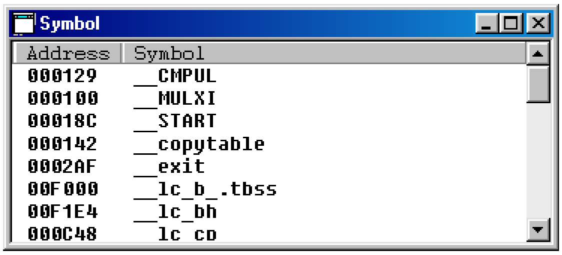
∗ Updating of display

The display is updated when registers are dumped (rd command), when register data is modified (rs command), when the CPU is reset (rst command), or after program execution (g, gr, s, se, or n command) is completed. After program execution is completed, the value changed during execution is displayed in red.

(2) Direct modification of register contents

The [Register] window allows direct modification of register contents. To modify data on the [Register] window, select (highlight) the data to be modified and type a hexadecimal number (0–9, a–f), then press [Enter]. The register data will be modified with the entered number.

13.4.6 [Symbol] Window



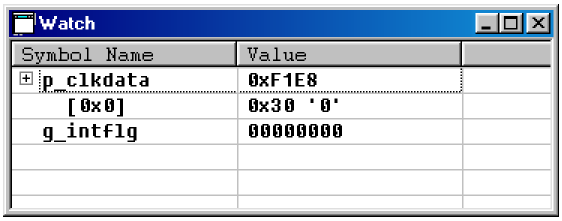
```
Address  Symbol
000129   __CMPUL
000100   __MULXI
00018C   __START
000142   __copytable
0002AF   __exit
00F000   __lc_b_.tbss
00F1E4   __lc_bh
000C48   __lc_cp
```

The [Symbol] window can display the symbol list, if symbol information is loaded.

Symbols are listed in alphabetical order by default. It can be changed to address order using the "sy /a" command.

∗ The symbol file is automatically read when a target program file in the Motorola S2 format is loaded. However, it must be the same name (extension is .sy) and be located in the same directory as the target program file. Note that a symbol file is not read when an IEEE-695 program file is loaded.

13.4.7 [Watch] Window



Symbol Name	Value
⊞ p_clkdata	0xF1E8
[0x0]	0x30 '0'
g_intflg	00000000

The window shows the name and the current value of the symbol registered using the w command or the [Source] window [Watch] button. The value is displayed in the format specified by the w command. If the symbol is an array, structure, or union, a ⊞ icon is displayed. Clicking this icon displays the array, structure, or union members hierarchically.

The registered symbols can also be removed or have their display formats changed (e.g., from hexadecimal to decimal) from a menu displayed by right-clicking the symbol. However, display formats can be changed only for types such as int, char, long, and short, and cannot be changed for addresses. The addresses are always displayed in hexadecimal notation.

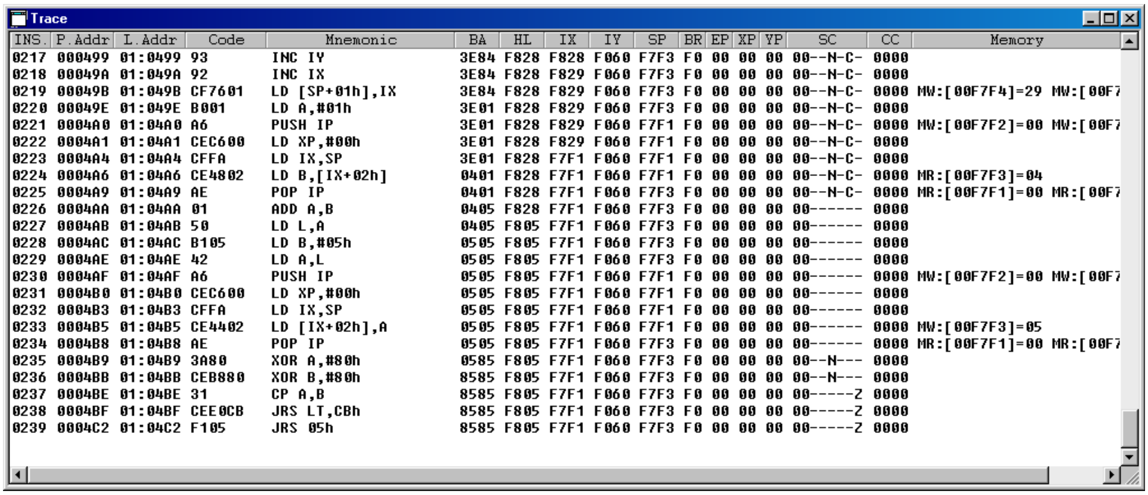
Note that symbol display on this window is possible only when an absolute object file (.abs) in IEEE-695 format containing information on the specified symbol is loaded.

Note: If the -O1 option is specified when compiling, unnecessary symbols may be removed for code optimization, and no symbol information may be generated. Such symbols cannot be registered in the [Watch] window.

∗ Updating of display

The display is updated after program execution (g, gr, s, se, or n command) is completed (default). This condition can be changed so that the display is updated while the program is running using the dialog box that appears by selecting the [Run | Setting...] menu command (see Section 13.8.4, "Executing Program").

13.4.8 [Trace] Window



```
INS. P.Addr L.Addr  Code    Mnemonic            BA   HL   IX   IY   SP   BR EP XP YP SC         CC   Memory
0217 000499 01:0499 93      INC IY              3E84 F828 F828 F060 F7F3 F0 00 00 00 00--N-C-  0000
0218 00049A 01:049A 92      INC IX              3E84 F828 F829 F060 F7F3 F0 00 00 00 00--N-C-  0000
0219 00049B 01:049B CF7601  LD [SP+01h],IX      3E84 F828 F829 F060 F7F3 F0 00 00 00 00--N-C-  0000 MW:[00F7F4]=29 MW:[00F7
0220 00049E 01:049E B001    LD A,#01h           3E01 F828 F829 F060 F7F3 F0 00 00 00 00--N-C-  0000
0221 0004A0 01:04A0 A6      PUSH IP             3E01 F828 F829 F060 F7F1 F0 00 00 00 00--N-C-  0000 MW:[00F7F2]=00 MW:[00F7
0222 0004A1 01:04A1 CEC600  LD XP,#00h          3E01 F828 F829 F060 F7F1 F0 00 00 00 00--N-C-  0000
0223 0004A4 01:04A4 CFFA    LD IX,SP            3E01 F828 F7F1 F060 F7F1 F0 00 00 00 00--N-C-  0000
0224 0004A6 01:04A6 CE4802  LD B,[IX+02h]       0401 F828 F7F1 F060 F7F1 F0 00 00 00 00--N-C-  0000 MR:[00F7F3]=04
0225 0004A9 01:04A9 AE      POP IP              0401 F828 F7F1 F060 F7F3 F0 00 00 00 00--N-C-  0000 MR:[00F7F1]=00 MR:[00F7
0226 0004AA 01:04AA 01      ADD A,B             0405 F828 F7F1 F060 F7F3 F0 00 00 00 00------  0000
0227 0004AB 01:04AB 50      LD L,A              0405 F805 F7F1 F060 F7F3 F0 00 00 00 00------  0000
0228 0004AC 01:04AC B105    LD B,#05h           0505 F805 F7F1 F060 F7F3 F0 00 00 00 00------  0000
0229 0004AE 01:04AE 42      LD A,L              0505 F805 F7F1 F060 F7F3 F0 00 00 00 00------  0000
0230 0004AF 01:04AF A6      PUSH IP             0505 F805 F7F1 F060 F7F1 F0 00 00 00 00------  0000 MW:[00F7F2]=00 MW:[00F7
0231 0004B0 01:04B0 CEC600  LD XP,#00h          0505 F805 F7F1 F060 F7F1 F0 00 00 00 00------  0000
0232 0004B3 01:04B3 CFFA    LD IX,SP            0505 F805 F7F1 F060 F7F1 F0 00 00 00 00------  0000
0233 0004B5 01:04B5 CE4402  LD [IX+02h],A       0505 F805 F7F1 F060 F7F1 F0 00 00 00 00------  0000 MW:[00F7F3]=05
0234 0004B8 01:04B8 AE      POP IP              0505 F805 F7F1 F060 F7F3 F0 00 00 00 00------  0000 MR:[00F7F1]=00 MR:[00F7
0235 0004B9 01:04B9 3A80    XOR A,#80h          0585 F805 F7F1 F060 F7F3 F0 00 00 00 00--N---  0000
0236 0004BB 01:04BB CEB880  XOR B,#80h          8585 F805 F7F1 F060 F7F3 F0 00 00 00 00--N---  0000
0237 0004BE 01:04BE 31      CP A,B              8585 F805 F7F1 F060 F7F3 F0 00 00 00 00-----Z  0000
0238 0004BF 01:04BF CEE0CB  JRS LT,CBh          8585 F805 F7F1 F060 F7F3 F0 00 00 00 00-----Z  0000
0239 0004C2 01:04C2 F105    JRS 05h             8585 F805 F7F1 F060 F7F3 F0 00 00 00 00-----Z  0000
```

After the trace function is turned on by the md command, the debugger samples trace information while the target program is running. The trace data buffer has a capacity for 8192 instructions (overwritten from the beginning if the capacity is exceeded), and its data can be displayed in the [Trace] window.

The following lists the trace contents:

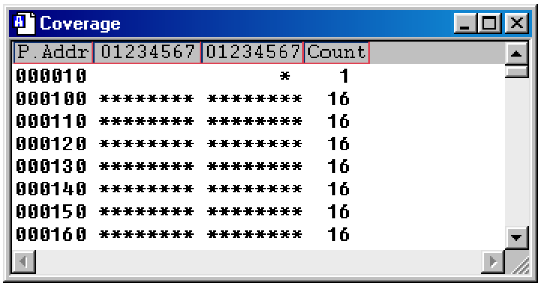
- Instruction number
- Fetched code and disassembled contents
- Register and condition flag contents
- Memory access status (R/W, address, data)

This window also displays the trace data search results by the ts command.

∗ Updating of display

The contents of the [Trace] window are cleared when the target program is executed. After the execution has finished, the [Trace] window displays the contents of the trace buffer.

13.4.9 [Coverage] Window



```
P.Addr 01234567 01234567 Count
000010                *      1
000100 ******** ********    16
000110 ******** ********    16
000120 ******** ********    16
000130 ******** ********    16
000140 ******** ********    16
000150 ******** ********    16
000160 ******** ********    16
```

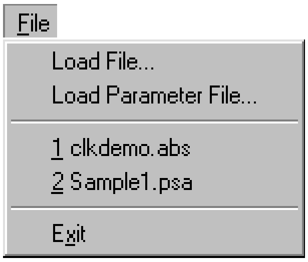
This window shows the coverage information (executed address information) acquired by the ICE. The displayed contents indicate the memory map in 16 bytes per line. The value at the beginning of each line is a physical address (hexadecimal value). Asterisks (∗) in the line indicate the executed addresses within a 16-byte area beginning with the displayed address. The Count values are number of executed addresses in the line.

The [Coverage] window does not update its displayed contents automatically even if a program execution is suspended. To update the display, select [Coverage] from the [Coverage] menu or execute the cv command. To clear the coverage information acquired in the ICE and display contents in the [Coverage] window, select [Coverage Clear] from the [Coverage] menu.

13.5 Menu

This section outlines the menu bar available with the debugger.
The menu bar has nine menus, each including frequently-used commands.

[File] Menu



The file names listed in this menu are recently used files. Selecting one opens the file.

[Load File...]

This button reads an object file in the IEEE-695 format or an internal ROM HEX file in Motorola S2 format into the debugger. It performs the same function when the lf command is executed.

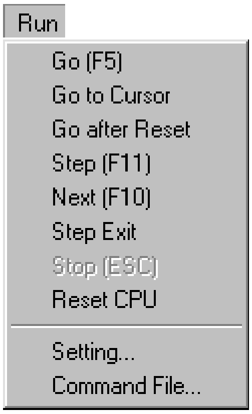
[Load Parameter File...]

This button reads a parameter file into the debugger. It performs the same function when the par command is executed.

[Exit]

This menu item quits the debugger. It performs the same function when the q command is executed.

[Run] Menu



[Go]

This menu item executes the target program from the address indicated by the current PC. The [F5] key can also be used. It performs the same function when the g command is executed.

[Go to Cursor]

This menu item executes the target program from the address indicated by the current PC to the cursor position in the [Source] window (the address of that line). Before this menu item can be selected, the [Source] window must be open and the address line where the program is to break must be clicked.

[Go after Reset]

This button resets the CPU and then executes the target program after fetching the reset vector. It performs the same function when the gr command is executed.

[Step]

This menu item executes one instruction step at the address indicated by the current PC. The [F11] key can also be used. It performs the same function when the s command is executed.

[Next]

This button executes one instruction step at the address indicated by the current PC. If the instruction to be executed is cars, carl, call, or int, it is assumed that a program section until control returns to the next address constitutes one step and all steps of their subroutines are executed. The [F10] key can also be used. It performs the same function when the n command is executed.

[Step Exit]

This button executes the target program from the address indicated by the current PC. If the program starts from inside a subroutine, the program execution will stop when the sequence returns to the parent routine. This button performs the same function when the se command is executed.

[Stop]

This menu item forcibly breaks execution of the target program. The [Esc] key can also be used.

[Reset CPU]

This menu item resets the CPU. It performs the same function when the rst command is executed.

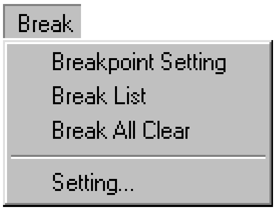
[Setting...]

This menu item displays a dialog box for setting options related to program execution (execution monitor interval, interrupt mode during single stepping, watch update mode, and unit of execution time measurement).

[Command File...]

This menu item reads a command file and executes the debug commands written in that file. It performs the same function when the com or cmw command is executed.

[Break] Menu



[Breakpoint Setting]

This menu item sets or clears PC breakpoints and data break conditions using a dialog box. It performs the same function as executing the bp, bpa, ba and bd command.

[Break List]

This menu item displays the all break conditions that have been set. It performs the same function as executing the bl command.

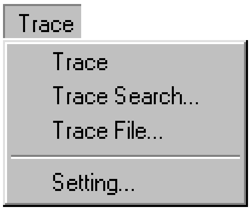
[Break All Clear]

This menu item clears all break conditions. It performs the same function as executing the bac command.

[Setting...]

This menu item displays a dialog box for setting a software break enable area and sequential break mode.

[Trace] Menu



[Trace]

This menu item activates the [Trace] window to displays the trace information sampled in the ICE trace data buffer. It performs the same function as executing the td command.

[Trace Search...]

This menu item searches trace information from the trace data buffer under the condition specified using a dialog box. It performs the same function as executing the ts command.

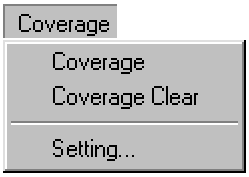
[Trace File...]

This menu item saves the specified range of the trace information displayed in the [Trace] window to a file. It performs the same function as executing the tf command.

[Setting...]

This menu item displays a dialog box for setting the trace mode.

[Coverage] Menu



[Coverage]

This menu item activates the [Coverage] window to displays the coverage information acquired in the ICE. It performs the same function as executing the cv command.

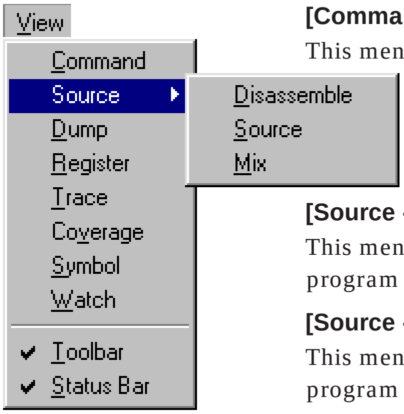
[Coverage Clear]

This menu item clears the coverage information acquired in the ICE and display contents in the [Coverage] window. It performs the same function as executing the cvc command.

[Setting...]

This menu item displays a dialog box for setting coverage options (coverage area and coverage mode).

[View] Menu



[Command]

This menu item activates the [Command] window.

[Source - Disassemble]

This menu item opens or activates the [Source] window and displays the program in the disassemble display mode.

[Source - Source]

This menu item opens or activates the [Source] window and displays the program in the source display mode.

[Source - Mix]

This menu item opens or activates the [Source] window and displays the program in the mix display mode.

[Dump]

This menu item opens or activates the [Dump] window and displays the memory contents from the memory start address.

[Register]

This menu item opens or activates the [Register] window and displays the current values of the registers.

[Trace]

This menu item opens or activates the [Trace] window and displays the trace data sampled in the ICE trace data buffer.

[Coverage]

This menu item opens or activates the [Coverage] window and displays the coverage information acquired in the ICE.

[Symbol]

This menu item opens or activates the [Symbol] window and displays the symbol list if a symbol information has been loaded.

[Watch]

This menu item opens or activates the [Watch] window and displays the contents of the symbol registered.

[Toolbar]

This menu item shows or hides the toolbar.

[Status Bar]

This menu item shows or hides the status bar.

[Option] Menu

[Log...]

This menu item starts or stops logging using a dialog box. It performs the same function as executing the log command.

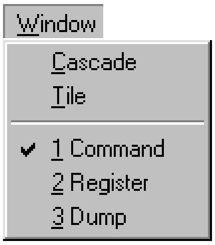
[Record...]

This menu item starts or stops recording of a command execution using a dialog box. It performs the same function as executing the rec command.

[Setting...]

This menu item displays a dialog box for setting system options (emulation clock, firmware clock, self-rewrite check function, and wait time for the cmw command).

[Window] Menu



[Cascade]

This menu item cascades the opened windows.

[Tile]

This menu item tiles the opened windows.

This menu shows the currently opened window names. Selecting one activates the window.

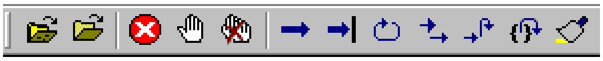
[Help] Menu

[About DB88...]

This menu item displays an About dialog box for the debugger.

13.6 Tool Bar

This section outlines the tool bar available with the debugger.
The tool bar has 12 buttons, each one assigned to a frequently used command.



The specified function is executed when you click on the corresponding button.

[Load File] button

This button reads an absolute object file in IEEE-695 format, a program file in Motorola S2 format, or a function option file into the debugger. It performs the same function when the lf command is executed.

[Load Parameter] button

This button reads a parameter file into the debugger. It performs the same function when the par command is executed.

[Key Break] button

This button forcibly breaks execution of the target program. This function can be used to cause the program to break when the program has fallen into an endless loop.

[Break] button

Use this button to set and clear a breakpoint at the address where the cursor is located in the [Source] window. This function is valid only when the [Source] window is open.

[Break All Clear] button

This button clears all break conditions. It performs the same function as executing the bac command.

[Go] button

This button executes the target program from the address indicated by the current PC. It performs the same function when the g command is executed.

[Go to Cursor] button

This button executes the target program from the address indicated by the current PC to the cursor position in the [Source] window (the address of that line).
Before this button can be selected, the [Source] window must be open and the address line where the program is to break must be clicked.

[Go after Reset] button

This button resets the CPU and then executes the target program after fetching the reset vector. It performs the same function when the gr command is executed.

[Step] button

This button executes one instruction step at the address indicated by the current PC. It performs the same function when the s command is executed.

[Next] button

This button executes one instruction step at the address indicated by the current PC. If the instruction to be executed is cars, carl, call, or int, it is assumed that a program section until control returns to the next address constitutes one step and all steps of their subroutines are executed.
This button performs the same function when the n command is executed.

[Step Exit] button

This button executes the target program from the address indicated by the current PC. If the program starts from inside a subroutine, the program execution will stop when the sequence returns to the parent routine. This button performs the same function when the se command is executed.

[Reset CPU] button

This button resets the CPU. It performs the same function when the rst command is executed.

13.7 Method for Executing Commands

All debug functions can be performed by executing debug commands. This section describes how to execute these commands.

13.7.1 Entering Commands from Keyboard

Select the [Command] window (by clicking somewhere on the [Command] window). When the prompt ">" appears on the last line in this window and a cursor is blinking behind it, the system is ready to accept a command from the keyboard. Input a debug command at the prompt position. The commands are not case-sensitive; they can be input in either uppercase or lowercase.

General command input format

```
>command   [ parameter [ parameter ...  parameter ] ] ↵
```

- A space is required between a command and parameter.
- Space is required between parameters.

Use the arrow keys, [Back Space] key, or [Delete] key to correct erroneous input.
When you press the [Enter] key after entering a command, the system executes that command. (If the command entered is accompanied by guidance, the command is executed when the necessary data is input according to the displayed guidance.)
Input example:

```
>g↵                   (Only a command is input.)
>lf test.abs↵         (A command and parameter are input.)
```

Command input accompanied by guidance

For commands that cannot be executed unless a parameter or the commands that modify the existing data are specified, a guidance mode is entered when only a command is input. In this mode, the system brings up a guidance field, so input a parameter there.
Input example:

```
>cmw↵
File name   ? :test.cmd↵ ← Input data according to the guidance (underlined part).
           :
```

- **Commands requiring parameter input as a precondition**

The cmw command shown in the above example reads a program file into the debugger. Commands like this that require an entered parameter as a precondition are not executed until the parameter is input and the [Enter] key pressed. If a command has multiple parameters to be input, the system brings up the next guidance, so be sure to input all necessary parameters sequentially. If the [Enter] key is pressed without entering a parameter in some guidance session of a command, the system assumes the command is canceled and does not execute it.

- **Commands that replace existing data after confirmation**

The commands that rewrite memory or register contents one by one provide the option of skipping guidance (do not modify the contents), returning to the immediately preceding guidance, or terminating during the input session.

[Enter] key	Skips input.
[^] key	Returns to the immediately preceding guidance.
[q] key	Terminates the input session.

Input example:

```
>de↵                                ← Command to modify data memory.
Data enter address ? :00ff00↵       ← Inputs the start address.
00FF00 A:1↵                         ← Modifies address 00ff00H to 1.
00FF01 A:^↵                         ← Returns to the immediately preceding address.
00FF00 1:0↵                         ← Inputs address 00ff00H back again.
00FF01 A:↵
00FF02 A:↵
00FF01 A:q↵                         ← Terminates the input session.
>
```

Numeric data format of parameter

For numeric values to be accepted as a parameter, they must be input in hexadecimal numbers for almost all commands. However, some parameters accept decimal or binary numbers.
The following characters are valid for specifying numeric data:

Hexadecimal:	0–9, a–f, A–F, ∗
Decimal:	0–9
Binary:	0, 1, ∗

("∗" is used to mask bits when specifying a data pattern.)

Specification with a symbol

For address specifications, the symbols can also be used when an IEEE-695 absolute object file (.abs) or a symbol file (.sy) is loaded.
Input example:

```
>u Main↵        ← Displays the program from the label Main
```

∗ The symbol file (.sy) is automatically loaded simultaneously with the target program in the Motorola S2 format. However, it must be the same name (extension is .sy) and be located in the same directory as the target program file. When an IEEE-695 program file is specified, the debugger does not load a symbol file.

Notes: • *If the specified symbol is not found, db88 handles the specified string as a hexadecimal (e.g., ABC). However, if the string includes other than the specified hexadecimal characters, an error is assumed.*

• *If the -O1 option is specified when compiling the C source, some symbols written in the source may not actually be used for reasons of code optimization. In such cases, debug information for that symbol is not output to the .abs file, whether or not the -g option is specified.*

Example:

```
int x,y,xy;
x = GLOBAL_X * 100;
y = GLOBAL_Y * 100;
xy = x * y;
```

In this example, because variable xy become nonexistent due to optimization, the contents of xy cannot be referenced when debugging.
If after evaluating the executable file created by specifying the -O0 option (optimization OFF), it is recreated by specifying the -O1 option (optimization ON), program behavior cannot be guaranteed. Be sure to reverify the executable file whenever it is recreated in this way.

Successive execution using the [Enter] key

The commands listed below can be executed successively by using only the [Enter] key after executing once. Successive execution here means repeating the previous operation or continuous display of the previous contents.

Execution commands:	g, s, n, se, com
Display commands:	u, dd, td

The successive execution function is terminated when some other command is executed.

13.7.2 Executing from Menu or Tool Bar

The menu and tool bar are assigned frequently-used commands as described in Sections 13.5 and 13.6. A command can be executed simply by selecting desired menu command or clicking on the tool bar button. Table 13.7.2.1 lists the commands assigned to the menu and tool bar.

Table 13.7.2.1 Commands that can be specified from menu or tool bar

Command	Function	Menu	Button
lf	Load program file	[File | Load File...]	📂
par	Load parameter file	[File | Load Parameter File...]	📂
g	Execute program successively	[Run | Go]	➡
–	Execute program to cursor position successively	[Run | Go to Cursor]	➡|
gr	Reset CPU and execute program successively	[Run | Go after Reset]	⟲
s	Single step execution	[Run | Step]	↳
n	Step execution with skip subroutine	[Run | Next]	↷
se	Exit from subroutine	[Run | Step Exit]	{}→
com	Load and execute command file	[Run | Command File...]	–
cmw	Load and execute command file with wait	[Run | Command File...]	–
rst	Reset CPU	[Run | Reset CPU]	🔌
bp, bpa, bpr, bc, bpc	Set/clear software breakpoint	[Break | Breakpoint Setting]	✋
bas	Set sequential break mode	[Break | Setting]	–
ba, bar	Set/clear hardware breakpoint	[Break | Breakpoint Setting]	–
bd, bdc	Set/clear data break conditions	[Break | Breakpoint Setting]	–
bl	Break list	[Break | Break List]	–
bac	Clear all break conditions	[Break | Break All Clear]	✋
td	Display trace information	[View | Trace], [Trace | Trace]	–
ts	Search trace information	[Trace | Trace Search...]	–
tf	Save trace information to file	[Trace | Trace File...]	–
cv	Display coverage information	[Coverage | Coverage]	–
cvc	Clear coverage information	[Coverage | Coverage Clear]	–
u	Disassemble display	[View | Source | Disassemble]	▤ *
sc	Source display	[View | Source | Source]	▤ *
m	Mix display	[View | Source | Mix]	▤ *
dd	Dump memory	[View | Dump]	–
rd	Display register values	[View | Register]	–
sy	Display symbol list	[View | Symbol]	–
w	Display symbol information	[View | Watch]	–
	Register symbols	–	👓 *
log	Turn log output on or off	[Option | Log...]	–
rec	Record commands to a command file	[Option | Record...]	–

∗ Located in the [Source] window

13.7.3 Executing from a Command File

Another method for executing commands is to use a command file that contains descriptions of a series of debug commands. By reading a command file into the debugger the commands written in it can be executed.

Creating a command file

Create a command file as a text file using an editor.
Although there are no specific restrictions on the extension of a file name, Seiko Epson recommends using ".cmd".
Command files can also be created using the rec command. The rec command creates a command file and saves the executed commands to the file.

Example of a command file

The example below shows a command group that loads a program file, sets a breakpoint and then executes the program.

Example: File name = start.cmd

```
lf test.abs
bp 0004d7
g
```

A command file to write the commands that come with a guidance mode can be executed. In this case, be sure to break the line for each guidance input item as a command is written.

Reading in and executing a command file

The debugger has the com and cmw commands available that can be used to execute a command file.
The com command reads in a specified file and executes the commands in that file sequentially in the order they are written.
The cmw command performs the same function as the com command except that each command is executed at intervals specified by the md command (1 to 256 seconds).

Example:

```
com start.cmd

cmw test.cmd
```

The commands written in the command file are displayed in the [Command] window.

Restrictions

Another command file can be read from within a command file. However, nesting of these command files is limited to a maximum of five levels. An error is assumed and the subsequent execution is halted when the com or cmw command at the sixth level is encountered.

13.7.4 Log File

The executed commands and the execution results can be saved to a file in text format that is called a "log file". This file allows verification of the debug procedures and contents.
The contents displayed in the [Command] window are saved to this file.

Command example

```
>log tst.log
```

After the debugger is set to the log mode by the log command (after it starts outputting to a log file), the log command toggles (output turned on in log mode ↔ output turned off in normal mode). Therefore, you can output only the portions needed can be output to the log file.

Display of [Command] window in log mode

The contents displayed in the [Command] window during log mode differ from those appearing in normal mode.

(1) When executing a command when each window is open
(When the window that displays the command execution result is opened)
Normal mode: The contents of the relevant display window are updated. The execution results are not displayed in the [Command] window.
Log mode: The same contents as those displayed in the relevant window are also displayed in the [Command] window. However, changes made to the relevant window by scrolling or opening it are not reflected in the [Command] window.

(2) When executing a command while each window is closed
When the relevant display window is closed, the execution results are always displayed in the [Command] window regardless of whether operation is in log mode or normal mode.

13.8 Debug Functions

This section outlines the debug features of the debugger, classified by function.

13.8.1 Loading Files

Table 13.8.1.1 lists the files read by the debugger and the load commands.

Table 13.8.1.1 Files and load commands

File	Type	Generation tool	Command	Menu	Button
1. Parameter file	.par	–	par	[File | Load Parameter File...]	📂
2. IEEE-695 absolute object file	.abs	lc88	lf	[File | Load File...]	📂
3. Motorola S2 program file	.psa	fil88xxx	lf	[File | Load File...]	📂
4. Function option file	.fsa	fog88xxx or winfog	lf	[File | Load File...]	📂
5. Symbol file	.sy	sy88, sym88	–	–	–
6. Command file	.cmd	–	com/cmw	[Run | Command File...]	–
7. FPGA data file	.mot	–	xfwr ;S	–	–
	.mcs	–	xfwr ;H	–	–

Loading a parameter file resets the debugger. The memory mapping information set by the parameter file can be displayed using the ma command. Refer to Chapter 12, "88xxx.par File", for more information on the parameter file.

The lf command loads an IEEE-695 absolute object file (.abs), a Motorola S2 program file (.psa) or a function option HEX file (.fsa). The debugger distinguishes these files with the specified extension. It is necessary to load an IEEE-695 absolute object file that contains debugging information to perform source level debugging.

The symbol file is required to specify addresses using the symbols defined in the source when debugging a Motorola S2 program file. Debugging can be done even if this file is not loaded. The symbol file is loaded simultaneously with the program file by the lf command. However, it must be the same name (extension is .sy) and be located in the same directory as the program file. When the symbol file is loaded, the contents of the file can be displayed in the [Symbol] window or the [Command] window using the sy command.

When an IEEE-695 absolute object file that contains symbol information is loaded, the debugger does not read the symbol file as the object file allows symbolic debugging.

Refer to Section 13.7.3, for the command file.

A FPGA data file is used to program the FPGA on the peripheral board (S5U1C88000P) for an S1C88 Family model. When this data is written to the FPGA once, rewriting is not necessary until the development for the model has been completed.

13.8.2 Source Display and Symbolic Debugging Function

The debugger allows program debugging while displaying the C source statements. Address specification using a symbol name is also possible.

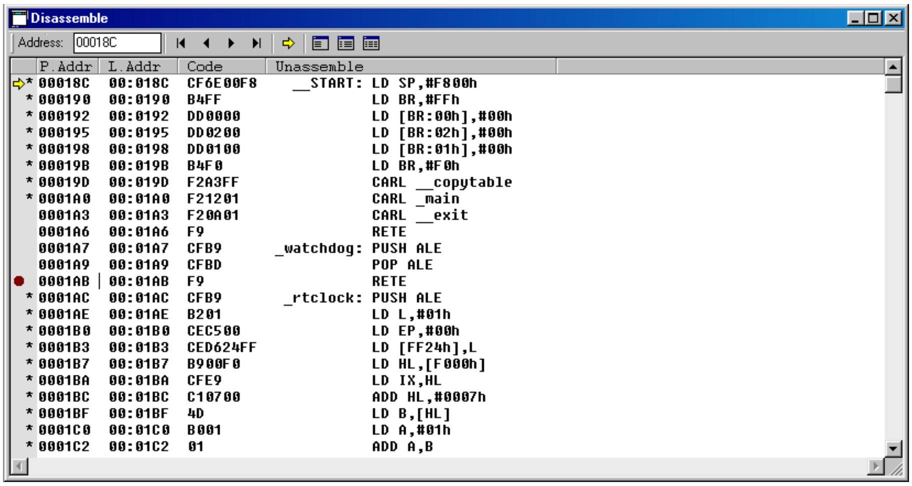
Displaying program code

The [Source] window displays the program in the specified display mode. The display mode can be selected from among the three modes: Disassemble display mode, Source display mode and Mix display mode.

Table 13.8.2.1 Commands/menu items/tool bar buttons to switch display mode

Function	Command	Menu	Button
Disassemble display mode	u	[View | Source | Disassemble]	▤
Source display mode	sc	[View | Source | Source]	▤
Mix display mode	m	[View | Source | Mix]	▦

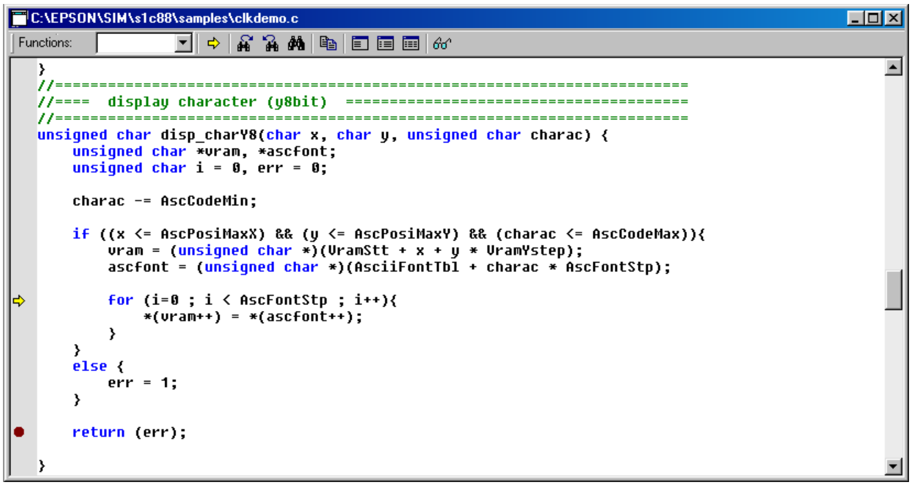
(1) Disassemble display mode



```
   P.Addr  L.Addr  Code         Unassemble
⇨* 00018C  00:018C CF6E00F8       __START: LD SP,#F800h
 * 000190  00:0190 B4FF                    LD BR,#FFh
 * 000192  00:0192 DD0000                  LD [BR:00h],#00h
 * 000195  00:0195 DD0200                  LD [BR:02h],#00h
 * 000198  00:0198 DD0100                  LD [BR:01h],#00h
 * 00019B  00:019B B4F0                    LD BR,#F0h
 * 00019D  00:019D F2A3FF                  CARL __copytable
 * 0001A0  00:01A0 F21201                  CARL _main
   0001A3  00:01A3 F20A01                  CARL __exit
   0001A6  00:01A6 F9                      RETE
   0001A7  00:01A7 CFB9         _watchdog: PUSH ALE
   0001A9  00:01A9 CFBD                    POP ALE
●  0001AB  00:01AB F9                      RETE
 * 0001AC  00:01AC CFB9          _rtclock: PUSH ALE
 * 0001AE  00:01AE B201                    LD L,#01h
 * 0001B0  00:01B0 CEC500                  LD EP,#00h
 * 0001B3  00:01B3 CED624FF                LD [FF24h],L
 * 0001B7  00:01B7 B900F0                  LD HL,[F000h]
 * 0001BA  00:01BA CFE9                    LD IX,HL
 * 0001BC  00:01BC C10700                  ADD HL,#0007h
 * 0001BF  00:01BF 4D                      LD B,[HL]
 * 0001C0  00:01C0 B001                    LD A,#01h
 * 0001C2  00:01C2 01                      ADD A,B
```

In this mode, the debugger displays the program codes after disassembling into mnemonics.

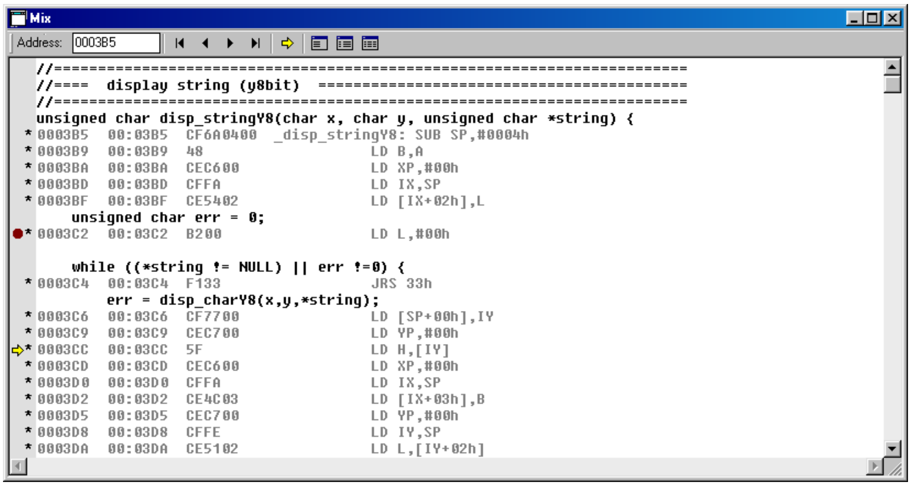
(2) Source display mode



```
  }
  //========================================================================
  //====  display character (y8bit)  =======================================
  //========================================================================
  unsigned char disp_charY8(char x, char y, unsigned char charac) {
      unsigned char *vram, *ascfont;
      unsigned char i = 0, err = 0;

      charac -= AscCodeMin;

      if ((x <= AscPosiMaxX) && (y <= AscPosiMaxY) && (charac <= AscCodeMax)){
          vram = (unsigned char *)(VramStt + x + y * VramYstep);
          ascfont = (unsigned char *)(AsciiFontTbl + charac * AscFontStp);

⇨         for (i=0 ; i < AscFontStp ; i++){
              *(vram++) = *(ascfont++);
          }
      }
      else {
          err = 1;
      }

●     return (err);

  }
```

In this mode, the source that contains the code at the current PC address is displayed. This mode is available only when an IEEE-695 absolute object file that contains source debugging information has been loaded.

(3) Mix display mode



```
   //========================================================================
   //====  display string (y8bit)  ==========================================
   //========================================================================
   unsigned char disp_stringY8(char x, char y, unsigned char *string) {
 * 0003B5   00:03B5  CF6A0400  _disp_stringY8: SUB SP,#0004h
 * 0003B9   00:03B9  48                    LD B,A
 * 0003BA   00:03BA  CEC600                LD XP,#00h
 * 0003BD   00:03BD  CFFA                  LD IX,SP
 * 0003BF   00:03BF  CE5402                LD [IX+02h],L
       unsigned char err = 0;
●* 0003C2   00:03C2  B200                  LD L,#00h

       while ((*string != NULL) || err !=0) {
 * 0003C4   00:03C4  F133                  JRS 33h
           err = disp_charY8(x,y,*string);
 * 0003C6   00:03C6  CF7700                LD [SP+00h],IY
 * 0003C9   00:03C9  CEC700                LD YP,#00h
⇨* 0003CC   00:03CC  5F                    LD H,[IY]
 * 0003CD   00:03CD  CEC600                LD XP,#00h
 * 0003D0   00:03D0  CFFA                  LD IX,SP
 * 0003D2   00:03D2  CE4C03                LD [IX+03h],B
 * 0003D5   00:03D5  CEC700                LD YP,#00h
 * 0003D8   00:03D8  CFFE                  LD IY,SP
 * 0003DA   00:03DA  CE5102                LD L,[IY+02h]
```

This mode displays both sources and the disassembled codes of the corresponding object codes. This mode is available only when an IEEE-695 absolute object file that contains source debugging information has been loaded.

Refer to Section 13.4.3, "[Source] Window" for display contents and operation on the window.

Symbol reference

When debugging a program after loading an object file (.abs) in the IEEE-695 format, the symbols defined in the source file can be used to specify an address. This feature can be used when entering a command having <address> in its parameter from the [Command] window or a dialog box. However, the object file loaded must contain symbol information.

To perform symbolic debugging after loading a program file (.psa) in the Motorola S2 format, it is necessary to prepare a symbol file with the same name as the program file in the same directory. The symbol file is loaded simultaneously with the program file.

The symbols used in the program and the defined addresses can be displayed in the [Command] window or the [Symbol] window.

Table 13.8.2.2 Symbol list display command/menu item

Function	Command	Menu	Button
Displaying symbol list	sy	[View | Symbol]	–

13.8.3 Displaying/Modifying Memory and Register Data

The debugger has functions to operate on the memory and registers. Available memory area is set to the debugger according to the map information that is given in a parameter file.

Memory operation

The following operations can be performed on the memory areas (ROM, RAM, display memory, I/O memory):

Table 13.8.3.1 Memory operation commands/menu item

Function	Command	Menu	Button
Dumping memory data	dd	[View | Dump]	–
Entering/modifying memory data	de	–	–
Rewriting specified area	df	–	–
Coping specified area	dm	–	–
Searching data	ds	–	–

(1) Dumping memory

The memory contents are displayed in a specified size (Byte, Word, Long, Float, Double) hexadecimal dump format. If the [Dump] window is opened, the contents of the [Dump] window are updated; if not, the contents of the data memory are displayed in the [Command] window.

(2) Entering/modifying data

Data at a specified address is rewritten by entering hexadecimal data. Data can be directly modified on the [Dump] window.

(3) Rewriting specified area

An entire specified area is rewritten with specified data.

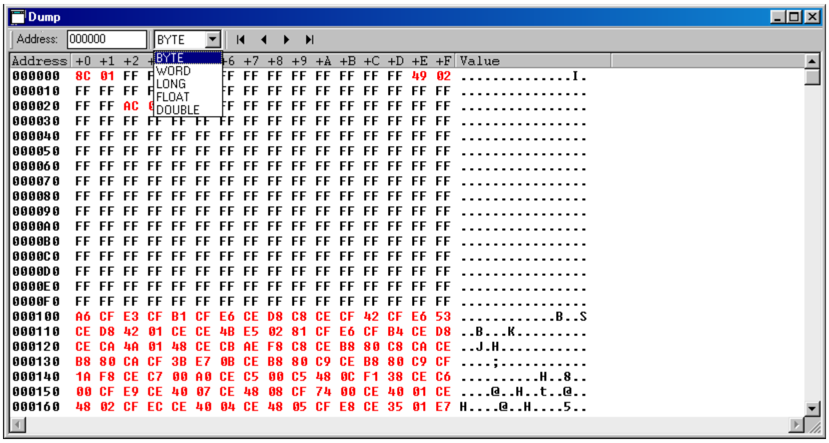
(4) Copying specified area

The content of a specified area is copied to another area.

(5) Searching data

An specified data can be searched within a specified area. The [Command] window displays the results up to 256 found data. The [Dump] window shows found data within the current displayed area in green.

See Section 13.4.4, "[Dump] Window", for display contents and operation on the [Dump] window.



Address: 000000 BYTE

```
Address +0 +1 +2 +                +6 +7 +8 +9 +A +B +C +D +E +F Value
000000  8C 01 FF F               F FF FF FF FF FF FF FF 49 02 ..............I.
000010  FF FF FF F               F FF FF FF FF FF FF FF FF FF ................
000020  FF FF AC 0               F FF FF FF FF FF FF FF FF FF ................
000030  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
000040  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
000050  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
000060  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
000070  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
000080  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
000090  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
0000A0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
0000B0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
0000C0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
0000D0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
0000E0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
0000F0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
000100  A6 CF E3 CF B1 CF E6 CE D8 C8 CE CF 42 CF E6 53 ............B..S
000110  CE D8 42 01 CE CE 4B E5 02 81 CF E6 CF B4 CE D8 ..B...K.........
000120  CE CA 4A 01 48 CE CB AE F8 C8 CE B8 80 C8 CA CE ..J.H...........
000130  B8 80 CA CF 3B E7 0B CE B8 80 C9 CE B8 80 C9 CF ....;...........
000140  1A F8 CE C7 00 A0 CE C5 00 C5 48 0C F1 38 CE C6 ..........H..8..
000150  00 CF E9 CE 40 07 CE 48 08 CF 74 00 CE 40 01 CE ....@..H..t..@..
000160  48 02 CF EC CE 40 04 CE 48 05 CF E8 CE 35 01 E7 H....@..H....5..
```

Operating registers

The following operations can be performed on registers:

Table 13.8.3.2 Register operation commands/menu item

Function	Command	Menu	Button
Displaying register values	rd	[View | Register]	–
Modifying register value	rs	–	–

(1) Displaying registers

Register contents and the contents of the memory specified in register indirect addressing can be displayed in the [Register] or [Command] window.

Registers: PC, SP, IX, IY, A, B, H, L, BR, CB, NB, EP, XP, YP, SC (I1, I0, U, D, N, V, C, Z)
and CC (F3, F2, F1, F0)

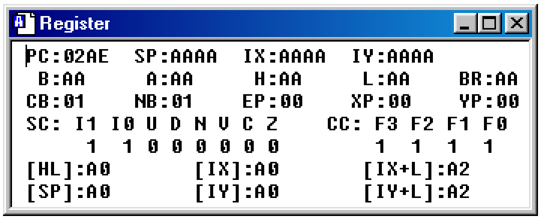
Memory: [HL], [SP], [IX], [IY], [IX+L], [IY+L]

(2) Modifying register values

The contents of the above registers can be set to any desired value.

The register values can be directly modified on the [Register] window.

See Section 13.4.5, "[Register] Window", for display contents and operation on the [Register] window.



```
PC:02AE  SP:AAAA  IX:AAAA  IY:AAAA
 B:AA     A:AA     H:AA     L:AA    BR:AA
CB:01    NB:01    EP:00    XP:00    YP:00
SC: I1 I0 U D N V C Z    CC: F3 F2 F1 F0
     1  1 0 0 0 0 0 0         1  1  1  1
[HL]:A0       [IX]:A0       [IX+L]:A2
[SP]:A0       [IY]:A0       [IY+L]:A2
```

13.8.4 Executing Program

The debugger can execute the target program successively or execute instructions one step at a time (single-stepping).

Successive execution

(1) Types of successive execution

There are three types of successive execution available:

- Successive execution from the current PC
- Successive execution from the current PC to the cursor position in the [Source] window
- Successive execution after resetting the CPU

Table 13.8.4.1 Commands/menu items/tool bar buttons for successive execution

Function	Command	Menu	Button
Successive execution from current PC	g	[Run | Go]	➡
Successive execution from current PC to cursor position	–	[Run | Go to Cursor]	➡|
Successive execution after resetting CPU	gr	[Run | Go after Reset]	⟲

(2) Stopping successive execution

Temporary break addresses can be specified in the [Source] window.

If the cursor is placed on an address line in the [Source] window and the [Go to Cursor] button clicked, the program starts executing from the current PC address and breaks immediately before executing the instruction at the address the cursor is placed.

Note that when displaying C source in source display mode, the cursor must be located at one of the source lines expanded into effective source code. If the cursor is located at any source line, such as a comment line or declaration statement that is not compiled into object code, the program is not executed, even if you click the [Go to Cursor] button. (Refer to the description of the PC break function.)

Except being stopped by this temporary break, the program continues execution until it is stopped by one of the following causes:

- Break conditions set by a break set up command are met.
- A break signal is input to the ICE BRKIN pin.
- The [Key Break] button is clicked, the [Run | Stop] menu command is selected or the [Esc] key is pressed.
- A program execution error is detected.

⊗ *[Key Break] button* ∗ When the program does not stop, use this button to forcibly stop it.

Note: If program execution is halted in C source display mode, the debugger displays the source for an object that includes the halted address. However, if no sources exist at the halted address, a [Source Files] dialog box is displayed, prompting for selection of a source file.

(3) Display during successive executions

The display is updated as below due to a successive execution.

When program execution is halted, the [Command] window displays the number of executed cycles and execution time.

```
Example: >g
         BUS CYCLE : 428649                          ... Number of bus cycles
         Mode L    : 001min 002s  543ms 468us ... Execution time (1 µs units by default)
```

The [Source], [Register] and [Dump] windows do not change their display contents while the program is executing and updates after the program execution is halted. If the [Register] window is closed, its contents are displayed in the [Command] window. The [Trace] window clears its display contents when the program execution is started and re-displays the latest trace data after the program execution is halted. The [Watch] window is updated after the program execution is halted by default. It can be changed so that the window is updated in specified cycles using the dialog that appears by using the [Run | Setting...] menu command.

The [Symbol] and [Coverage] windows do not change their display contents due to successive executions.

Single-stepping

(1) Types of single-stepping

There are three types of single-stepping available:

- **Single-stepping C statements or instructions (STEP)**
  In C source display mode, the program is single-stepped, one C source line at a time. In disassemble display or mix display mode, the program is single-stepped, one instruction at a time.
- **Single-stepping other than functions or subroutines (NEXT)**
  In C source display mode, function calls in the program currently being executed are skipped by handling each function call from entry until the return simply as a single step. Other program parts are single-stepped in the same way as for STEP.
  In disassemble display or mix display mode, the cars, carl, call, and int instructions till returned to the next step by a return instruction are executed as a single step. Other instructions are singled-stepped in the same way as for STEP.
- **Terminating at a function or subroutine (STEP EXIT)**
  In C source display mode, the program is successively executed from the current function until it returns to the higher-level function, and is halted after returning. Do not run this single-stepping mode in the main function.
  In disassemble display or mix display mode, the program is successively executed from the current subroutine until it is returned to the higher-level subroutine by a return instruction, and is halted after returning. At the highest level, the program is executed in the same way as when run by the g command. If a lower-level subroutine is called, and returned from it, the program execution is not halted.

In either case, the program starts executing from the current PC.

Table 13.8.4.2 Commands/menu items/tool bar buttons for single-stepping

Function	Command	Menu	Button
Stepping	s	[Run | Step]	
Stepping except functions/subroutines	n	[Run | Next]	
Exit from function/subroutine	se	[Run | Step Exit]	

When executing s or n by command input, the number of steps to be executed can be specified, up to 65,535 steps. When using menu commands or tool bar buttons, the program is executed one step at a time.

In the following cases, single-stepping is terminated before a specified number of steps is executed:
- The [Key Break] button is clicked, the [Run | Stop] menu command is selected or the [Esc] key is pressed.
- A program execution error is detected.

Single-stepping is not suspended by breaks set by the user such as a PC break or data break.

[Key Break] button   * When the program does not stop, use this button to forcibly stop it.

(2) Display during single-stepping

In the initial debugger settings, the display is updated as follows:
When the [Source], [Register], [Dump], [Trace], or [Watch] window is open, the display contents are updated after the last step has been executed. If the [Register] window is closed, its contents are displayed in the [Command] window.
The [Symbol] and [Coverage] windows do not change their display contents due to single-stepping.

(3) Interrupts during single-stepping

The CPU is placed in a standby mode when the halt or slp instruction is executed. An interrupt is required to cancel this mode.

The debugger has a mode to enable or disable an external interrupt for use in single-step operation.

Table 13.8.4.3 External interrupt modes

	Enable mode	Disable mode
External interrupt	Interrupt is processed.	Interrupt is not processed.
halt and slp instructions	Executed as the halt instruction. Processing is continued by an external interrupt or clicking on the [Key Break] button.	The halt and slp instructions are replaced with a nop instruction as the instruction is executed.

In the initial settings, the debugger is set to the interrupt disable mode. The interrupt enable mode can also be set in the dialog box displayed by selecting [Setting...] from the [Run] menu.

(4) Precautions to be observed when single-stepping C sources

When single-stepping a program in C source display mode, the program is basically executed one source line at a time. However, source lines that do not have the corresponding object code, or lines without user sources (e.g., functions automatically generated by inline assembler or compiler) are skipped until the next line is reached that has effective object code. Accordingly, the number of steps executed varies depending on how C statements are written.

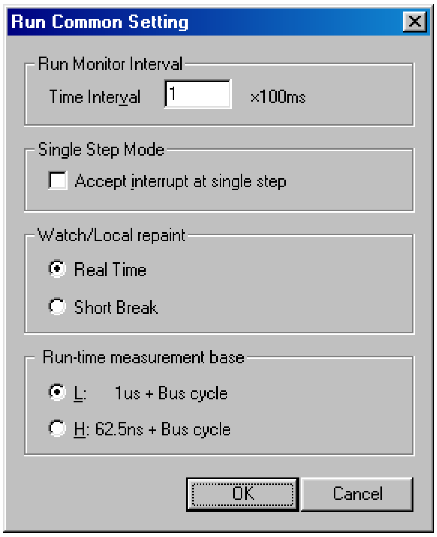
Example: `for(x=0; x<10; x++) a[x]=x;` ... Executed in one step.

```
for(x=0; x<10; x++)
    a[x]=x;
```

... 20 steps need to be executed before exiting the for statement.

Execution options

Four options are available for program execution. To select one of these options, use the dialog box that appears when [Setting...] is selected from the [Run] menu.



Run Common Setting

Run Monitor Interval
Time Interval: 1 ×100ms

Single Step Mode
□ Accept interrupt at single step

Watch/Local repaint
● Real Time
○ Short Break

Run-time measurement base
● L: 1us + Bus cycle
○ H: 62.5ns + Bus cycle

Run Monitor Interval

Set the display update interval in 100 ms increments when selecting "short break mode" as the [Watch] window update mode. This interval can be set from 1 (= 100 ms, default) to 10 (= 1 second).

Single Step Mode

Choose whether to enable or disable interrupts while single-stepping a program. (See Table 13.8.4.3.) To enable interrupts, select (check) the check box.

Watch/Local repaint

Set the [Watch] window's update mode. The default real-time mode ([with Real Time] selected) is provided for running programs in real time. In this mode, the [Watch] window is updated after a break in program execution. In short break mode (with [Short Break] selected), the contents displayed in the window are updated at intervals set by [Run Monitor Interval]. In this mode, however, program execution is temporarily suspended so that display can be updated. Therefore, programs cannot be run in real time.

Run-time measurement base

The ICE contains a 31-bit execution cycle counter, allowing you to measure the time and number of bus cycles in which a program was run continuously. The run time here can be measured in units of 1 µs (default) or 62.5 ns as selected with this option. Bus cycles can be counted up to 2,147,483,647 cycles (with ±0 error).

The maximum times that can be measured are shown below.

When measured in 1 µs units: About 35 minutes, 50 seconds (with ±1 µs error)
When measured in 62.5 ns units: About 2 minutes, 15 seconds (with ±62.5 ns error)

The measurement result is displayed in the [Command] window after a break in continuous program execution, as shown below.

```
Example: >g
         BUS CYCLE : 428649                       ... Number of bus cycles
         Mode L    : 001min 002s  543ms 468us     ... Execution time (in 1 µs units, default)

         >g
         BUS CYCLE : 35095                        ... Number of bus cycles
         Mode L    : 003s  094ms 152us 0.0ns      ... Execution time (in 62.5 ns units)
```

If the counter's maximum count is exceeded, the debugger indicates "Count overflow" for the number of bus cycles and "Time over" for the execution time.
The counter is reset when successive program execution starts.
No measurements are made when single-stepping a program.

Resetting the CPU

Table 13.8.4.4 Commands/menu items/tool bar buttons for resetting CPU

Function	Command	Menu	Button
Reset CPU	rst	[Run | Reset CPU]	
Successive execution after resetting CPU	gr	[Run | Go after Reset]	

The CPU is reset when the gr command is executed, or by executing the rst command.
The following shows the initial settings when the CPU is reset.

(1) Internal registers of the CPU and memory

The CPU internal registers are initialized as follows during initial reset:

PC:	Reset exception processing loads the reset vector stored in bank 0, 000000H–000001H into the PC.
SP, IX, IY:	0xAAAA
B, A, H, L, BR:	0xAA
CB, NB:	0x01
EP, XP, YP:	0x00
SC:	0b11000000
CC:	0b1111

The internal RAM and external RAM are not initialized at initial reset.
The respectively stipulated initializations are done for internal peripheral circuits.

(2) Redisplaying the [Source] and [Register] windows

Because the PC is reset, the [Source] window is redisplayed beginning with that address.
The [Register] window is redisplayed with the settings above.

13.8.5 Break Functions

The target program is made to stop executing by one of the following causes:

- Break conditions set by a break set up command are met. (for successive execution only)
- A break signal is input to the ICE BRKIN pin. (for successive execution only)
- The [Key Break] button is clicked, the [Run | Stop] menu command is selected or the [Esc] key is pressed.
- A program execution error is detected.

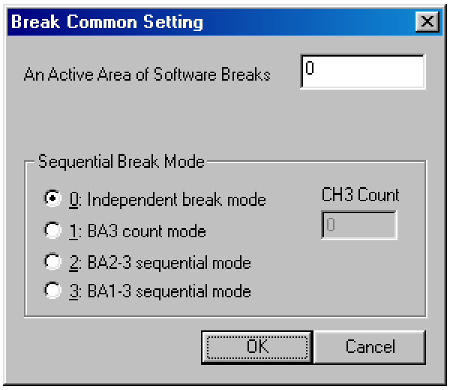
Break by command

The debugger has three types of break functions that allow the break conditions to be set by a command. When the set conditions in one of these break functions are met, the program under execution is made to break.

(1) Software breakpoints and a software break area

When the PC matches a set address, a break occurs. The program fetches the instruction from that address and breaks before executing that instruction. Software breakpoints can be set at up to 64 separate addresses and in one area with a specified address range.

However, these breakpoints are effective in only a 1 MB active break area. If any address outside this area is specified, no breaks can occur at that address, although the address is registered as an invalid breakpoint. The 8 MB of code space is divided into eight 1 MB active break areas, one of which can be selected from the [Break Common Setting] dialog box that is displayed by the [Break | Setting...] menu command. At debugger startup, a 1 MB area (from 0x0 to 0x0fffff) is automatically selected as the active break area.



Break Common Setting

An Active Area of Software Breaks: 0

Sequential Break Mode

● 0: Independent break mode

○ 1: BA3 count mode

○ 2: BA2-3 sequential mode

○ 3: BA1-3 sequential mode

CH3 Count: 0

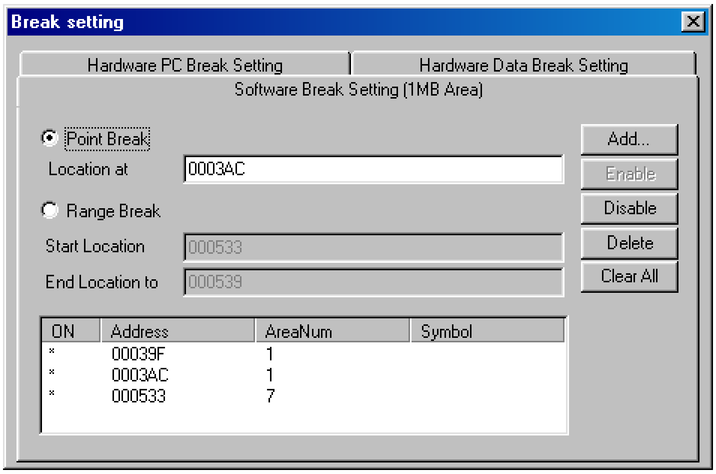
To select an active break area, enter your desired value in the [An Active Area of Software Breaks] text box. A value from 0 to 7 can be entered.

0: 0x000000 to 0x0fffff
1: 0x100000 to 0x1fffff
2: 0x200000 to 0x2fffff
:
7: 0x700000 to 0x7fffff

Table 13.8.5.1 Commands/menu items/tool bar button to set breakpoints

Function	Command	Menu	Button
Set software breakpoints	bp	[Break | Breakpoint Setting]	✋
Set software break area	bpa	[Break | Breakpoint Setting]	–
Clear software breakpoints	bpr bc (bpc)	[Break | Breakpoint Setting]	✋

Selecting [Breakpoint Setting] from the [Break] menu displays the [Break setting] dialog box. The [Software Break Setting (1MB Area)] tab of this dialog box shows a list of PC breakpoints that have been set.


Break setting
Hardware PC Break Setting
Hardware Data Break Setting
Software Break Setting (1MB Area)
Add...
Enable
Disable
Delete
Clear All

● Point Break

Location at: 0003AC

○ Range Break

Start Location: 000533

End Location to: 000539

ON	Address	AreaNum	Symbol
*	00039F	1	
*	0003AC	1	
*	000533	7	

To set a software breakpoint, select the [Point Break] radio button and enter an address in the [Location at] text box. Then click the [Add] button to register the address you entered as a valid breakpoint. Up to 64 breakpoints can be added to the list. Exceeding this limit prompts a warning. In such case, delete unnecessary breakpoints before adding a new one.

To set a software break area, select the [Range Break] radio button, then enter the start and end addresses of that area in the [Start Location] and [End Location to] text boxes, respectively. Then click the [Add] button to register the area you entered as a valid software break area. All addresses in that area are assumed to have breakpoints set. The start address of the area is shown in the Address column of the list, with area size (in bytes) shown in the AreaNum column. Setting a new area with a software break area already registered prompts a warning. In such case, delete the registered software break area before setting a new one.

Any address including those in a software break area can be registered only once as a breakpoint. Neither addresses nor areas (that contain a breakpoint address) can be set twice or more as a breakpoint or break area.

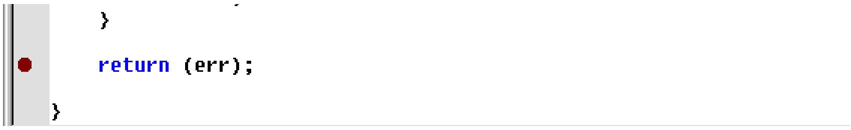
To disable a valid breakpoint (whose address is preceded by an asterisk (*) in the list), select that address from the list (by clicking the ON part), then click the [Disable] button. The asterisk disappears and the breakpoint is disabled.

To enable an invalid breakpoint, select the address of that breakpoint from the list, then click the [Enable] button. The address is then marked with an asterisk (*) to indicate that a breakpoint is enabled at that address.

To clear a breakpoint, select the address of that breakpoint from the list, then click the [Delete] button. The [Clear All] button allows you to clear all breakpoints that have been set, including those in a software break area.

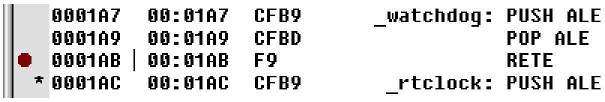
The addresses that are set as PC breakpoints are marked with a ● as they are displayed in the [Source] window.

Example in source display mode



```
      }

●     return (err);

  }
```

Example in disassemble display mode



```
  0001A7  00:01A7  CFB9      _watchdog: PUSH ALE
  0001A9  00:01A9  CFBD                 POP ALE
● 0001AB  00:01AB  F9                   RETE
* 0001AC  00:01AC  CFB9      _rtclock:  PUSH ALE
```

Using the [Break] button easily allows the setting and canceling of breakpoints.

✋ *[Break] button*

Click on the line in the [Source] window at where the program break is desired (after moving the cursor to that position) and then click on the [Break] button. A ● mark will be placed at the beginning of the line indicating that a breakpoint has been set there, and the address is registered in the breakpoint list. Clicking on the line that begins with a ● and then the [Break] button cancels the breakpoint you have set, in which case the address is deleted from the breakpoint list.

Setting breakpoints during source display mode

In the [Source] window in source display mode, there are lines at which breakpoints can be set and those at which breakpoints cannot be set. No breakpoints can be set in source lines that do not have actual code generated.

```
Example: 1     void func(void)        // NG
         2     {                      // OK
         3          int a;            // NG
         4          int x=0;          // OK
         5          a = x;            // OK
         6     }                      // OK
```

Line 1 is a function declaration that does not have actual code (same as a label declaration in the assembler). A breakpoint cannot be set here.
Line 3 is a variable declaration that does not have actual code. A breakpoint cannot be set here.
Line 4 is a variable declaration that has initialization code generated for it. A breakpoint can be set here.
Line 2 allows a breakpoint to be set. However, the breakpoint is set in line 4 (instruction at the beginning of that function).
Line 5 is an effective line that has actual code. A breakpoint can be set here.
Line 6 is a function termination (equivalent to mnemonic ret). A breakpoint can be set here.

However, if optimized during compiling, some lines become unusable in terms of setting a breakpoint. In the above example, since nothing is derived by executing each line (rewriting of only local variables involved, and that of global variables nonexistent), the actual code may be lost by optimization.

The same applies for lines whose execution can be halted by the [Go to Cursor] button.

(2) Sequential break function

The sequential break function causes a break to occur after the target program executes specified addresses following a specific sequence.
Three channels (BA1 to BA3) are provided for use in sequential breaks. On address can be set individually for each channel. For BA3, an execution count or number of times the program is to be run can be set, in addition to a break address.
The break addresses set here are effective in the entire code space, regardless of where active break area is selected.

One of the following four sequential break modes can be set depending on the channels used.

Independent break mode

In this mode, each channel acts as an independent breakpoint. When a program fetches the instruction at the address set on the channel, a break occurs before the program can execute that instruction. The run count specified for BA3 is not effective.

BA3 count mode

In this mode, program execution is made to break when the program has fetched the instruction at the address set on BA3 the specified number of times. Settings on BA1 and BA2 are not effective.

BA2–3 sequential mode

In this mode, program execution is made to break when the program has fetched the instruction at the address set on BA3 the specified number of times after executing the instruction more than once at the address set on BA2. Setting on BA1 is not effective.

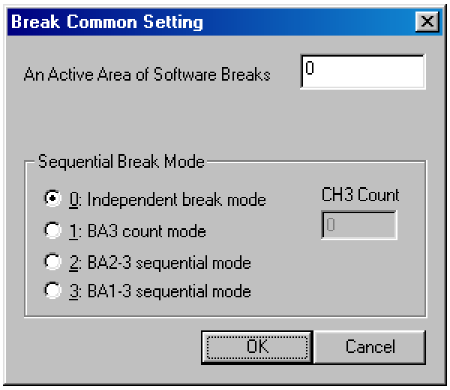
BA1–3 sequential mode

In this mode, program execution is made to break when the program has fetched the instruction at the address set on BA3 the specified number of times after executing the instructions more than once in that order at the addresses set on BA1 and BA2.

Table 13.8.5.2 Sequential break setting commands

Function	Command	Menu	Button
Set sequential break mode	bas	[Break | Setting...]	–
Set hardware breakpoints	ba	[Break | Breakpoint Setting]	–
Clear hardware breakpoints	bar	[Break | Breakpoint Setting]	–

To set sequential break mode, select [Setting...] from the [Break] menu.



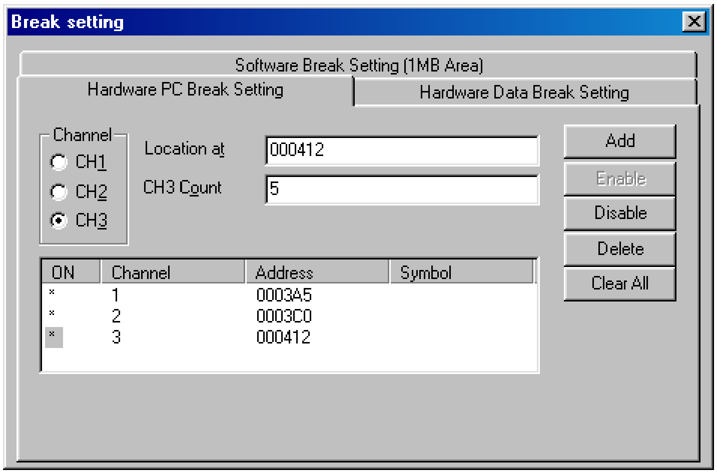
An Active Area of Software Breaks: 0

Sequential Break Mode
- ● 0: Independent break mode
- ○ 1: BA3 count mode
- ○ 2: BA2-3 sequential mode
- ○ 3: BA1-3 sequential mode

CH3 Count: 0

The [Break Common Setting] dialog box then appears. Select one of the [Sequential Break Mode] radio buttons from this dialog box to set the desired mode.
When you choose any radio button for BA3 counter-based mode, the [CH3 Count] text box becomes active. Therefore, enter an execution count in this text box. The program does not break until it fetches the instruction (the number of times as specified here) at the BA3 address.

To set an address on each channel, use the [Break setting] dialog box that appears when [Breakpoint Setting] is selected from the [Break] menu. When the [Break setting] dialog box appears, select the [Hardware PC Break Setting] tab in the dialog box.


Break setting
Software Break Setting (1MB Area)
Hardware PC Break Setting
Hardware Data Break Setting
Add
Enable
Disable
Delete
Clear All

Channel
○ CH1
○ CH2
● CH3

Location at 000412

CH3 Count 5

ON	Channel	Address	Symbol
*	1	0003A5	
*	2	0003C0	
*	3	000412	

Use the radio buttons to select the channel on which you want to set an address, then enter the desired address in the [Location at] text box.
To specify an execution count on BA3, enter a hexadecimal number for the desired count in the [CH3 Count] text box. If a count was set from the [Break Common Setting] dialog box, the value you entered is reflected in this text box.
Click the [Add] button to register the address you've set as a valid breakpoint. Each channel can have only one address set. Setting a new address on any channel for which an address is already set will overwrite the existing address. Also note that attempting to set an address that has already been registered as a hardware PC breakpoint will prompt a warning.
If addresses are set on each channel as shown above in BA1–3 sequential mode, program execution is made to break after the program executes instructions at each set address as follows:

1. Start running
   :
2. Execute instruction at address 0x0003A5 once or more
   :
3. Execute instruction at address 0x0003C0 once or more
   :
4. Execute instruction at address 0x000412 four times
   :
5. Fetch instruction at address 0x000412 again

At step 5, the program is made to break before executing the instruction at address 0x000412.

To disable a valid breakpoint (whose address is preceded by an asterisk (*) in the list), select that address from the list (by clicking the ON part), then click the [Disable] button. The asterisk disappears and the breakpoint is disabled.
To enable an invalid breakpoint, select the address of that breakpoint from the list, then click the [Enable] button. The address is marked with an asterisk (*) to indicate that a breakpoint is enabled at that address.
To clear a breakpoint, select the address of that breakpoint from the list, then click the [Delete] button.
The [Clear All] button allows you to clear all breakpoints that have been set.

(3) Data break function

The data break function causes a break to occur when a program accesses memory as specified. Four channels (CH0 to CH3) are provided for use in data breaks. The following three conditions can be specified on each channel individually.

Address When an address is specified, the target program is made to break when it accesses that address.

Data When data is specified, the target program is made to break when it writes or reads the specified data. Here, specify one byte of data. The data bits can be masked so that the program can be made to break when only the desired (but not all) bits match.

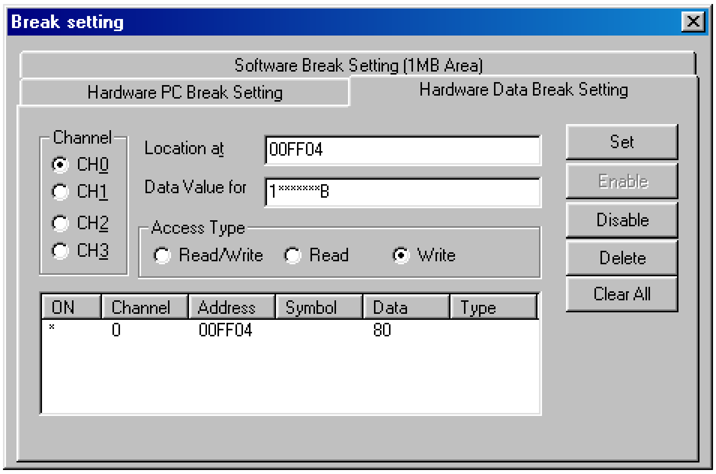
Read/write The program can be made to break in only a read or a write cycle or in both, as specified.

Of the above, specify one or more conditions. When two or more conditions are specified, the program is made to break after accessing memory to satisfy all specified conditions.

Table 13.8.5.3 Data break setting commands

Function	Command	Menu	Button
Set data break conditions	bd	[Break | Breakpoint Setting]	–
Clear data break conditions	bdr	[Break | Breakpoint Setting]	–

Select [Breakpoint Setting] from the [Break] menu to display the [Break setting] dialog box. Select (click) the [Hardware Data Break Setting] tab in the dialog box.


Break setting
Software Break Setting (1MB Area)
Hardware PC Break Setting
Hardware Data Break Setting
Set
Enable
Disable
Delete
Clear All

Channel
● CH0
○ CH1
○ CH2
○ CH3

Location at: 00FF04

Data Value for: 1*******B

Access Type: ○ Read/Write ○ Read ● Write

ON	Channel	Address	Symbol	Data	Type
*	0	00FF04		80	

Use the radio buttons to select the channel on which you want to set break conditions, then enter an address in the [Location at] text box and data in the [Data Value for] text box (optional). Use the radio buttons to select the desired read / write condition, then click the [Set] button to register what you've entered as valid break conditions. Note that setting a new condition on any channel for which conditions are already set will overwrite the existing conditions.

In the above example, the target program is made to break when it writes data whose MSB = 1 to address 0x00ff04.

To disable valid break conditions on any channel (preceded by an asterisk (*) in the list), select that channel from the list (by clicking the ON part), then click the [Disable] button. The asterisk disappears and break conditions on the channel are disabled.
To enable invalid break conditions on any channel, select that channel from the list, then click the [Enable] button. The channel is marked with an asterisk (*) to indicate that break conditions are enabled on the channel.
To clear break conditions on any channel, select that channel from the list, then click the [Delete] button. The [Clear All] button allows you to clear all break conditions that have been set.

(4) Other break commands

Commands are available to display all break conditions set in the [Command] window and to clear all break conditions.

Table 13.8.5.4 Other break commands

Function	Command	Menu	Button
Display all break conditions	bl	[Break | Break List]	–
Clear all break conditions	bac	[Break | Break All Clear]	✋

Forced break

The [Key Break] button, [Run | Stop] menu command, and [ESC] key can be used to forcibly terminate the program being executed.

⊗ *[Key Break] button*

Low level input to the ICE BRKIN pin

By setting the BRKIN pin of the ICE to LOW, a break occurs at the rising edge of the signal.

Break due to program execution error

A break occurs when the ICE has detected one of the operations below during a program execution.

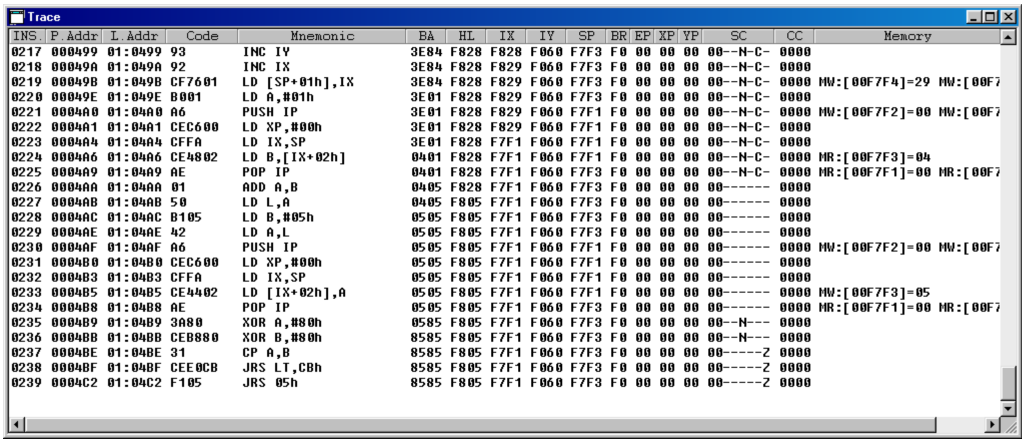
- Writing data to the ROM area
- Stack operation outside of the stack area
- Access to an undefined area
- Executing an illegal instruction (that is not available in the model)

These errors are detected using the memory and other information described in the parameter file.

13.8.6 Trace Functions

Trace data buffer and trace information

The ICE has a trace data buffer. When the debugger executes the program, the trace information on each executed instruction is taken into this buffer. The trace data buffer has the capacity to store information for 8,192 cycles. When the trace information exceeds this capacity, the data is overwritten, the oldest data first. Consequently, the trace information stored in the trace data buffer is always within 8,192 cycles. The trace data buffer is cleared when a program is executed, starting to trace the new execution data.



```
Trace
INS. P.Addr L.Addr  Code    Mnemonic           BA   HL   IX   IY   SP   BR EP XP YP SC       CC   Memory
0217 000499 01:0499 93      INC IY             3E84 F828 F828 F060 F7F3 F0 00 00 00 00--N-C- 0000
0218 00049A 01:049A 92      INC IX             3E84 F828 F829 F060 F7F3 F0 00 00 00 00--N-C- 0000
0219 00049B 01:049B CF7601  LD [SP+01h],IX     3E84 F828 F829 F060 F7F3 F0 00 00 00 00--N-C- 0000 MW:[00F7F4]=29 MW:[00F7
0220 00049E 01:049E B001    LD A,#01h          3E01 F828 F829 F060 F7F3 F0 00 00 00 00--N-C- 0000
0221 0004A0 01:04A0 A6      PUSH IP            3E01 F828 F829 F060 F7F1 F0 00 00 00 00--N-C- 0000 MW:[00F7F2]=00 MW:[00F7
0222 0004A1 01:04A1 CEC600  LD XP,#00h         3E01 F828 F829 F060 F7F1 F0 00 00 00 00--N-C- 0000
0223 0004A4 01:04A4 CFFA    LD IX,SP           3E01 F828 F7F1 F060 F7F1 F0 00 00 00 00--N-C- 0000
0224 0004A6 01:04A6 CE4802  LD B,[IX+02h]      0401 F828 F7F1 F060 F7F1 F0 00 00 00 00--N-C- 0000 MR:[00F7F3]=04
0225 0004A9 01:04A9 AE      POP IP             0401 F828 F7F1 F060 F7F3 F0 00 00 00 00--N-C- 0000 MR:[00F7F1]=00 MR:[00F7
0226 0004AA 01:04AA 01      ADD A,B            0405 F828 F7F1 F060 F7F3 F0 00 00 00 00------ 0000
0227 0004AB 01:04AB 50      LD L,A             0405 F805 F7F1 F060 F7F3 F0 00 00 00 00------ 0000
0228 0004AC 01:04AC B105    LD B,#05h          0505 F805 F7F1 F060 F7F3 F0 00 00 00 00------ 0000
0229 0004AE 01:04AE 42      LD A,L             0505 F805 F7F1 F060 F7F3 F0 00 00 00 00------ 0000
0230 0004AF 01:04AF A6      PUSH IP            0505 F805 F7F1 F060 F7F1 F0 00 00 00 00------ 0000 MW:[00F7F2]=00 MW:[00F7
0231 0004B0 01:04B0 CEC600  LD XP,#00h         0505 F805 F7F1 F060 F7F1 F0 00 00 00 00------ 0000
0232 0004B3 01:04B3 CFFA    LD IX,SP           0505 F805 F7F1 F060 F7F1 F0 00 00 00 00------ 0000
0233 0004B5 01:04B5 CE4402  LD [IX+02h],A      0505 F805 F7F1 F060 F7F1 F0 00 00 00 00------ 0000 MW:[00F7F3]=05
0234 0004B8 01:04B8 AE      POP IP             0505 F805 F7F1 F060 F7F3 F0 00 00 00 00------ 0000 MR:[00F7F1]=00 MR:[00F7
0235 0004B9 01:04B9 3A80    XOR A,#80h         0585 F805 F7F1 F060 F7F3 F0 00 00 00 00--N--- 0000
0236 0004BB 01:04BB CEB880  XOR B,#80h         8585 F805 F7F1 F060 F7F3 F0 00 00 00 00--N--- 0000
0237 0004BE 01:04BE 31      CP A,B             8585 F805 F7F1 F060 F7F3 F0 00 00 00 00-----Z 0000
0238 0004BF 01:04BF CEE0CB  JRS LT,CBh         8585 F805 F7F1 F060 F7F3 F0 00 00 00 00-----Z 0000
0239 0004C2 01:04C2 F105    JRS 05h            8585 F805 F7F1 F060 F7F3 F0 00 00 00 00-----Z 0000
```

The following lists the trace information that is taken into the trace data buffer in every instruction execution cycle. This list is corresponded to display in the [Trace] window.

INS: Executed cycle number (0 to 8191, decimal)
0000 means oldest trace data.

P Addr: PC address (hexadecimal physical address)

L Addr: PC address (hexadecimal logical address)

Code: Instruction code (hexadecimal)

Mnemonic: Disassembled instruction code

BA to YP: Values of the CPU registers (hexadecimal)

SC, CC: Condition flag status

Memory: Memory access status (other than code fetch status)
MR: Memory read
MW: Memory write
[<address>] = <data>: Accessed memory address and read / write data (hexadecimal)

Trace modes

Two trace modes are provided for selection depending on how trace information is captured.

All trace mode

Information is recorded on all bus cycles executed. In this mode, the latest trace data (for up to 8,192 cycles) can always be obtained.

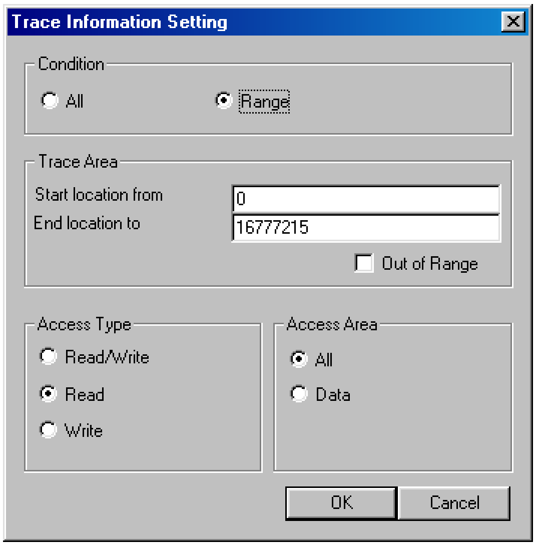
Range-specified trace mode

In this mode, memory access conditions can be specified. Information is only recorded on the bus cycles that match the specified conditions.

The following lists the memory access conditions that can be specified:

- Specify an address range and whether to trace inside or outside the specified address range
- Specify whether to trace both program fetch and data read / write cycles, or only data read / write cycles
- Specify whether to trace either read or write cycles (or both)

To set trace mode, select [Setting...] from the [Break] menu.



Trace Information Setting

Condition

○ All ● Range

Trace Area

Start location from: 0

End location to: 16777215

□ Out of Range

Access Type

○ Read/Write

● Read

○ Write

Access Area

● All

○ Data

To set all trace mode, select the [All] radio button and click [OK].

To set range-specified trace mode, select the [Range] radio button, then specify an address range by entering the start and end addresses in the [Start location from] and [End location to] text boxes in decimal notation, respectively. To trace outside that address range, select the [Out of Range] radio button. Then select a read / write condition with the [Access Type] radio button. Use the radio buttons under [Access Area] to specify that all accesses be traced (All) or only data read / write accesses be traced (Data). After making the above selections, click the [OK] button.

To stop setting trace mode, click the [Cancel] button.

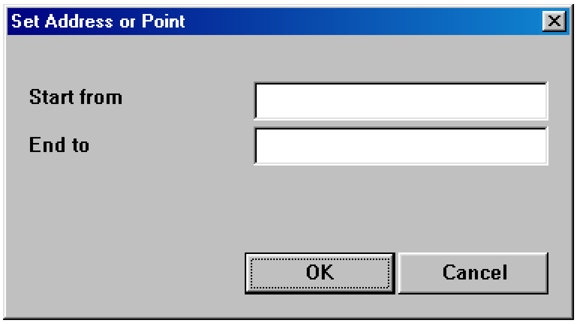
Displaying and searching trace information

The sampled trace information is displayed in the [Trace] window after a program execution has finished. In the [Trace] window, the entire trace data buffer can be seen by scrolling the window. The trace information can be displayed beginning from a specified cycle using a command. The display contents are as described above.
If the [Trace] window is closed, the information can be displayed in the [Command] window using a command.

Table 13.8.6.1 Command/menu item to display trace information

Function	Command	Menu	Button
Display trace information	td	[Trace | Trace]	–

When [Trace] is selected from the [Trace] menu, the dialog box shown below appears.



Enter the display start and end cycle numbers (in hexadecimal) to the [Start from] and [End to] text boxes, respectively, and then click the [OK] button. When number entry is omitted, the debugger assumes the start cycle number is 0 and the end cycle number is 0x1fff (8191).
To cancel trace data display, click the [Cancel] button.

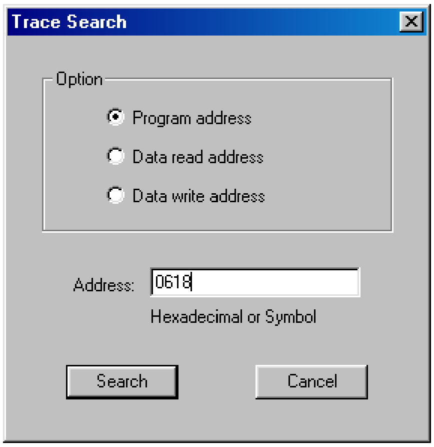
It is possible to specify a search condition and display the trace information that matches a specified condition.
The search condition can be selected from the following three:
1. Program's execution address
2. Address from which data is read
3. Address to which data is written

When the above condition and one address are specified, the system starts searching. When the trace information that matches the specified condition is found, the system displays the found data in the [Trace] window (or in the [Command] window if the [Trace] window is closed).

Table 13.8.6.2 Command/menu item to search trace information

Function	Command	Menu	Button
Search trace information	ts	[Trace | Trace Search...]	–

When [Trace Search...] is selected from the [Trace] menu, the dialog box shown below appears.



Address: 0618

Choose a search condition using the radio button, enter an address, and then click the [Search] button.
To cancel searching trace data, click the [Cancel] button.

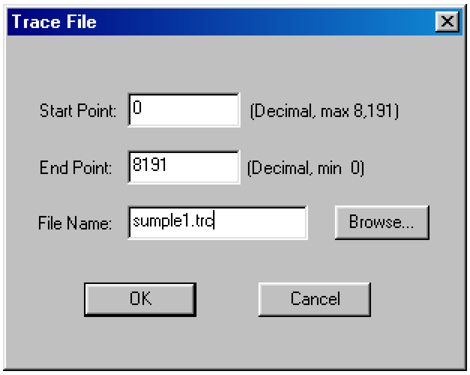
Saving trace information

The trace information within the specified range can be saved to a file.

Table 13.8.6.3 Command/menu item to save trace information

Function	Command	Menu	Button
Save trace information	tf	[Trace | Trace File...]	–

When [Trace File...] is selected from the [Trace] menu, the dialog box shown below appears.



Start Point: 0 (Decimal, max 8,191)

End Point: 8191 (Decimal, min 0)

File Name: sumple1.trc

Enter the start and end cycle numbers of the range to be saved to the [Start Point] and [End Point] text boxes, respectively.
Enter the file name to the [File Name] text box or choose a folder / file using the [Browse...] button.
Then click the [OK] button to start saving.
To cancel saving trace data, click the [Cancel] button.

13.8.7 Coverage

The ICE has a coverage function that allows you to record the memory addresses accessed.
The coverage information is recorded according to the acquisition mode and acquisition range specified with the debugger's coverage options.

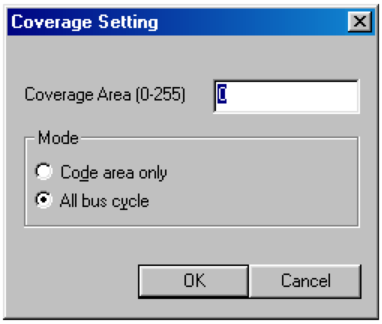
Acquisition mode

Specify whether to acquire coverage information for access to both code and data spaces, or for access to only code space. By default, coverage information is acquired for access to both code and data spaces.

Acquisition range

The ICE divides the 16-MB address space into 64 KB × 256 areas, with coverage information acquired from each 64-KB area. A 64-KB area from 0x00000 to 0x00FFFF is the default acquisition range. Therefore, if coverage information must be acquired from another area, you should specify that area before running the program.

To set coverage options, select [Setting...] from the [Coverage] menu.



Coverage Setting

Coverage Area (0-255): 0

Mode
○ Code area only
● All bus cycle

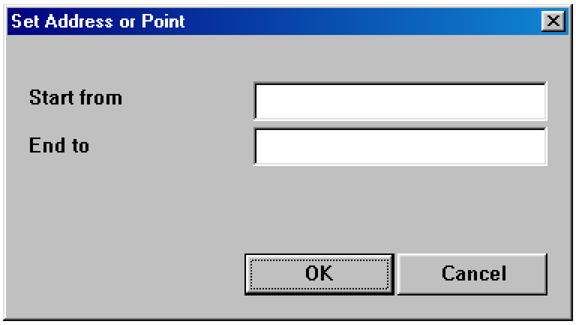
Enter a numeric value from 0 to 255 in the [Coverage Area (0-255)] text box to specify the desired acquisition range. Use the radio buttons to select the desired acquisition mode. Click the [OK] button to confirm what you've set.
To stop setting coverage, click the [Cancel] button.

The acquired coverage information can be displayed in the [Coverage] window.

Table 13.8.7.1 Coverage commands

Function	Command	Menu	Button
Display coverage information	cv	[Coverage | Coverage]	–
Clear coverage information	cvc	[Coverage | Coverage Clear]	–

Selecting [Coverage] from the [Coverage] menu opens the [Coverage] window, and the dialog box shown below appears.



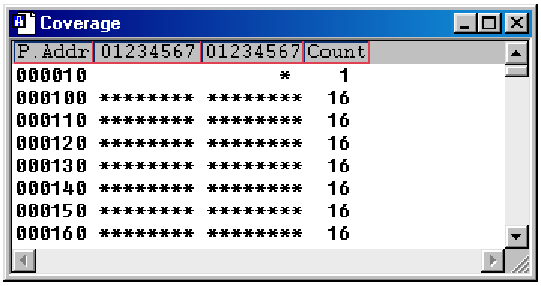
Set Address or Point

Start from:

End to:

Enter the address in hexadecimal notation from which to start displaying coverage information in the [Start from] text box, then click the [OK] button. To display coverage information in the [Coverage] window, you can leave [End to] blank. Note that the start and the end addresses of the 64 KB area selected are assumed if start and end addresses are not entered in these text boxes.
To stop setting addresses, click the [Cancel] button.

Coverage information is displayed in the [Coverage] window as shown below.



```
P.Addr 01234567 01234567 Count
000010                *      1
000100 ******** ********    16
000110 ******** ********    16
000120 ******** ********    16
000130 ******** ********    16
000140 ******** ********    16
000150 ******** ********    16
000160 ******** ********    16
```

Coverage information is displayed 16 bytes per row. P.Addr indicates the start address (physical address) of each line. The accessed addresses are marked with an asterisk (∗), and addresses not accessed are marked with a space " ". The Count value indicates the total addresses accessed (in bytes) among the 16 bytes on each line.

In addition to the [Coverage] window, the executed addresses in the [Source] window are marked with an asterisk (∗), except in source display mode.

Executing the cv command while the [Coverage] window is closed displays information in the [Command] window as shown below.

```
Example:  >cv 0
          00001e
          000100 - 00010f
                 :
```

13.8.8 Writing Data to the FPGA on the Standard Peripheral Circuit Board

The standard peripheral circuit board S5U1C88000P is configured for the supported model by writing the peripheral function data to the on-board FPGA. This writing is necessary the first time the standard peripheral circuit board is used or before beginning development of another model.
The debugger supports the following FPGA data handling functions:

(1) Erasing FPGA

All contents of the FPGA are erased.

(2) Writing data to FPGA

Data in the specified file is written to the FPGA. Also, the write command supports erasing the FPGA. Data for the supported models are provided as "c88xxx.mot" files in the "epson\s1c88\ice\fpga" directory (default).

(3) FPGA data comparison

The contents of the FPGA and specified file are compared.

(4) FPGA data dump

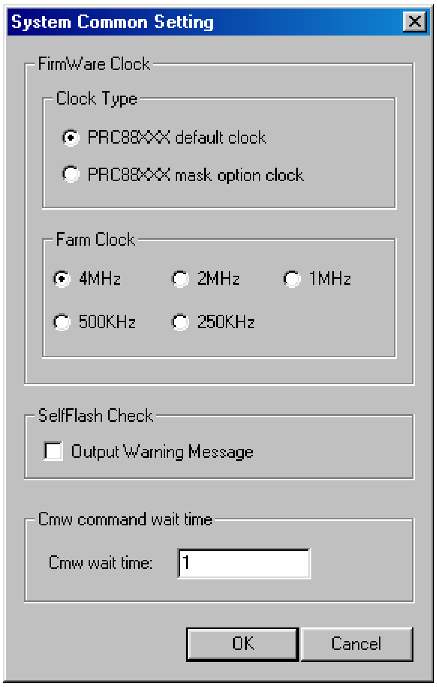
The FPGA data is displayed in a hexadecimal dump format.

Table 13.8.8.1 FPGA commands

Function	Command	Menu	Button
Erase FPGA	xfer	–	–
Write data to FPGA	xfwr	–	–
Compare FPGA data	xfcp	–	–
Dump FPGA data	xdp	–	–

13.8.9 System Options

The [System Common Setting] dialog box that appears when [Setting...] is selected from the [Option] menu is provided to set the options associated with ICE hardware.



System Common Setting

FirmWare Clock

Clock Type

● PRC88XXX default clock

○ PRC88XXX mask option clock

Farm Clock

● 4MHz ○ 2MHz ○ 1MHz

○ 500KHz ○ 250KHz

SelfFlash Check

□ Output Warning Message

Cmw command wait time

Cmw wait time: 1

Clock Type

One of the following two clocks can be selected for use in emulation:

(1) Default clock of peripheral board (default)

(2) Mask option clock of peripheral board

Selecting the peripheral board (PRC88XXX)'s default clock means that the clock on the peripheral board is used as the clocking source during emulation regardless of how the mask option is set. Some MPUs do not support this default clock.

For details about clock frequencies, refer to the technical manual supplied with your MPU.

Firm Clock

One of the following five firmware clocks can be selected for the ICE:

(1) 4 MHz (selected by default)

(2) 2 MHz

(3) 1 MHz

(4) 500 kHz

(5) 250 kHz

The ICE uses the firmware clock to execute its debugging functions. For example, a memory dump is performed using the firmware clock. Therefore, if the target board you're using consists of a low-speed device or one that may cause a delay in data output, the memory dump contents and contents read out by running the program may not match. In such case, set the firmware clock to a lower appropriate frequency.

SelfFlash Check

Turn the SelfFlash or self-rewriting check function on or off. Although the SelfFlash check function is automatically set according to a description in the parameter file, this option may be used to forcibly turn it on or off.

Cmw command wait time

Specify an interval time at which to execute commands after loading a command file with the cmw command. The interval time can be set from 1 to 256 seconds in 1-second increments. The interval time initially is set to 1 second.

13.9 Command Reference

13.9.1 Command List

Table 13.9.1.1 lists the debug commands available with the debugger.

Table 13.9.1.1 Command list

Classification	Command	Function	Page
Memory operation	**dd** [<addr1> [<addr2>]] [{-B|-W|-L|-F|-D}] [<addr1> <@size>] [{-B|-W|-L|-F|-D}]	Dump memory data	137
	de [<addr> <data1> [..<data16>]]	Enter memory data	140
	df [<addr1> <addr2> <data>]	Fill memory area	142
	dm [<addr1> <addr2> <addr3>] [<addr1> <@size> <addr3>]	Copy memory area	143
	ds <addr1> {<addr2>|@<byte>}... ...{"<str>"|<data>[:{B|W|L}]} [S=<step>]	Search memory data	144
Register operation	**rd**	Display register values	145
	rs [<reg> <value>]	Modify register value reg={PC|SP|IX|IY|A|B|HL|BR|CB|EP|XP|YP|SC|I1|I0|U|D|N|V|Z|C}	146
Program execution	**g** [<addr>]	Execute program successively from current PC	148
	gr [<addr>]	Execute program successively after resetting CPU	150
	s [<step>]	Single stepping from current PC	151
	n [<step>]	Single stepping with skip subroutines	153
	se	Exit from subroutine	154
CPU reset	**rst**	Reset CPU	155
Break	**bp** {-|+|_} <addr>	Set software breakpoints	156
	bpa <addr1> <addr2>	Set software break area	158
	bpr **bc** [<addr>] **bpc** [<addr>]	Clear software breakpoints	160
	bas {0|1|2|3}	Set sequential break mode	161
	ba <ch> <addr> [<count>] <ch> {-|+|_}	Set hardware breakpoints	162
	bar	Clear hardware breakpoints	164
	bd <ch> [A=<addr>][D=<data>][{R|W|}] <ch> {-|+|_}	Set hardware data break condition	165
	bdr	Clear hardware data break condition	167
	bl	Display all break conditions	168
	bac	Clear all break conditions	169
Program display	**u** [<addr>]	Disassemble code display	170
	sc [<addr>]	Source display	172
	m [<addr>]	Mix display	174
Symbol display	**sy** [/a]	Display symbol list	176
	w <symbol> [;{H|D|Q|B}] [/A]	Display symbol information	177
Load file	**lf** [<file>]	Load program/option HEX file	178
	par [<file>]	Load parameter file	179
Trace	**td** [<cycle>]	Display trace information	180
	ts [{pc|dr|dw} <addr>]	Search trace information	183
	tf [<file> [<cycle1> [<cycle2>]]]	Save trace information	185
Coverage	**cv** [<addr1> [<addr2>]]	Display coverage information	186
	cvc	Clear coverage information	188
Command file	**com** [<file> [<interval>]]	Load and execute command file	189
	cmw [<file>]	Load and execute command file with execution interval	190
	rec [<file>]	Record executed commands to file	191
Log	**log** [<file>]	Logging	192
Map information	**ma**	Display map information	193
FPGA operation	**xfer**	Erase FPGA	194
	xfwr <file> ;{H|S} [;N]	Write FPGA data	195
	xfcp <file> ;{H|S}	Compare FPGA data	196
	xdp <addr1> [<addr2>]	Dump FPGA data	197
Quit	**q**	Quit debugger	198
Help	**?**	Display command usage	199

13.9.2 Reference for Each Command

The following sections explain all the commands by functions.

The explanations contain the following items.

Function

Indicates the functions of the command.

Format

Indicates the keyboard input format and parameters required for execution.

Example

Indicates a sample execution of the command.

Note

Shows notes on using.

GUI utility

Indicates a menu item or tool bar button if they are available for the command.

Notes:

- *In the command format description, the parameters enclosed by < > indicate they are necessary parameters that must be input by the user; while the ones enclosed by [ ] indicate they are optional parameters.*
- *The input commands are case-insensitive, you can use either upper case or lower case letters or even mixed.*
- *An error results if the number of parameters is not correct when you input a command using direct input mode.*

```
Error : Incorrect number of parameters
```

13.9.3 Memory Operation

dd (data dump)

Function

This command displays the content of the memory in a 16 words/line hexadecimal dump format.

Format

(1) >dd [<address1> [<address2>]] [<option>]↵ (direct input mode)

(2) >dd [<address1> @<size>] [<option>]↵ (direct input mode)

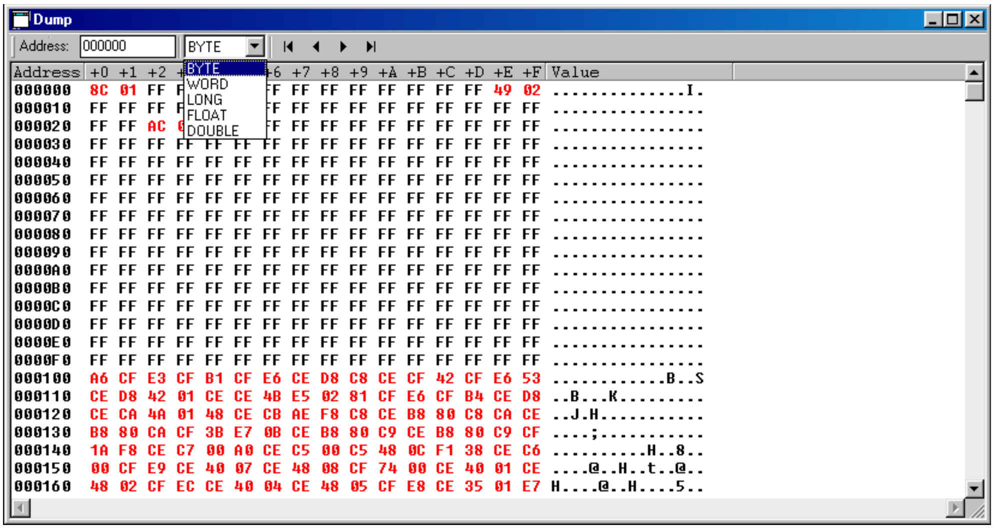
<address1>: Start address to display; hexadecimal or symbol (IEEE-695 format only)
<address2>: End address to display; hexadecimal or symbol (IEEE-695 format only)
<size>: Size of display area (in bytes); hexadecimal
<option>: Display format; specify with a symbol below.

-B	Byte (default)
-W	Word
-L	Long
-F	Float
-D	Double

Condition: 0 ≤ address1 ≤ address2 ≤ 0xffffff, 0 ≤ size ≤ 0xffffff

Display

(1) When [Dump] window is opened



```
Address +0 +1 +2 +           +6 +7 +8 +9 +A +B +C +D +E +F Value
000000  8C 01 FF F           F FF FF FF FF FF FF FF 49 02 ..............I.
000010  FF FF FF F           F FF FF FF FF FF FF FF FF FF ................
000020  FF FF AC 0           F FF FF FF FF FF FF FF FF FF ................
000030  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
000040  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
000050  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
000060  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
000070  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
000080  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
000090  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
0000A0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
0000B0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
0000C0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
0000D0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
0000E0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
0000F0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
000100  A6 CF E3 CF B1 CF E6 CE D8 C8 CE CF 42 CF E6 53 ............B..S
000110  CE D8 42 01 CE CE 4B E5 02 81 CF E6 CF B4 CE D8 ..B...K.........
000120  CE CA 4A 01 48 CE CB AE F8 C8 CE B8 80 C8 CA CE ..J.H...........
000130  B8 80 CA CF 3B E7 0B CE B8 80 C9 CE B8 80 C9 CF ....;...........
000140  1A F8 CE C7 00 A0 CE C5 00 C5 48 0C F1 38 CE C6 ..........H..8..
000150  00 CF E9 CE 40 07 CE 48 08 CF 74 00 CE 40 01 CE ....@..H..t..@..
000160  48 02 CF EC CE 40 04 CE 48 05 CF E8 CE 35 01 E7 H....@..H....5..
```

If both <address1> and <address2> are not defined, the [Dump] window is redisplayed beginning with address 0x000000.
If <address1> is defined , the [Dump] window is redisplayed in such a way that <address1> is displayed at the uppermost line.
Even when <address1> specifies somewhere in 16 addresses/line, data is displayed beginning with the top of that line. For example, even though you may have specified address 0x00ff08 for <address1>, data is displayed beginning with address 0x00ff00. However, if an address near the uppermost part of data memory (e.g. maximum address is 0xffffff), such as 0xffffc0, is specified as <address1>, the last line displayed in the window in this case is 0xfffff0, the specified address is not at the top of the window.
Since the [Dump] window can be scrolled to show the entire data memory, defining <address2> or @<size> does not have any specific effect. Only defining <address1> and both defining <address1> and <address2> or @<size> has same display result.

(2) When [Dump] window is closed

If both <address1> and <address2> are not defined, the debugger displays data for 256 words from address 0x000000 in the [Command] window.

```
>dd↵
Address +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +A +B +C +D +E +F Value
000000  AE 02 F0 F0 C9 02 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 ................
000010  00 A4 E0 48 0A 08 E0 80 EE 6A FC BA 3E BA 4A 01 ...H.....j..>.J.
  :                            :                               :
0000F0  A6 A2 22 82 A0 0C 04 02 FE F7 BD 9E FE 7F BA FB ..".............
>
```

If only <address1> is defined, the debugger displays data for 256 words from <address1>.

```
>dd ff00↵
Address +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +A +B +C +D +E +F Value
00FF00  30 00 00 FF FF FF FF FF FF FF FF FF FF FF FF FF 0...............
00FF10  00 00 1F 00 FF FF FF FF FF FF FF FF FF FF FF FF ................
  :                            :                               :
00FFF0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
>
```

If both <address1> and <address2> are defined, the debugger displays data from <address1> to <address2>.

```
>dd ff00 ff1f↵
Address +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +A +B +C +D +E +F Value
00FF00  30 00 00 FF FF FF FF FF FF FF FF FF FF FF FF FF 0...............
00FF10  00 00 1F 00 FF FF FF FF FF FF FF FF FF FF FF FF ................
>
```

If @<size> is defined in place of <address2>, the debugger displays the specified bytes of data from <address1>.

```
>dd ff00 @20↵
Address +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +A +B +C +D +E +F Value
00FF00  30 00 00 FF FF FF FF FF FF FF FF FF FF FF FF FF 0...............
00FF10  00 00 1F 00 FF FF FF FF FF FF FF FF FF FF FF FF ................
>
```

(3) Display format options

The display format option allows selection of a data type same as the pull-down list on the [Dump] window. When option specification is omitted, data is displayed in byte units. The following shows display examples in each option:

```
>dd -b↵                                                   ... Byte format (default)
Address +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +A +B +C +D +E +F Value
000000  AE 02 F0 F0 C9 02 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 ................
  :                            :
>dd -w↵                                                   ... Word format
Address  +0   +2   +4   +6   +8   +A   +C   +E  Value
000000  02AE F0F0 02C9 F0F0 F0F0 F0F0 F0F0 F0F0 ................
  :                        :
>dd -l↵                                                   ... Long format
000000  F0F002AE F0F002C9 F0F0F0F0 F0F0F0F0 ................
  :                       :
>dd -f↵                                                   ... Float format
000000  AE 02 F0 F0 -5.942371e+029
000004  C9 02 F0 F0 -5.942382e+029
  :                 :
>dd -d↵                                                   ... Double format
000000  AE 02 F0 F0 C9 02 F0 F0 -1.018151011077231e+236
000008  F0 F0 F0 F0 F0 F0 F0 F0 -1.077308742674321e+236
  :                            :
>
```

(4) During log output

If a command execution is being output to a log file by the log command when you dump the data memory, data is displayed in the [Command] window even if the [Dump] window is opened and are also output to the log file.
If the [Dump] window is closed, data is displayed in the [Command] window in the same way as in (2) above.
If the [Dump] window is open, it is redisplayed to show data in the same way as in (1) above. In this case, the same number of lines is displayed in the [Command] window as are displayed in the [Dump] window.

(5) Successive display

Once you execute the dd command, data can be displayed successively with the [Enter] key only until some other command is executed.
When you hit the [Enter] key, the [Dump] window is scrolled one full screen.
When displaying data in the [Command] window, data is displayed for the 16 lines following the previously displayed address (same number of lines as displayed in the [Dump] window during log output).

```
>dd↵
Address +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +A +B +C +D +E +F Value
000000  AE 02 F0 F0 C9 02 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 ................
000010  00 A4 E0 48 0A 08 E0 80 EE 6A FC BA 3E BA 4A 01 ...H.....j..>.J.
  :                            :                               :
0000F0  A6 A2 22 82 A0 0C 04 02 FE F7 BD 9E FE 7F BA FB ..".............
>↵
000100  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
000110  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
   :          :                :
0001F0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
>
```

Notes

- Both the start and end addresses specified here must be within the range of the memory area available with each microcomputer model.
  An error results if the limit is exceeded or the input one is not a hexadecimal number or not a valid symbol.
- An error results if the start address is larger than the end address.

GUI utility

[View | Dump] menu item

When this menu item is selected, the [Dump] window opens or becomes active and displays the current data memory contents.

de (data enter)

Function

This command rewrites the contents of the memory with the input hexadecimal data. Data can be written to continuous memory locations beginning with a specified address.

Format

(1) >de <address> <data1> [<data2> [...<data16>]]↵ **(direct input mode)**

(2) >de↵ **(guidance mode)**
Data enter address ? : <address>↵
Address *Original data* **: <data>↵**
....
>

<address>: Start address from which to write data; hexadecimal or symbol (IEEE-695 format only)
<data(1–16)>: Write data; hexadecimal
Condition: 0 ≤ address ≤ 0xffffff, 0 ≤ data ≤ 0xff

Examples

Format (1)

```
>de ff10 0↵                     ... Rewrites data at address 0x0xff10 with 0.
```

Format (2)

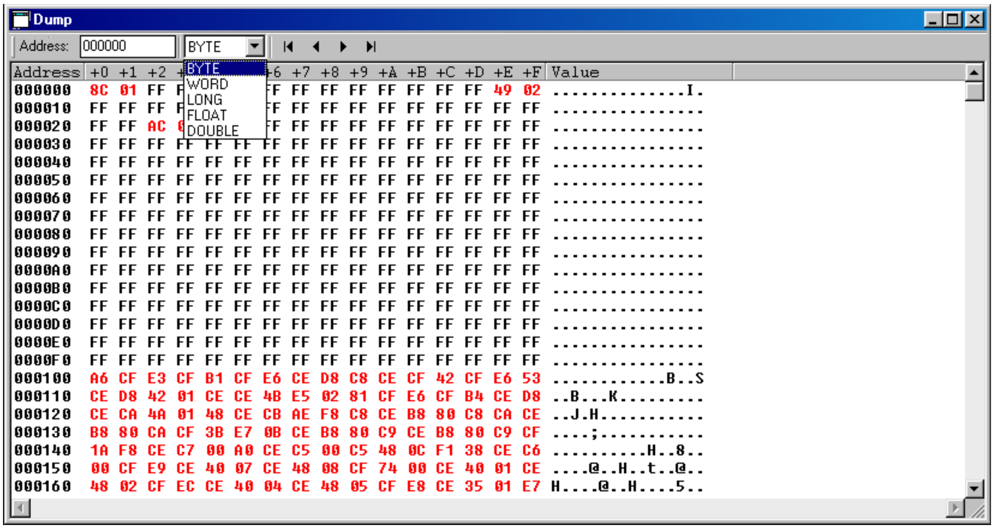
```
>de↵
Data enter address ? :ff10↵  ... Address is input.
00FF10   0 : a↵              ... Data is input.
00FF11   0 : ↵               ... Skipped.
00FF12   0 : q↵              ... Command is terminated.
>
```

Notes

- The start address specified here must be within the range of the memory area available with each microcomputer model.
  An error results if the limit is exceeded or the input one is not a hexadecimal number or a valid symbol.
- The contents of the unused area will be marked as "*". If you encounter any address marked by "*", press [Enter] key to skip that address or terminate the command.
- Data must be input using a hexadecimal number in the range of 8 bits (0–0xff). An error results if the limit is exceeded.
- When the contents of the data memory is modified using the de command, the displayed contents of the [Dump] window are updated automatically.
- In guidance mode, the following keyboard inputs have special meaning:
  "q↵" … Command is terminated. (finish inputting and start execution)
  "^↵" … Return to previous address.
  "↵" … Input is skipped. (keep current value)
  If the maximum address of data memory is reached and gets a valid input other than "^↵", the command is terminated.

GUI utility

[Dump] window



Address: 000000 BYTE

```
Address +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +A +B +C +D +E +F Value
000000  8C 01 FF F         F FF FF FF FF FF FF FF 49 02 ..............I.
000010  FF FF FF F         F FF FF FF FF FF FF FF FF FF ................
000020  FF FF AC 0         F FF FF FF FF FF FF FF FF FF ................
000030  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
000040  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
000050  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
000060  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
000070  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
000080  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
000090  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
0000A0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
0000B0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
0000C0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
0000D0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
0000E0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
0000F0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
000100  A6 CF E3 CF B1 CF E6 CE D8 C8 CE CF 42 CF E6 53 ............B..S
000110  CE D8 42 01 CE CE 4B E5 02 81 CF E6 CF B4 CE D8 ..B...K.........
000120  CE CA 4A 01 48 CE CB AE F8 C8 CE B8 80 C8 CA CE ..J.H...........
000130  B8 80 CA CF 3B E7 0B CE B8 80 C9 CE B8 80 C9 CF ....;...........
000140  1A F8 CE C7 00 A0 CE C5 00 C5 48 0C F1 38 CE C6 ..........H..8..
000150  00 CF E9 CE 40 07 CE 48 08 CF 74 00 CE 40 01 CE ....@..H..t..@..
000160  48 02 CF EC CE 40 04 CE 48 05 CF E8 CE 35 01 E7 H....@..H....5..
```

The [Dump] window allows direct modification of memory contents. To modify data on the [Dump] window, place the cursor at the front of the data to be modified or double click the data, and then type a hexadecimal character (0–9, a–f). Data in the address will be modified with the entered number and the cursor will move to the next address. This allows successive modification of a series of addresses.

df (data fill)

Function

This command rewrites the contents of the specified memory area with the specified data.

Format

(1) >df <address1> <address2> <data>↵ **(direct input mode)**

(2) >df↵ **(guidance mode)**

```
Start address ? <address1>↵
End address ? <address2>↵
Data pattern ? <data>↵
>
```

<address1>: Start address of specified range; hexadecimal or symbol (IEEE-695 format only)
<address2>: End address of specified range; hexadecimal or symbol (IEEE-695 format only)
<data>: Write data; hexadecimal
Condition: 0 ≤ address1 ≤ address2 ≤ 0xffffff, 0 ≤ data ≤ 0xff

Examples

Format (1)

```
>df ff200 ff2ff 0↵        ... Fills the memory area from address 0xff200 to address 0xff2ff with 0x0.
```

Format (2)

```
>df↵
Start address ? ff200↵  ... Start address is input.
End address ? ff2ff↵    ... End address is input.
Data pattern ? 0↵       ... Data is input.
>
```

* Command execution can be canceled by entering only the [Enter] key and nothing else.

Notes

- Both the start and end addresses specified here must be within the range of the memory area available with each microcomputer model.
  An error results if the limit is exceeded or the input one is not a hexadecimal number or a valid symbol.
- An error results if the start address is larger than the end address.
- Data must be input using a hexadecimal number in the range of 8 bits (0 to 0xff). An error results if the limit is exceeded.
- Write operation is not performed to the read only address of the I/O area.
- When there is an unused area in the specified address range, no error occurs. The area other than the unused area will be filled with the specified data.
- When the contents of the data memory is modified using the df command, the displayed contents of the [Dump] window are updated automatically.

GUI utility

None

dm (data move)

Function

This command copies the contents of the specified memory area to another area.

Format

(1) >dm <address1> <address2> <address3>↵ (direct input mode)

(2) >dm <address1> @<size> <address3>↵ (direct input mode)

(3) >dm↵ (guidance mode)
Start address ? <address1>↵
End address ? <address2>↵
Destination address ? <address3>↵
>

<address1>: Start address of source area to be copied from; hexadecimal or symbol (IEEE-695 format only)
<address2>: End address of source area to be copied from; hexadecimal or symbol (IEEE-695 format only)
<address3>: Address of destination area to be copied to; hexadecimal or symbol (IEEE-695 format only)
<size>: Size of the source area (in bytes); hexadecimal
Condition: 0 ≤ address1 ≤ address2 ≤ 0xffffff, 0 ≤ address3 ≤ 0xffffff, 0 ≤ size ≤ 0xffffff

Examples

Format (1)

```
>dm ff200 ff2ff ff280↵            ... Copies data within the range from address 0xff200 to address
                                      0xff2ff to the area from address 0xff280.
```

Format (2)

```
>dm ff200 @100 ff280↵             ... Same as above.
```

Format (3)

```
>dm↵
Start address ? ff200↵            ... Source area start address is input.
End address ? ff2ff↵              ... Source area end address is input.
Destination address ? ff280↵      ... Destination area start address is input.
>
```

* Command execution can be canceled by entering only the [Enter] key and nothing else.

Notes

- All the addresses specified here must be within the range of the memory area available with each microcomputer model.
  An error results if the limit is exceeded or the input one is not a hexadecimal number or a valid symbol.
- Write operation is not performed to the read-only address of the I/O area.
- Data in the write-only area cannot be read. If the source area contains write-only address, 0 is written to the corresponding destination. If the destination area contains read-only address, the data of that address can not be rewritten. If the source and destination areas contain I/O address of mixed read-only bits and write-only bits, either read or write operation can be executed for the corresponding bits.
- When the contents of the data memory is modified using the dm command, the displayed contents of the [Dump] window are updated automatically.

GUI utility

None

ds (data search)

Function

This command searches for a specified data or string from a specified range of memory. When the search data or string is found, the address of the data or string found is indicated in the [Command] window. In addition, if specified data is found in the address range displayed in the [Dump] window, the data found is displayed in green.

Format

>ds <address1> {<address2>|@<byte>} {"<string>"|<data> [:<size>]} [S=<step>]↵

(direct input mode)

<address1>: Start address of search range; hexadecimal or symbol (IEEE-695 format only)
<address2>: End address of search range; hexadecimal or symbol (IEEE-695 format only)
<byte>: Size of search range (in bytes); hexadecimal
<string>: String to search, consisting of up to four ASCII characters
<data>: Data to search, equal in size to <size> represented in hexadecimal or binary notation. The data bytes or bits can be masked with an asterisk (*).
<size>: Data size, specifying using the following symbols:
B for byte (1 byte) (default)
W for word (2 bytes)
L for long (4 bytes)
<step>: Step (in bytes) in which increments to search, equal to data size (specified by <size>) when omitted
Condition: 0 ≤ address1 ≤ address2 ≤ 0xffffff, address2 ≤ address1+0xffff, byte ≤ 0x10000,
1 ≤ step ≤ 0xffff

Examples

```
>ds f000 30:W S=10↵
00F000  00F070
>
```

In this example, the command searches for word data "0x0030" starting from address 0x00f000. Because the step is specified to be 16 bytes, word data at only the 16-byte boundary addresses (0x00f000, 0x00f010, ...) are checked. Even if word data "0x0030" exists at address 0xf002, for example, it does not appear in the search result.

```
>ds f000 f0ff "ABC"↵
00F022
>
```

In this example, the command searches for string "ABC" (= 0x41, 0x42, 0x43) in the address range from 0x00f000 to 0x00f0ff. The string is searched in byte steps or increments (default).

Notes

- The address specified here must be within the range of the memory area available with each microcomputer model.
  An error results if the limit is exceeded or the input one is not a hexadecimal number or a valid symbol.
- Search is made within a 64 KB range. Specifying an address exceeding this range results in an error.

GUI utility

None

13.9.4 Register Operation

rd (register display)

Function

This command displays the contents of the CPU registers.

Format

>rd↵ (direct input mode)

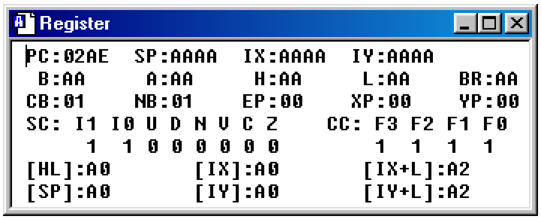
Display

(1) Contents of display

This command displays the contents of the following registers and memory addresses pointed by the registers.
Register: PC, SP, IX, IY, B, A, H, L, BR, SC, CC
Memory: [HL], [IX], [IX+L], [SP], [IY], [IY+L]

∗ If the memory locations indicated by the registers are in an unused area, data in that area is marked by an "∗" as it is displayed.

(2) When [Register] window is opened



```
PC:02AE   SP:AAAA   IX:AAAA   IY:AAAA
 B:AA      A:AA      H:AA      L:AA     BR:AA
CB:01     NB:01     EP:00     XP:00     YP:00
SC: I1 I0 U D N V C Z    CC: F3 F2 F1 F0
     1  1 0 0 0 0 0 0         1  1  1  1
[HL]:A0        [IX]:A0        [IX+L]:A2
[SP]:A0        [IY]:A0        [IY+L]:A2
```

When the [Register] window is opened, all the above contents are displayed in the [Register] window according to the program execution. When you use the rd command, the displayed contents of the [Register] window is updated.

(3) When [Register] window is closed

Data is displayed in the [Command] window in the following manner:

```
>rd↵
PC:02AE  SP:AAAA  IX:AAAA  IY:AAAA
 B:AA     A:AA     H:AA     L:AA    BR:AA
CB:01    NB:01    EP:00    XP:00    YP:00
SC:I1 I0 U D N V C Z   CC:F3 F2 F1 F0
    1  1 0 0 0 0 0 0       1  1  1  1
>
```

(4) During log output

If a command execution result is being output to a log file by the log command, the register values are displayed in the [Command] window even if the [Register] window is opened and are also output to the log file.

GUI utility

[View | Register] menu item

When this menu item is selected, the [Register] window opens or becomes active and displays the current register contents.

rs (register set)

Function

This command modifies the register values.

Format

(1) >rs <register> <value>↵ **(direct input mode)**

(2) >rs↵ **(guidance mode)**

```
  PC = Old value  : <value>↵
  SP = Old value  : <value>↵
   IX = Old value  : <value>↵
   IY = Old value  : <value>↵
    A = Old value  : <value>↵
    B = Old value  : <value>↵
   I1 = Old value  : <value>↵
   I0 = Old value  : <value>↵
    U = Old value  : <value>↵
    D = Old value  : <value>↵
    N = Old value  : <value>↵
    V = Old value  : <value>↵
    C = Old value  : <value>↵
    Z = Old value  : <value>↵
  HL = Old value  : <value>↵
  BR = Old value  : <value>↵
  CB = Old value  : <value>↵
  EP = Old value  : <value>↵
  XP = Old value  : <value>↵
  YP = Old value  : <value>↵
>
```

<register>: Register name (PC, SP, IX, IY, A, B, HL, BR, CB, EP, XP, YP, SC, I1, I0, U, D, N, V, Z, C)
<value>: Value to be set to the register; hexadecimal

Examples

Format (1)

```
>rs SC 0↵                  ... Resets all the flags in the SC register.
```

Format (2)

```
>rs↵
 PC=02ae : 180↵
 SP=aaaa : f0ff↵
 IX=aaaa : f000↵
 IY=aaaa : f000↵
  A=  aa : 0↵
  B=  aa : 0↵
 HL=aaaa : 0↵
 BR=  aa : 0↵
 I1=   0 : 1↵
 I0=   0 : 1↵
  U=   0 : ↵
  D=   0 : ↵
  N=   0 : ↵
  V=   0 : ↵
  C=   0 : ↵
  Z=   0 : ↵
 CB=  01 : ↵
 EP=  00 : ↵
 XP=  00 : ↵
 YP=  00 : ↵
>
```

When a register is modified, the [Register] window is updated to show the contents you have input. If you input "q↵" to stop entering in the middle, the contents input up to that time are updated.

Notes

- An error results if you input a value exceeding the register's bit width.
- An error results if you input an illegal register name in direct input mode.
- In guidance mode, the following keyboard inputs have special meaning:
  "q↵" … Command is terminated. (finish inputting and start execution)
  "^↵" … Return to previous register.
  "↵" … Input is skipped. (keep current value)

GUI utility

[Register] window

The [Register] window allows direct modification of data. Click the [Register] window, select the displayed data to be modified and enter a value then press [Enter].

13.9.5 Program Execution

g (go)

Function

This command executes the target program from the current PC address or specified address.

Format

>g [<address>]↵   (direct input mode)

<address>: Break address; hexadecimal or symbol (IEEE-695 format only)
Condition: 0 ≤ address ≤ last program memory address

Operation

(1) Program execution

If <address> is not specified, the target program is executed from the address indicated by the PC. If <address> is specified, the target program is executed from the specified address. Program execution is continued until it is made to break for one,œf the following causes:

- Break conditions set by a break set up command are met.
- A break signal is input to the ICE BRKIN pin.
- The [Key Break] button is clicked, the [Run | Stop] menu command is selected or the [Esc] key is pressed.
- A program execution error is detected.

If a break address is specified, the program execution will be suspended before executing the instruction at the specified address.

`>g 1a0↵` ... Executes the program from the current PC address to address 0x1a0.

When program execution breaks, the system stands by waiting for a command input after displaying the number of executed cycles / execution time. When you hit the [Enter] key here, program execution is resumed beginning with the break address. The break address setting is also valid.

(2) Window display by program execution

The [Source] window is updated after a break in such a way that the break address is displayed within the window.
If the [Trace] window is opened, the display contents are cleared as the program is executed. It is updated with the new trace information after a break.
If the [Dump] or [Register] window is opened, the display contents are updated after a break.
If the [Watch] window is set in short-break mode using the [Run | Setting...] menu item, its display contents are updated in the specified cycles.

(3) Display during log mode

If the program is executed after turning on the log mode, the same contents as when executing the rd command are displayed in the [Command] window after the number of executed cycles and execution time are displayed due to a break.

Example:

```
>g
 BUS CYCLE : 86519
 Mode L    : 004s   036ms  943us
OK!
PC:0618  SP:F7FE  IX:21F8  IY:F1E4
 B:01     A:05     H:F1     L:E4    BR:F0
CB:01    NB:01    EP:00    XP:04    YP:00
SC:I1 I0 U D N V C Z   CC:F3 F2 F1 F0
    0  0 0 0 0 0 0 0       0  0  0  0
>
```

When a break occurs, the same display appears as when data is displayed by the rd command.

(4) Execution cycle counter

When the target program execution is suspended, the debugger displays the number of executed cycles and execution time in the [Command] window. (Refer to Section 13.8.4 for details.)
The execution cycle counter is reset each time the g command is issued.

Notes

- If a break condition is met, program execution is suspended and the PC will be set to the program address at the breakpoint.
- The address you specified must be within the range of the program memory area available with each microcomputer model.
  An error results if the limit is exceeded or the input one is not a hexadecimal number or a valid symbol.

GUI utility

[Run | Go] menu item, [Go] button

When this menu item or button is selected, the g command without break address specification is executed.

→ *[Go] button*

[Run | Go to Cursor] menu item, [Go to Cursor] button

When this menu item or button is selected after placing the cursor to the temporary break address line in the [Source] window, the g command with a break address is executed. The program execution will be suspended before executing the address at the cursor position.

→| *[Go to Cursor] button*

gr (go after reset CPU)

Function

This command executes the target program from the boot address after resetting the CPU.

Format

>gr [<address>]↵ (direct input mode)

<address>: Break address; hexadecimal or symbol (IEEE-695 format only)
Condition: 0 ≤ address ≤ last program memory address

Operation

This command resets the CPU before executing the program. This causes the PC to be set at the boot address, from which the command starts executing the program.
Once the program starts executing, the command operates in the same way as the g command, except that the gr command does not support the function for restarting execution by hitting the [Enter] key. Refer to the explanation of the g command for more information.

Note

If a break condition is met, program execution is suspended and the PC will be set to the program address at the breakpoint.

GUI utility

[Run | Go after Reset] menu item, [Go after Reset] button

When this menu item or button is selected, the gr command is executed.

[Go after Reset] button

S (step)

Function

This command single-steps the target program from the current PC position by executing one instruction at a time.

Format

>s [<step>]↵ **(direct input mode)**

<step>: Number of steps to be executed; decimal (default is 1)
Condition: 0 ≤ step ≤ 65,535

Operation

(1) Step execution

If the <step> is omitted, only the program step at the address indicated by the PC is executed, otherwise the specified number of program steps is executed from the address indicated by the PC.

```
>s↵        ...Executes one step at the current PC address.
>s 20↵     ...Executes 20 steps from the current PC address.
```

The program execution is suspended by the following cause even before the specified number of steps is completed.

- The [Key Break] button is clicked or the [Esc] key is pressed

After each step is completed, the register contents in the [Register] window are updated. If the [Register] window is closed, the register contents are displayed in the [Command] window same as executing the rd command.
When program execution is completed by stepping through instructions, the system stands by waiting for command input. If you hit the [Enter] key here, the system single-steps the program in the same way again.

(2) HALT and SLEEP states and interrupts

When the halt or slp instruction is executed, the CPU is placed in standby mode. An interrupt is required to clear this mode. The debugger has a mode to enable or disable an external interrupt for use in a single-step operation.

	Enable mode	Disable mode
External interrupt	Interrupt is processed.	Interrupt is not processed.
halt and slp instructions	Executed as the halt instruction. Processing is continued by an external interrupt or clicking on the [Key Break] button.	The halt and slp instructions are replaced with a nop instruction as the instruction is executed.

In the initial settings, the debugger is set to the interrupt disable mode.
The interrupt enable mode can be set using the [Run | Setting...] menu item.

(3) Execution cycle counter

After the last step is completed, the debugger displays the number of executed cycles and execution time in the [Command] window. (Refer to Section 13.8.4 for details.)
The execution cycle counter is reset each time the s command is issued.

(4) During log mode

If the program is single-stepped after turning on the log mode, the same contents as when executing the rd command are displayed in the [Command] window after the last step is completed.

Notes

- The step count must be specified within the range of 0 to 65,535. An error results if the limit is exceeded.
- If the [Dump] window is opened, its display contents are updated after the execution.
- The program will not break even if the break condition set by a command is met while this command is processed.

GUI utility

[Run | Step] menu item, [Step] button

When this menu item or button is selected, the s command without step count is executed.

[Step] button

n (next)

Function

This command single-steps the target program from the current PC position by executing one instruction at a time.

Format

>n [<step>]↵　　　　　**(direct input mode)**

<step>: Number of steps to be executed; decimal (default is 1)
Condition: 0 ≤ step ≤ 65,535

Operation

This command basically operates in the same way as the s command.
However, the call instructions, including all subroutines until control returns to the next address, are executed as one step.

Notes

- The step count must be specified within the range of 0 to 65,535. An error results if the limit is exceeded.
- If the [Dump] window is opened, its display contents are updated after the execution.
- The program will not break even if the break condition set by a command is met while this command is processed.

GUI utility

[Run | Next] menu item, [Next] button

When this menu item or button is selected, the n command without step count is executed.

[Next] button

se (step exit)

Function

This command single-steps the target program from the current PC position and stops execution after exiting from the current function or subroutine.

Format

```
>se↵                (direct input mode)
```

Operation

The target program starts from the current PC address in single-stepping and stops immediately after it returns to the caller routine.

Notes

- Do not execute the se command in the main (top-level) routine.
- If the [Dump] window is opened, its display contents are updated after the execution.
- During a single-step operation, the program will not break even if the break condition set by a command is met.

GUI utility

[Run | Step Exit] menu item, [Step Exit] button

When this menu item or button is selected, the se command is executed.

[Step Exit] button

13.9.6 CPU Reset

rst (reset CPU)

Function

This command resets the CPU.

Format

>rst↵ (direct input mode)

Notes

- The registers and flags are set as follows:

PC:	Reset exception processing loads the reset vector stored in bank 0, 000000H–000001H into the PC.
SP, IX, IY:	0xAAAA
B, A, H, L, BR:	0xAA
CB, NB:	0x01
EP, XP, YP:	0x00
SC:	0b11000000
CC:	0b1111

The internal RAM and external RAM are not initialized at initial reset.
The respectively stipulated initializations are done for internal peripheral circuits.

* Reset exception processing loads the preset values stored in 0 bank, 000000H–000001H into the PC. At the same time, 01H of the NB initial value is loaded into CB.

- If the [Source] window is open, the window is redisplayed beginning with the boot address. If the [Register] window is open, the window is redisplayed with the above contents.
- The debug status, such as memory contents, breaks, and trace, is not reset.

GUI utility

[Run | Reset CPU] menu item, [Reset] button

When this menu item or button is selected, the rst command is executed.

[Reset] button

13.9.7 Break

bp (software breakpoint set)

Function

This command sets or clears software breakpoints at addresses where program execution is halted. When a program fetches an instruction at any valid software breakpoint that has been set in a 1 MB active break area, a break occurs immediately before that instruction is executed.

Format

>bp [<option>] <address>↵ (direct input mode)

<option>:	Specify to clear, enable or disable breakpoints - Clear breakpoint + Enable breakpoint (default) _ Disable breakpoint
<address>:	Break address; hexadecimal or symbol (IEEE-695 format only)
Condition:	0 ≤ address ≤ last program memory address (0x7fffff)

Examples

`>bp 200↵`	... Sets address 0x200 as a breakpoint.
`>bp _ 200↵`	... Disables the breakpoint at address 0x200.
`>bp - 200↵`	... Clears the breakpoint at address 0x200.

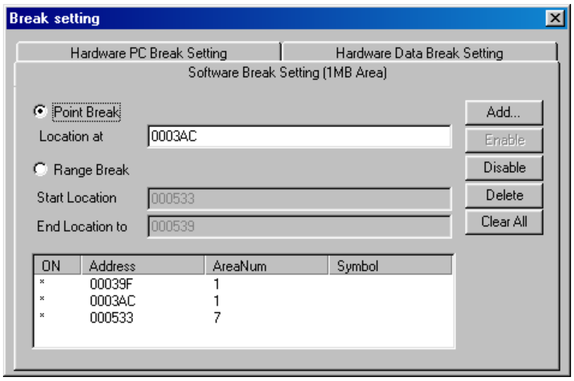
Notes

- If any address outside the 1 MB active break area set as the debugger's operating environment is specified, no breaks can occur at that address, although the address is registered as an invalid breakpoint. The 8 MB of code space is divided into eight 1 MB active break areas, one of which can be selected as a break option (by using [Break | Setting...]). At debugger startup, a 1 MB area from 0x0 to 0x0fffff is automatically selected as the active break area.
- Up to a total of 64 breakpoints can be set. Any attempt to exceed this limit prompts a warning.
- The addresses must be specified within the range of the program memory area available for each microcomputer model.
  An error results if the limit is exceeded or the input one is not a hexadecimal number or a valid symbol.
- Any attempt to set an address again that has already been set as a breakpoint will prompt a warning.
- Any attempt to clear an address where no breakpoints are set will result in an error being assumed.
- For a breakpoint, specify the start address of an instruction at which you want the program to break. If an intermediate address of that instruction is specified, no breaks can occur.
- No breakpoints can be set individually in a software break area set by the bpa command (because all addresses in that area already have breakpoints set). Any attempt to set a breakpoint at any address in that area will result in an error being assumed.
- When a program or parameter file is loaded, the contents of all breaks set are cleared.

GUI utility

[Break | Breakpoint Setting] menu item

Selecting this menu command displays a dialog box for setting or clearing breakpoints. Before performing any operation described below, select the [Software Break Setting (1MB Area)] tab.



Break setting
Hardware PC Break Setting
Hardware Data Break Setting
Software Break Setting (1MB Area)
Add...
Enable
Disable
Delete
Clear All

● Point Break
Location at: 0003AC

○ Range Break
Start Location: 000533
End Location to: 000539

ON	Address	AreaNum	Symbol
*	00039F	1	
*	0003AC	1	
*	000533	7	

To set a software breakpoint, select the [Point Break] radio button and enter an address in the [Location at] text box. Then click the [Add] button to register the address you entered as a valid breakpoint. Up to 64 breakpoints can be added to the list. Exceeding this limit prompts a warning. In such case, delete the unnecessary breakpoints before adding a new one.

To disable a valid breakpoint (whose address is preceded by an asterisk (*) in the list), select that address from the list (by clicking the ON part), then click the [Disable] button. The asterisk disappears and the breakpoint is disabled.

To enable an invalid breakpoint, select the address of that breakpoint from the list, then click the [Enable] button. The address is marked with an asterisk (*) to indicate that a breakpoint is enabled at that address.

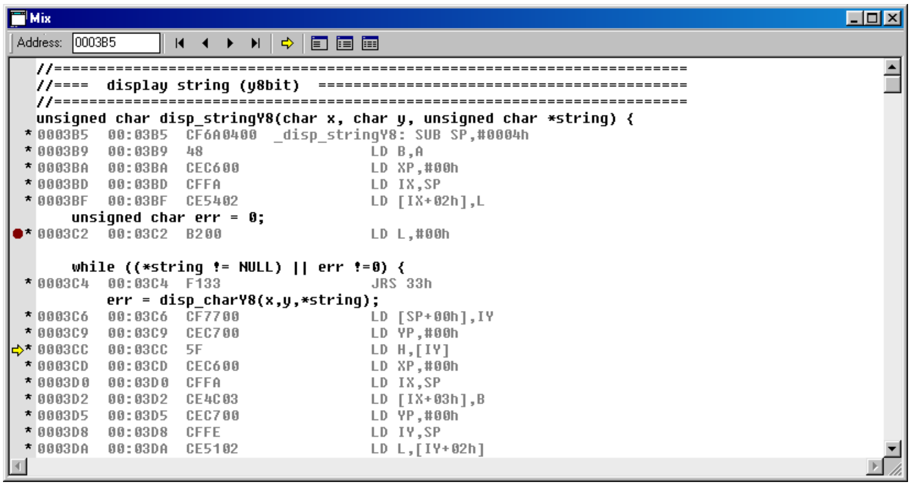
To clear a breakpoint, select the address of that breakpoint from the list, then click the [Delete] button. The [Clear All] button allows you to clear all breakpoints that have been set, including those in a software break area.

[Break] button

When this button is clicked after placing the cursor to a line in the [Source] window, the address at the cursor position is set as a breakpoint. If the address has been set as a breakpoint, this button clears the breakpoint.

[Break] button

The set breakpoints are marked with a ● at the beginning of the address lines in the [Source] window.



```
  //======================================================================
  //====  display string (y8bit)  ========================================
  //======================================================================
  unsigned char disp_stringY8(char x, char y, unsigned char *string) {
 *0003B5  00:03B5  CF6A0400  _disp_stringY8: SUB SP,#0004h
 *0003B9  00:03B9  48                        LD B,A
 *0003BA  00:03BA  CEC600                    LD XP,#00h
 *0003BD  00:03BD  CFFA                      LD IX,SP
 *0003BF  00:03BF  CE5402                    LD [IX+02h],L
      unsigned char err = 0;
●*0003C2  00:03C2  B200                      LD L,#00h

      while ((*string != NULL) || err !=0) {
 *0003C4  00:03C4  F133                      JRS 33h
          err = disp_charY8(x,y,*string);
 *0003C6  00:03C6  CF7700                    LD [SP+00h],IY
 *0003C9  00:03C9  CEC700                    LD YP,#00h
⇨*0003CC  00:03CC  5F                        LD H,[IY]
 *0003CD  00:03CD  CEC600                    LD XP,#00h
 *0003D0  00:03D0  CFFA                      LD IX,SP
 *0003D2  00:03D2  CE4C03                    LD [IX+03h],B
 *0003D5  00:03D5  CEC700                    LD YP,#00h
 *0003D8  00:03D8  CFFE                      LD IY,SP
 *0003DA  00:03DA  CE5102                    LD L,[IY+02h]
```

bpa (software area breakpoint set)

Function

This command sets a software break area or an address range in which program execution is halted. When the program fetches an instruction in a software break area that has been set in a 1 MB active break area, a break occurs immediately before that instruction is executed.

Format

(1) >bpa <address1> <address2>↵ (direct input mode)

(2) >bpa - <address1>↵ (direct input mode)

<address1>: Start address of break area; hexadecimal or symbol (IEEE-695 format only)
<address2>: End address of break area; hexadecimal or symbol (IEEE-695 format only)
Condition: 0 ≤ address1 ≤ address2 ≤ last program memory address (0x7fffff)

Examples

Format (1)

```
>bpa 100 1ff↵
```

... Sets the address range from 0x0100 to 0x01ff as software break area.

Format (2)

```
>bpa - 100↵
```

... Clears the software break area beginning with address 0x0100.

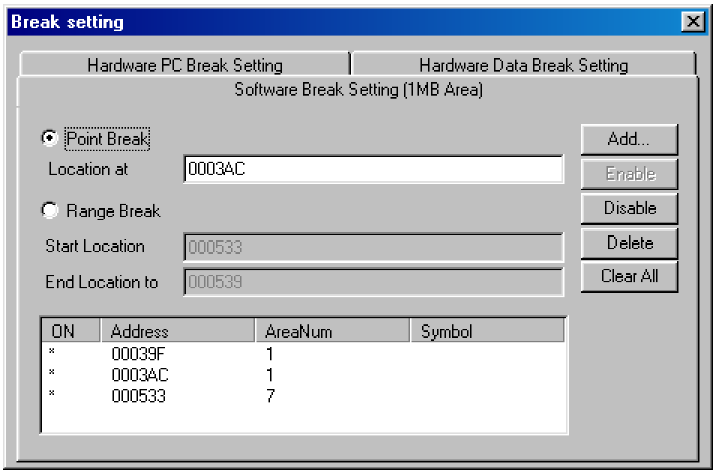
Notes

- Specifying any address outside the 1 MB active break area set as the debugger's operating environment results in an error being assumed. The 8 MB of code space is divided into eight 1 MB active break areas, one of which can be selected as a break option (by using [Break | Setting...]). At debugger startup, a 1 MB area from 0x0 to 0x0fffff is automatically selected as the active break area.
- Only one software break area can be set at a time. Before a new software break area can be set, the previously set area must be cleared.
- The addresses must be specified within the range of the program memory area available for each microcomputer model.
  An error results if the limit is exceeded or the input one is not a hexadecimal number or a valid symbol.
- Any attempt to set an area that contains an address already set individually as a breakpoint prompts a warning. Similarly, no breakpoints can be set individually in a software break area that has been set by the bpa command.
- For a break area's start and end addresses, specify the start address of an instruction at which you want the program to break. If an intermediate address of that instruction is specified, no breaks can occur.
- When a program or parameter file is loaded, the contents of all breaks set are cleared.

GUI utility

[Break | Breakpoint Setting] menu item

Selecting this menu command displays a dialog box for setting or clearing breakpoints. Before performing any operation described below, select the [Software Break Setting (1MB Area)] tab.


Break setting
Hardware PC Break Setting
Hardware Data Break Setting
Software Break Setting (1MB Area)
Add...
Enable
Disable
Delete
Clear All

● Point Break

Location at 0003AC

○ Range Break

Start Location 000533

End Location to 000539

ON	Address	AreaNum	Symbol
*	00039F	1	
*	0003AC	1	
*	000533	7	

To set a software break area, select the [Range Break] radio button, then enter the start and end addresses of that area in the [Start Location] and [End Location to] text boxes, respectively. Then click the [Add] button to register the area you entered as a valid software break area. All addresses in that area are assumed to have breakpoints set. The start address of the area is shown in the Address column of the list, and the area size (in bytes) is shown in the AreaNum column. Setting a new area with a software break area already registered prompts a warning. In such case, delete the registered software break area before setting a new one. Also note that because only one software break area can exist at a time, any area that contains an address already registered as a breakpoint cannot be set as a software break area.

To disable a valid breakpoint (whose address is preceded by an asterisk (*) in the list), select that address from the list (by clicking the ON part), then click the [Disable] button. The asterisk disappears and the breakpoint is disabled.

To enable an invalid breakpoint, select the address of that breakpoint from the list, then click the [Enable] button. The address is marked with an asterisk (*) to indicate that a breakpoint is enabled at that address.

To clear a breakpoint, select the address of that breakpoint from the list, then click the [Delete] button. The [Clear All] button allows you to clear all breakpoints that have been set, including those in a software break area.

bpr / bc / bpc (software breakpoint clear)

Function

This command clears the specified breakpoints or software break area that have been set.

Format

(1) >bpr↵ **(direct input mode)**

(2) >bc [<address>]↵ **(direct input mode)**

(3) >bpc [<address>]↵ **(direct input mode)**

<address>: Break address; hexadecimal or symbol (IEEE-695 format only)

Examples

`>bc 200↵`	... Clears a breakpoint at address 0x0200. When a break area is set from address0x0200, the break area is cleared.
`>bpr↵`	... Clears all breakpoints and break area.
`>bc↵`	... Clears all breakpoints and break area.
`>bpc↵`	... Clears all breakpoints and break area.

Notes

- The bc and bpc commands have the same functions.
- If no address parameter is specified for the bc or bpc command, it works the same as the bpr command and all the breakpoints and break area that have been set are cleared.
- An error results if an address that is not set at a breakpoint is specified.

GUI utility

[Break | Breakpoint Setting] menu item

When this menu item is selected, a dialog box appears for setting / clearing breakpoints. (See the bp command.)

[Break] button

When this button is clicked after placing the cursor to a break address line in the [Source] window, the breakpoint is cleared. If the address has not been set as a breakpoint, this button sets a new breakpoint at the address.

[Break] button

bas (sequential break setting)

Function

This command sets the sequential break mode.

Format

>bas[<mode>]↵ (direct input mode)

<mode>: Sequential break mode number
0 Independent break mode
1 BA3 count break mode
2 BA2&BA3 sequential break mode
3 BA1–BA3 sequential break mode

Examples

```
>bas3↵                          ... Sets BA1–BA3 sequential break mode.
>bas↵                           ... If <mode> is omitted, the current mode is displayed.
 Independent Break Mode
>
```

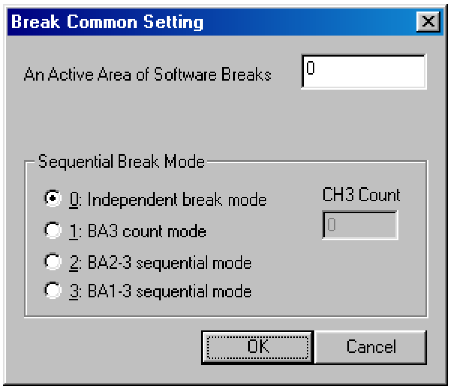
Notes

- Do not insert any space between "bas" and <mode>.
- See the ba command for the operation in each mode and setting each break channel.
- The debugger is configured to independent break mode at the time it starts up.
- The set break conditions are all cleared when a program or a parameter file is loaded.

GUI utility

[Break | Setting...] menu item

When this menu item is selected, a dialog box appears for selecting break options.



Break Common Setting

An Active Area of Software Breaks: 0

Sequential Break Mode

● 0: Independent break mode
○ 1: BA3 count mode
○ 2: BA2-3 sequential mode
○ 3: BA1-3 sequential mode

CH3 Count: 0

Select a sequential break mode using the [Sequential Break Mode] radio buttons.
The [CH3 Count] text box is enabled to enter a BA3 execution count value when a radio button for the mode that uses the BA3 counter is selected.

ba (hardware breakpoint set)

Function

This command sets or clears hardware breakpoints at which the program is halted when it executes a specified sequence. The breakpoints set on each channel and the execution count set on CH3 are enabled or disabled according to the sequential break mode set by the bas command.
Break occurrence conditions in each sequential break mode are described below.

1. Independent break mode (BAS0) (default)
   In this mode, program execution is made to break when the program fetches an instruction at a breakpoint set on each channel. The execution count specified for CH3 (BA3) is not effective.
2. BA3 count mode (BAS1)
   In this mode, the count function of CH3 (BA3) is effective. Program execution is made to break when the program has fetched the instruction as many times as set by <count> at the breakpoint set on CH3. Breakpoints set on CH1 and CH2 are not effective.
3. BA2&BA3 sequential mode (BAS2)
   In this mode, program execution is made to break when the program has fetched the instruction as many times as set by <count> at the breakpoint set on CH3 after executing the instruction more than once at the breakpoint set on CH2. The breakpoint set on CH1 is not effective.
4. BA1–BA3 sequential mode (BAS3)
   In this mode, program execution is made to break when the program has fetched the instruction as many times as set by <count> at the breakpoint set on CH3 after executing the instructions more than once in that order at the breakpoints set on CH1 and CH2.

Format

(1) >ba<channel> <address> [<count>]↵ (direct input mode)

(2) >ba<channel> <option>↵ (direct input mode)

<channel>: Break channel number (1–3)
<address>: Break address; hexadecimal or symbol (IEEE-695 format only)
<count>: CH3 count value; decimal (default: 1)
<option>: Specify to clear, enable or disable breakpoints
- Clear breakpoint
+ Enable breakpoint (default)
_ Disable breakpoint
Condition: 0 ≤ address1 ≤ last program memory address (0x7fffff), 0 ≤ count ≤ 4095

Examples

```
>bas0↵
>ba1 200↵
>
```

In this example, independent break mode is selected, with the CH3 breakpoint set at address 0x0200. Program execution is made to break when the program fetches the instruction at address 0x0200. This breakpoint is effective even when set outside a 1-MB active break area.

```
>ba1 _↵
>
```

In this example, the breakpoint on CH1 is disabled.

```
>bas2↵
>ba2 200↵
>ba3 300 2↵
>
```

In this example, BA2&BA3 sequential mode is selected, with the CH2 and CH3 breakpoints set at addresses 0x0200 and 0x0300, respectively. Also, the CH3 counter is set to 2. When the program executes the instruction at 0x0300 once and fetches the instruction at 0x0300 again after executing the instruction at 0x0200 once or more, a break occurs before that instruction is executed. These breakpoints are effective even when set outside a 1 MB active break area.

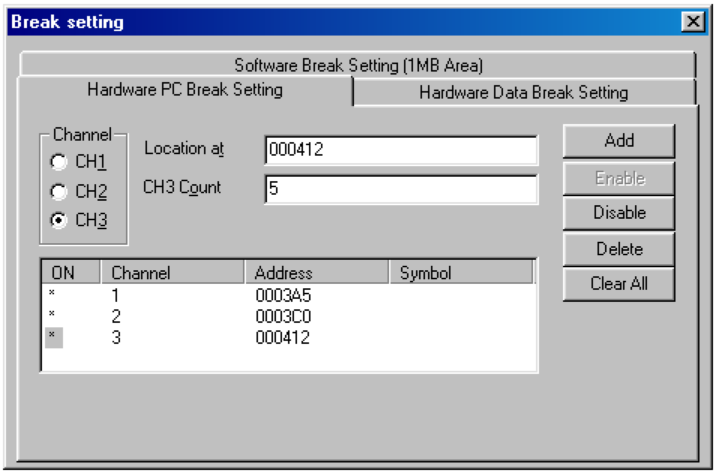
Notes

- Do not insert a space between "ba" and <channel>.
- If count specification is omitted when setting CH3, the counter is set to 1 by default. Specifying a count of 0 sets the counter to 4,096 by default.
- Even in independent break mode, a execution count for CH3 can be set without causing an error, but the count is not effective.
- The addresses must be specified within the range of the program memory area available for each microcomputer model.
  An error results if the limit is exceeded or the input one is not a hexadecimal number or a valid symbol.
- Any attempt to set the same address again that has already been set as a breakpoint will prompt a warning.
- Any attempt to clear an address at which no breakpoints are set will result in an error being assumed.
- For a breakpoint, specify the start address of an instruction at which you want the program to break. If an intermediate address of that instruction is specified, no breaks can occur.
- When a program or parameter file is loaded, the contents of all breaks set are cleared.

GUI utility

[Break | Breakpoint Setting] menu item

Selecting this menu command displays a dialog box for setting or clearing breakpoints. Before performing any operation described below, select the [Hardware PC Break Setting] tab.


Break setting
Software Break Setting (1MB Area)
Hardware PC Break Setting
Hardware Data Break Setting
Add
Enable
Disable
Delete
Clear All

Channel
○ CH1
○ CH2
● CH3

Location at: 000412

CH3 Count: 5

ON	Channel	Address	Symbol
*	1	0003A5	
*	2	0003C0	
*	3	000412	

Use the radio buttons to select the channel on which you want to set an address, then enter the desired address in the [Location at] text box. To specify an execution count on BA3, enter a hexadecimal number for the desired count in the [CH3 Count] text box. If a count was set from the [Break Common Setting] dialog box, the value you entered is reflected in this text box.

Click the [Add] button to register the address you've set as a valid breakpoint. Each channel can have only one address set. Setting a new address on a channel for which an address is already set overwrites the existing address. Any attempt to set an address already registered as a hardware PC breakpoint prompts a warning.

To disable a valid breakpoint (whose address is preceded by an asterisk (*) in the list), select that address from the list (by clicking the ON part), then click the [Disable] button. The asterisk disappears and the breakpoint is disabled.

To enable an invalid breakpoint, select the address of that breakpoint from the list, then click the [Enable] button. The address is marked with an asterisk (*) to indicate that a breakpoint is enabled at that address.

To clear a breakpoint, select the address of that breakpoint from the list, then click the [Delete] button. The [Clear All] button allows you to clear all breakpoints that have been set.

bar (hardware breakpoint clear)

Function

This command clears the hardware breakpoints that have been set and the CH3 counter.

Format

>bar↵ **(direct input mode)**

Example

```
>bar↵                    ... Clears all the hardware breakpoints set.
```

Note

An error results if no hardware breakpoint is set.

GUI utility

[Break | Breakpoint Setting] menu item

When this menu item is selected, a dialog box appears for setting/clearing breakpoints. (See the ba command.)

bd (hardware data breakpoint set)

Function

This command sets or clears hardware data breaks at which the program is halted when it performs a memory access under the specified conditions.
Data break conditions can be set individually on each of four channels. Data breaks on each channel can be individually enabled or disabled.
The following data break conditions can be set.

1. Address condition
   Specify this condition to cause the program to break when it accesses a particular address.
2. Data condition
   Specify this condition to cause the program to break when it reads or writes a particular byte of data from or to memory. Specifying data in other than decimal notation allows any data bits to be masked (excluded from data conditions) when marked with an asterisk (*).
3. Read/write condition
   Specify whether you want the program to break in a read or a write cycle. If this specification is omitted, a break occurs in both cycles.

These three conditions can be specified in any desired combination. In such case, a break occurs when the program accesses memory to satisfy all set conditions.

Format

(1) >bd<channel> [A=<address>] [D=<data>] [{R|W}]↵ (direct input mode)

(2) >bd<channel> <option>↵ (direct input mode)

<channel>: Data break channel number (0–3)
<address>: Memory address; hexadecimal or symbol (IEEE-695 format only)
<data>: Data pattern (1 byte)
Specifying data in other than decimal notation allows any bits to be masked when marked with an asterisk (*).
R|W: R for break in a read cycle
W for break in a write cycle
If this specification is omitted, a break occurs in both read and write cycles.
<option>: Specify whether to clear, enable, or disable settings.
- Clear break conditions
+ Enable break conditions (default)
_ Disable break conditions
Condition: 0 ≤ address ≤ 0xffffff, 0 ≤ data ≤ 0xff

Examples

```
>bd0 A=f100 D=1*******B R↵
>
```

In this example, data break is set on CH0. A break occurs when the program reads data whose MSB = 1 from address 0xf100. This address is effective even when set outside a 1 MB active break area.

```
>bd0 _↵
>
```

In this example, break conditions are disabled on CH0.

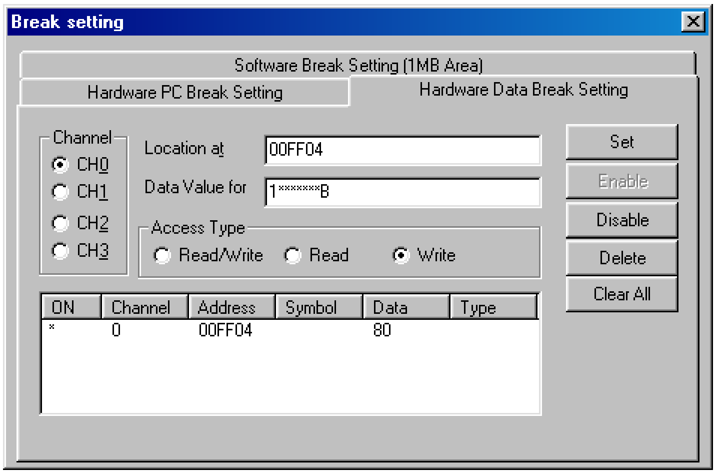
Notes

- Do not insert a space between "bd" and <channel>.
- The addresses must be specified within the range of the memory area available for each microcomputer model.
  An error results if the limit is exceeded or the input one is not a hexadecimal number or a valid symbol.
- Any attempt to clear a channel on which no break conditions are set results in an error being assumed.
- When a program or parameter file is loaded, the contents of all breaks set are cleared.

GUI utility

[Break | Breakpoint Setting] menu item

Selecting this menu command displays a dialog box for setting or clearing breakpoints. Before performing any operation described below, select (click) the [Hardware Data Break Setting] tab.


Break setting
Software Break Setting (1MB Area)
Hardware PC Break Setting
Hardware Data Break Setting
Set
Enable
Disable
Delete
Clear All

Channel: ● CH0 ○ CH1 ○ CH2 ○ CH3

Location at: 00FF04

Data Value for: 1*******B

Access Type: ○ Read/Write ○ Read ● Write

ON	Channel	Address	Symbol	Data	Type
*	0	00FF04		80	

Use the radio buttons to select the channel on which you want to set break conditions, then enter an address in the [Location at] text box and data in the [Data Value for] text box (optional). Use the radio buttons to select the desired read / write condition, then click the [Set] button to register what you've entered as valid break conditions. Setting a new condition on a channel for which conditions are already set overwrites the existing conditions.

To disable valid break conditions on a channel (preceded by an asterisk (*) in the list), select that channel from the list (by clicking the ON part), then click the [Disable] button. The asterisk disappears and break conditions on the channel are disabled.

To enable invalid break conditions on any channel, select that channel from the list, then click the [Enable] button. The channel is marked with an asterisk (*) to indicate that break conditions are enabled on the channel.

To clear break conditions on any channel, select that channel from the list, then click the [Delete] button.

The [Clear All] button allows you to clear all break conditions that have been set.

bdr (hardware data breakpoint clear)

Function

This command clears the hardware data break conditions that have been set.

Format

>bdr↵ **(direct input mode)**

Example

```
>bdr↵                    ... Clears all the hardware  data break conditions set.
```

Note

An error results if no hardware data break condition is set.

GUI utility

[Break | Breakpoint Setting] menu item

When this menu item is selected, a dialog box appears for setting/clearing breakpoints. (See the bd command.)

bl (breakpoint list)

Function

This command lists the current setting of all break conditions.

Format

>bl↵ **(direct input mode)**

Example

```
>bl↵
PC break:
Software Break:
    1:  0005fa ENABLE
    2:  000618 ENABLE
    3:  00062d ENABLE
Area Break:
 000100 - 0001ff ENABLE
Hardware Break:
    1:  CH1 000728 ENABLE
    2:  CH2 000742 ENABLE
    3:  CH3 000786 ENABLE
Sequential Break Mode:
 BA1 - BA3 Sequential Mode : Count(3)
Data break:
 CH0 DATA: 1*******   R/W: R   R/W AREA: 00F010 ENABLE
>
```

GUI utility

[Break | Break List] menu item

When this menu item is selected, the bl command is executed.

bac (break all clear)

Function

This command clears all break conditions set by the bp, bpa, bas, ba and / or bd commands.

Format

>bac↵ **(direct input mode)**

GUI utility

[Break | Break All Clear] menu item, [Break All Clear] button

When this menu item or button is selected, the bac command is executed.

[Break All Clear] button

13.9.8 Program Display

u (unassemble)

Function

This command displays the program in the [Source] window after disassembling it. The display contents are as follows:

- • Physical memory address
- • Logical memory address
- • Object code
- • Unassembled contents of the program

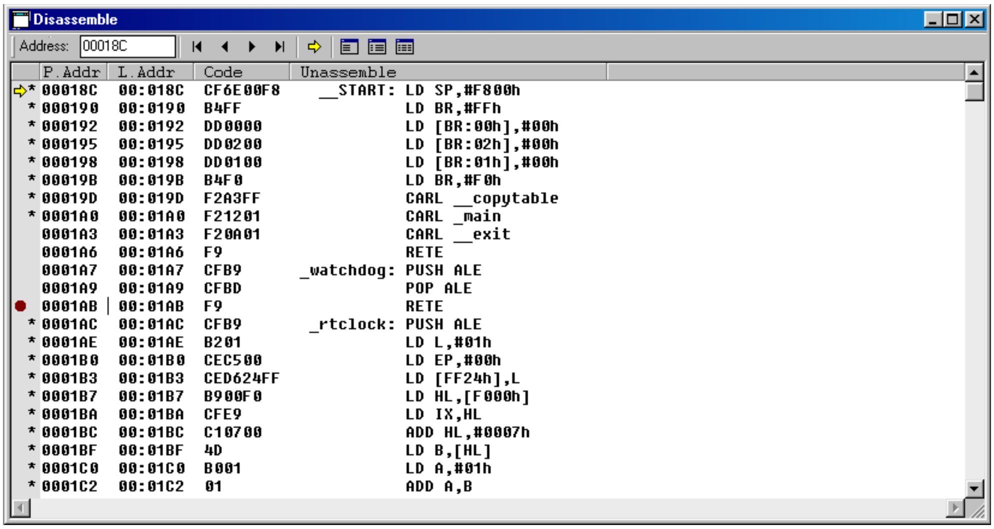
Format

>u [<address>]↵ (direct input mode)

<address>: Start address for display; hexadecimal or symbol (IEEE-695 format only)
Condition: 0 ≤ address ≤ last program memory address (0x7fffff)

Display

(1) When [Source] window is opened



```
Address: 00018C

   P.Addr   L.Addr   Code                 Unassemble
⇨* 00018C   00:018C  CF6E00F8     __START: LD SP,#F800h
 * 000190   00:0190  B4FF                  LD BR,#FFh
 * 000192   00:0192  DD0000                LD [BR:00h],#00h
 * 000195   00:0195  DD0200                LD [BR:02h],#00h
 * 000198   00:0198  DD0100                LD [BR:01h],#00h
 * 00019B   00:019B  B4F0                  LD BR,#F0h
 * 00019D   00:019D  F2A3FF                CARL __copytable
 * 0001A0   00:01A0  F21201                CARL _main
   0001A3   00:01A3  F20A01                CARL __exit
   0001A6   00:01A6  F9                    RETE
   0001A7   00:01A7  CFB9        _watchdog: PUSH ALE
   0001A9   00:01A9  CFBD                  POP ALE
●  0001AB   00:01AB  F9                    RETE
 * 0001AC   00:01AC  CFB9         _rtclock: PUSH ALE
 * 0001AE   00:01AE  B201                  LD L,#01h
 * 0001B0   00:01B0  CEC500                LD EP,#00h
 * 0001B3   00:01B3  CED624FF              LD [FF24h],L
 * 0001B7   00:01B7  B900F0                LD HL,[F000h]
 * 0001BA   00:01BA  CFE9                  LD IX,HL
 * 0001BC   00:01BC  C10700                ADD HL,#0007h
 * 0001BF   00:01BF  4D                    LD B,[HL]
 * 0001C0   00:01C0  B001                  LD A,#01h
 * 0001C2   00:01C2  01                    ADD A,B
```

If <address> is not specified, display in the [Source] window is changed to the disassemble display mode. If <address> is specified, display in the [Source] window is changed to the disassemble display mode. At the same time, code is displayed beginning with <address>.

(2) When [Source] window is closed

The 16 lines of disassembled result are displayed in the [Command] window. The system then waits for a command input.

If <address> is not specified, this display begins with the current PC. If <address> is specified, the display begins with <address>.

```
>u↵
P.ADDR  L.ADDR   CODE                 UNASSEMBLE
0002AE  00:02AE  CF6E00F8    __START: LD SP,#F800h
0002B2  00:02B2  B4FF                 LD BR,#FFh
0002B4  00:02B4  DD0000               LD [BR:00h],#00h
0002B7  00:02B7  DD020C               LD [BR:02h],#0Ch
   :       :        :                    :
0002CF  00:02CF  B200                 LD L,#00h
0002D1  00:02D1  C30000               ADD IY,#0000h
>
```

(3) During log output

If the command execution result is being output to a log file as specified by the log command, code is displayed in the [Command] window and its contents are also output to the log file.

If the [Source] window is closed, the result is displayed in the same way as in (2) above.

If the [Source] window is opened, the window is redisplayed. In this case, the same number of lines is displayed in the [Command] window as displayed in the [Source] window.

(4) Successive display

If you execute the u command after entering it from the keyboard, code can be displayed successively by entering the [Enter] key only until some other command is executed.

When you press the [Enter] key, the [Source] window is scrolled forward one screen.

When displaying code in the [Command] window, 16 lines of code following the previously displayed address are displayed (the same number of lines as displayed in the [Source] window if the u command is executed during log output).

Note

The display start address you specified must be within the range of the program memory area available with each microcomputer model.

An error results if the limit is exceeded or the input one is not a hexadecimal number or not a valid symbol.

GUI utility

[View | Source | Disassemble] menu item, [Disassemble] button

When this menu item or button is selected, the [Source] window opens or activates and displays the program from the current PC address.

[Disassemble] button

SC (source code)

Function

This command displays the contents of the program source file in the [Source] window.

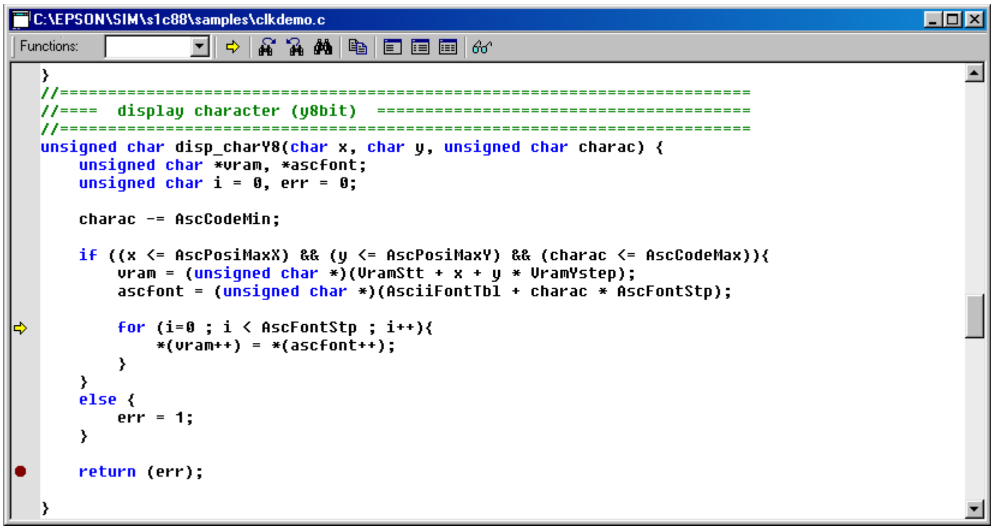
Format

>sc [<address>]↵ (direct input mode)

<address>: Start address for display; hexadecimal or symbol (IEEE-695 format only)
Condition: 0 ≤ address ≤ last program memory address (0x7fffff)

Display

(1) When [Source] window is opened



```
}
//========================================================================
//====  display character (y8bit)  =======================================
//========================================================================
unsigned char disp_charY8(char x, char y, unsigned char charac) {
    unsigned char *vram, *ascfont;
    unsigned char i = 0, err = 0;

    charac -= AscCodeMin;

    if ((x <= AscPosiMaxX) && (y <= AscPosiMaxY) && (charac <= AscCodeMax)){
        vram = (unsigned char *)(VramStt + x + y * VramYstep);
        ascfont = (unsigned char *)(AsciiFontTbl + charac * AscFontStp);

        for (i=0 ; i < AscFontStp ; i++){
            *(vram++) = *(ascfont++);
        }
    }
    else {
        err = 1;
    }

    return (err);

}
```

If <address> is not specified, display in the [Source] window is changed to the source display mode. If <address> is specified, display in the [Source] window is changed to the source display mode. At the same time, code is displayed beginning with <address>.

(2) When [Source] window is closed

The 17 lines of source code are displayed in the [Command] window. The system then waits for a command input.

If <address> is not specified, this display begins with the current PC. If <address> is specified, the display begins with <address>.

```
>sc↵
{
   #pragma asm

   GLOBAL     __START
   __START:

   ;=========================================================================
   ;===================  system initialization  =============================
   ;=========================================================================

   LD   SP,#@DOFF(__lc_es)               ; stack pointer initialize
   LD   BR,#0FFh                         ; BR register initialize to I/O area
   ;---------------  bus mode setting  --------------------------------------
                                         ; MCU & MPU mode
   LD   [BR:00h],#0
                                         ; Single Chip mode
                                         ; /CE0,/CE2,/CE3,/CE1:disenabled
>
```

(3) During log output

If the command execution result is output to a log file as specified by the log command, code is displayed in the [Command] window and its contents are also output to the log file.
If the [Source] window is closed, code is displayed in the same way as in (2) above.
If the [Source] window is open, the window is redisplayed. In this case, the same number of lines is displayed in the [Command] window as displayed in the [Source] window.

(4) Successive display

If you execute the sc command after entering it from the keyboard, code can be displayed successively by entering the [Enter] key only until some other command is executed.
When you press the [Enter] key, the [Source] window is scrolled forward one screen.
When displaying code in the [Command] window, 17 lines of code following the previously displayed address are displayed (the same number of lines as displayed in the [Source] window if the sc command is executed during log output).

Notes

- Source codes can be displayed only when an absolute object file that contains source debug information has been loaded.
- The display start address you specified must be within the range of the program memory area available with each microcomputer model.
  An error results if the limit is exceeded or the input one is not a hexadecimal number or not a valid symbol.

GUI utility

[View | Source | Source] menu item, [Source] button

When this menu item or button is selected, the [Source] window opens or activates and displays the program from the current PC address.

[Source] button

m (mix)

Function

This command displays the disassembled result of the program and the contents of the program source file in the [Source] window. The disassemble display contents are the same as the disassemble display mode.

Format

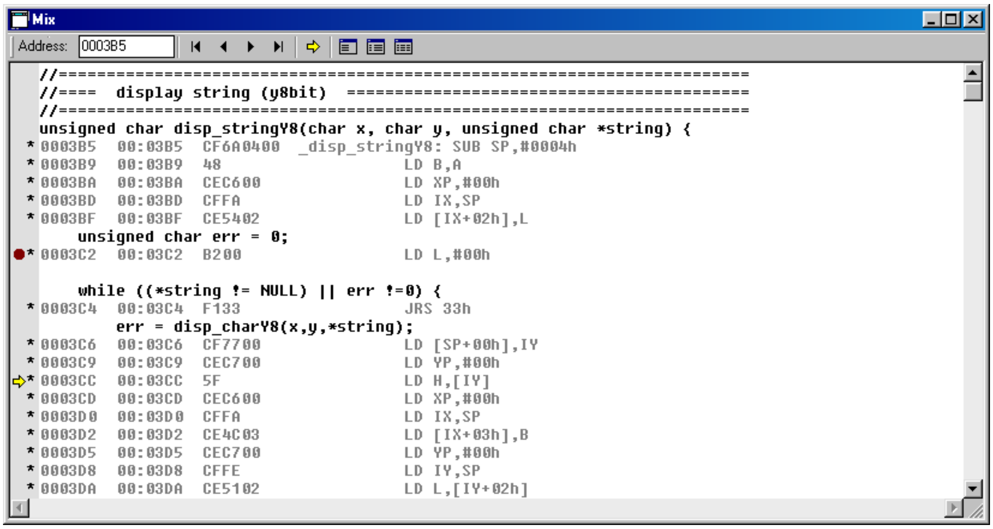
>m [<address>]↵   (direct input mode)

<address>: Start address for display; hexadecimal or symbol (IEEE-695 format only)

Condition: 0 ≤ address ≤ last program memory address (0x7fffff)

Display

(1) When [Source] window is opened



```
  //=================================================================
  //====  display string (y8bit)  ======================================
  //=================================================================
  unsigned char disp_stringY8(char x, char y, unsigned char *string) {
 *0003B5  00:03B5  CF6A0400  _disp_stringY8: SUB SP,#0004h
 *0003B9  00:03B9  48                        LD B,A
 *0003BA  00:03BA  CEC600                    LD XP,#00h
 *0003BD  00:03BD  CFFA                      LD IX,SP
 *0003BF  00:03BF  CE5402                    LD [IX+02h],L
      unsigned char err = 0;
●*0003C2  00:03C2  B200                      LD L,#00h

      while ((*string != NULL) || err !=0) {
 *0003C4  00:03C4  F133                      JRS 33h
          err = disp_charY8(x,y,*string);
 *0003C6  00:03C6  CF7700                    LD [SP+00h],IY
 *0003C9  00:03C9  CEC700                    LD YP,#00h
⇨*0003CC  00:03CC  5F                        LD H,[IY]
 *0003CD  00:03CD  CEC600                    LD XP,#00h
 *0003D0  00:03D0  CFFA                      LD IX,SP
 *0003D2  00:03D2  CE4C03                    LD [IX+03h],B
 *0003D5  00:03D5  CEC700                    LD YP,#00h
 *0003D8  00:03D8  CFFE                      LD IY,SP
 *0003DA  00:03DA  CE5102                    LD L,[IY+02h]
```

If <address> is not specified, display in the [Source] window is changed to the mix (source and disassemble) display mode. If <address> is specified, display in the [Source] window is changed to the mix display mode. At the same time, code is displayed beginning with <address>.

(2) When [Source] window is closed

The 16 lines of mix display are produced in the [Command] window. The system then waits for a command input.

If <address> is not specified, this display begins with the current PC. If <address> is specified, the display begins with <address>.

```
>m↵
_interrupt( 0x0000 )    /* Startup vector */
void _start_cpt( void )
{
0002AE  00:02AE  CF6E00F8    __START: LD SP,#F800h
0002B2  00:02B2  B4FF                 LD BR,#FFh
0002B4  00:02B4  DD0000               LD [BR:00h],#00h
0002B7  00:02B7  DD020C               LD [BR:02h],#0Ch
0002BA  00:02BA  DD0100               LD [BR:01h],#00h
0002BD  00:02BD  B4F0                 LD BR,#F0h
   :       :       :                        :
```

(3) During log output

If the command execution result is output to a log file as specified by the ***log*** command, code is displayed in the [Command] window and its contents are output to the log file also.
If the [Source] window is closed, code is displayed in the same way as in (2) above.
If the [Source] window is open, the window is redisplayed. In this case, the same number of lines is displayed in the [Command] window as displayed in the [Source] window.

(4) Successive display

If you execute the m command after entering it from the keyboard, code can be displayed successively by entering the [Enter] key only until some other command is executed.
When you press the [Enter] key, the [Source] window is scrolled forward one screen.
When displaying code in the [Command] window, 16 lines of code following the previously displayed address are displayed (the same number of lines as displayed in the [Source] window if the m command is executed during log output).

Notes

- Source codes can be displayed only when an absolute object file that contains source debug information has been loaded.
- The display start address you specified must be within the range of the program memory area available with each microcomputer model.
  An error results if the limit is exceeded or the input one is not a hexadecimal number or not a valid symbol.

GUI utility

[View | Source | Mix] menu item, [Mix] button

When this menu item or button is selected, the [Source] window opens or activates and displays the program from the current PC address.

▤ *[Mix] button*

13.9.9 Symbol Information

sy (symbol list)

Function

This command displays a list of symbols in the [Command] window.

Format

>sy [/a]↵ **(direct input mode)**

Examples

```
>sy↵
 Address  Symbol
 0004A5   __ANDXL
 0004E4   __BLCPS
 0004C6   __CMPSL
 00056B   __CMPUL
 0002CE   __DIVSI
    :        :
 000E48   _strtok
 0002C9   _watchdog
>
```

When /a is omitted, all the defined symbols are displayed in alphabetical order.

```
>sy /a↵
 Address  Symbol
 000100   __copytable
 00014A   _rtclock
 0002AE   __START
 0002AE   __start_cpt
 0002C9   _watchdog
    :        :
 00F1F2   __ungetc
 00F800   __lc_es
>
```

When /a is specified, the symbol list is sorted by address.

Note

The symbol list can only be displayed when the object file (.abs) in IEEE-695 format has been read or when the symbol file (.sy) is loaded simultaneously with the program HEX file (.psa).

GUI utility

None

w (symbol watch)

Function

This command displays the content of a specified symbol.

Format

(1) >w <symbol> [;<option>] [/a]↵ **(direct input mode)**

(2) >w↵ **(guidance mode)**

File name: <file name>↵
Function name: <function>↵
Symbol name: <symbol>↵
Format ? (B/Q/D/H) <option>↵
Display in watch window? (Y/N) {Y|N}↵
<symbol> = *Current value*
>

<symbol>:	Symbol name	
<option>:	Display format option	
	B	Binary
	Q	Octal
	D	Decimal
	H	Hexadecimal (default)
<file name>:	Source file name	
<function>:	Function name	

Examples

Format (1)

```
>w saveFlg ;B↵
saveFlg = 00000001       ... Shows the symbol value
>w saveFlg ;B /a↵        ... Shows the symbol value in the [Watch] window
>w xxx↵
No such symbol exists.   ... If the symbol cannot be found
>
```

When the /a option is specified, the symbol is registered in the watch symbol list when its name and value are displayed in the [Watch] window, and the displayed contents are automatically updated according to the [Watch] window's update mode.

Format (2)

```
>w↵
File name: calc.c↵
Function name: main↵
Symbol name: count↵
Format? (B/Q/D/H)H↵
Display in watch window? (Y/N)N↵
count = 0x00↵
>
```

To specify a global symbol, simply press the [Enter] key without entering a file name and function name.

Note

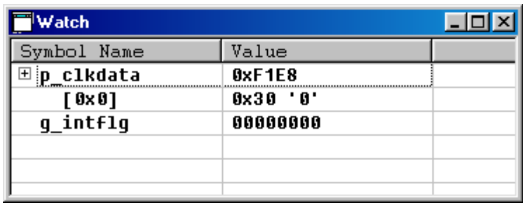
Symbol information can only be displayed using the w command when an IEEE-695 format object file (.abs) is loaded in the ICE.

GUI utility

[Watch] button (located in the [Source] window)

Select (highlight) a symbol name in the [Source] window by dragging it, then click the [Watch] button. The selected symbol is then registered in the symbol list in the [Watch] window. Once registered this way, the symbol value can be confirmed in the [Watch] window.

[Watch] button



Symbol Name	Value
⊞ p_clkdata	0xF1E8
[0x0]	0x30 '0'
g_intflg	00000000

13.9.10 Load File

lf (load file)

Function

This command loads an object file (.abs: IEEE-695 format, .psa: Motorola S2 format) and / or a function option HEX file (.fsa: Motorola S2 format) into the debugger.

Format

(1) >lf <file name>↵ **(direct input mode)**

(2) >lf↵ **(guidance mode)**

Program object file name (.ABS/.PSA) . . . ? <file name>↵
Function option file name (.FSA) . . . ? <file name>↵
OK!
>

<file name>: File name to be loaded (path can also be specified)

Examples

Format (1)

```
>lf test.abs↵
OK!
Symbol file is loaded.  ... Indicates that symbol information has been loaded.
>lf test.fsa↵
OK!
>
```

In format (1), the object file and function option file must be specified separately.

Format (2)

```
>lf↵
Program object file name(.ABS/.PSA) ... ? test.abs↵
Function option file name(.FSA) ... ? test.fsa↵
OK!
Symbol file is loaded.
>
```

In format (2), the object file and function option file can be loaded in one operation by entering both file names according to the guidance. You can skip loading one of the two files by simply pressing the [Enter] key.

Notes

- The debugger determines the type of file from the specified file name. Therefore, only files that have one of the above extensions can be loaded. Specifying other files results in an error.
- If you want to use source display and symbols when debugging a program, the object file must be in IEEE-695 format that contains debug information loaded into the debugger.
- If the [Source] window is opened when loading a file, its contents are updated. The program contents are displayed from the current PC address.
- If an error occurs when loading a file, portions of the file that have already been read will remain in the emulation memory.
- When a program file is loaded, all set breakpoints and break conditions are cleared, as are all trace information and coverage information acquired.

GUI utility

[File | Load File...] menu item, [Load File] button

When this menu item or button is selected, a dialog box appears allowing selection of an object file to be loaded.

📂 *[Load File] button*

par (load parameter file)

Function

This command loads a parameter file (.par) to set memory map information.
When a SelfFlash program address must be set, a break must be set at the end address of that program.

Format

(1) >par <file name>↵ **(direct input mode)**

(2) >par↵ **(guidance mode)**

```
File Name    . . . ? <file name>↵
>
```

<file name>: Parameter file name to be loaded (path can also be specified)

Examples

Format (1)

```
>par 88xxx.par↵
>
```

Format (2)

```
>par↵
File name   ? 88xxx.par↵
>
```

Notes

- When a parameter file is loaded, all set breakpoints and break conditions are cleared, as are all trace information and coverage information acquired.
- If the map information of the loaded parameter file is erroneous, the debugger fails to initialize the ICE and cannot run the program.

GUI utility

[File | Load Parameter File] menu item, [Load Parameter] button

When this menu item or button is selected, a dialog box appears allowing selection of a parameter file to be loaded.

[Load Parameter] button

13.9.11 Trace

td (trace data display)

Function

This command displays the trace information that has been sampled into the ICE trace memory.

Format

(1) >td [<cycle>]↵ (direct input mode)

(2) >td↵ (guidance mode)

Start index (ENTER as 0)? : <cycle>↵

(Trace data is displayed)

>

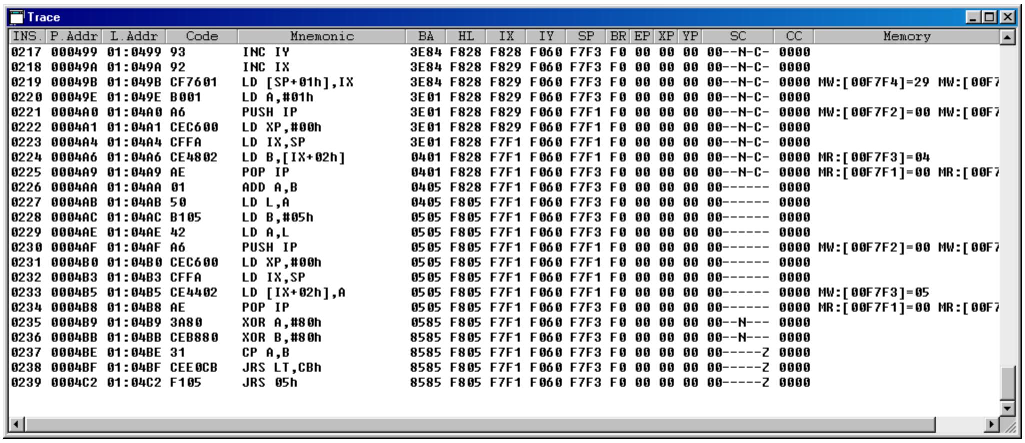
<cycle>: Start cycle number of trace data; decimal (from 0 to 8,191)

Display

The following lists the contents of trace information:

INS: CPU cycle number (decimal)
P. Addr: Physical address (hexadecimal)
L. Addr: Logical address (hexadecimal)
Code: Object code (hexadecimal)
Mnemonic: Disassembled code
BA to YP: Values of the CPU registers after finishing the cycle (hexadecimal)
SC, CC: Condition flag status
Memory: Memory access status (other than code fetch status)
MR: Memory read
MW: Memory write
[<address>] = <data>: Accessed memory address and read / write data (hexadecimal)

(1) When [Trace] window is opened:



```
INS. P.Addr L.Addr  Code    Mnemonic          BA   HL   IX   IY   SP   BR EP XP YP SC        CC   Memory
0217 000499 01:0499 93      INC IY            3E84 F828 F828 F060 F7F3 F0 00 00 00 00--N-C-  0000
0218 00049A 01:049A 92      INC IX            3E84 F828 F829 F060 F7F3 F0 00 00 00 00--N-C-  0000
0219 00049B 01:049B CF7601  LD [SP+01h],IX    3E84 F828 F829 F060 F7F3 F0 00 00 00 00--N-C-  0000 MW:[00F7F4]=29 MW:[00F7
0220 00049E 01:049E B001    LD A,#01h         3E01 F828 F829 F060 F7F3 F0 00 00 00 00--N-C-  0000
0221 0004A0 01:04A0 A6      PUSH IP           3E01 F828 F829 F060 F7F1 F0 00 00 00 00--N-C-  0000 MW:[00F7F2]=00 MW:[00F7
0222 0004A1 01:04A1 CEC600  LD XP,#00h        3E01 F828 F829 F060 F7F1 F0 00 00 00 00--N-C-  0000
0223 0004A4 01:04A4 CFFA    LD IX,SP          3E01 F828 F7F1 F060 F7F1 F0 00 00 00 00--N-C-  0000
0224 0004A6 01:04A6 CE4802  LD B,[IX+02h]     0401 F828 F7F1 F060 F7F1 F0 00 00 00 00--N-C-  0000 MR:[00F7F3]=04
0225 0004A9 01:04A9 AE      POP IP            0401 F828 F7F1 F060 F7F3 F0 00 00 00 00--N-C-  0000 MR:[00F7F1]=00 MR:[00F7
0226 0004AA 01:04AA 01      ADD A,B           0405 F828 F7F1 F060 F7F3 F0 00 00 00 00------  0000
0227 0004AB 01:04AB 50      LD L,A            0405 F805 F7F1 F060 F7F3 F0 00 00 00 00------  0000
0228 0004AC 01:04AC B105    LD B,#05h         0505 F805 F7F1 F060 F7F3 F0 00 00 00 00------  0000
0229 0004AE 01:04AE 42      LD A,L            0505 F805 F7F1 F060 F7F3 F0 00 00 00 00------  0000
0230 0004AF 01:04AF A6      PUSH IP           0505 F805 F7F1 F060 F7F1 F0 00 00 00 00------  0000 MW:[00F7F2]=00 MW:[00F7
0231 0004B0 01:04B0 CEC600  LD XP,#00h        0505 F805 F7F1 F060 F7F1 F0 00 00 00 00------  0000
0232 0004B3 01:04B3 CFFA    LD IX,SP          0505 F805 F7F1 F060 F7F1 F0 00 00 00 00------  0000
0233 0004B5 01:04B5 CE4402  LD [IX+02h],A     0505 F805 F7F1 F060 F7F1 F0 00 00 00 00------  0000 MW:[00F7F3]=05
0234 0004B8 01:04B8 AE      POP IP            0505 F805 F7F1 F060 F7F3 F0 00 00 00 00------  0000 MR:[00F7F1]=00 MR:[00F7
0235 0004B9 01:04B9 3A80    XOR A,#80h        0585 F805 F7F1 F060 F7F3 F0 00 00 00 00--N---  0000
0236 0004BB 01:04BB CEB880  XOR B,#80h        8585 F805 F7F1 F060 F7F3 F0 00 00 00 00--N---  0000
0237 0004BE 01:04BE 31      CP A,B            8585 F805 F7F1 F060 F7F3 F0 00 00 00 00-----Z  0000
0238 0004BF 01:04BF CEE0CB  JRS LT,CBh        8585 F805 F7F1 F060 F7F3 F0 00 00 00 00-----Z  0000
0239 0004C2 01:04C2 F105    JRS 05h           8585 F805 F7F1 F060 F7F3 F0 00 00 00 00-----Z  0000
```

When the td command is input without <cycle>, the [Trace] window redisplays the latest data; when the td command is input with <cycle>, the trace data starting from <cycle> is displayed in the [Trace] window.

The display contents of the [Trace] window is updated after an execution of the target program.

All trace data can be displayed by scrolling the window.

(2) When [Trace] window is closed:

When the td command is input without <cycle>, the debugger displays 11 lines of the latest trace data in the [Command] window. When the td command is input with <cycle>, the debugger displays 11 lines of the trace data from <cycle> in the [Command] window.

```
>td↵
Start index (ENTER as 0)? : ↵
Ins. P.Addr L.Addr  Code     Mnemonic           BA   HL   IX   IY   SP  BR EP XP YP    SC     CC  Memory
0000 000179 00:0179 CF7000   LD BA,[SP+00h]    xxxx xxxx xxxx F0E4 xxxx xx xx xx xx 11------ 00C0
0001 00017A 00:017A                            xxxx xxxx xxxx F0E4 xxxx xx xx xx xx 11------ 00C0
0002 00017B 00:017B                            xxxx xxxx xxxx F0E4 xxxx xx xx xx xx 11------ 00C0
0003 0077FC 00:F7FC                            xx00 xxxx xxxx F0E4 xxxx xx xx xx xx 11------ 00C0 MR:[00F7FC]=00
0004 00017C 00:017C                            xx00 xxxx xxxx F0E4 xxxx xx xx xx xx 11------ 00C0
0005 0077FD 00:F7FD                            0100 xxxx xxxx F0E4 xxxx xx xx xx xx 11------ 00C0 MR:[00F7FD]=01
0006 00017C 00:017C 98       DEC BA            0100 xxxx xxxx F0E4 xxxx xx xx xx xx 11------ 00C0
0007 00017D 00:017D                            0100 xxxx xxxx F0E4 xxxx xx xx xx xx 11------ 00C0
0008 00017D 00:017D CF7400   LD [SP+00h],BA    00FF xxxx xxxx F0E4 xxxx xx xx xx xx 11------ 00C0
0009 00017E 00:017E                            00FF xxxx xxxx F0E4 xxxx xx xx xx xx 11------ 00C0
0010 00017F 00:017F                            00FF xxxx xxxx F0E4 xxxx xx xx xx xx 11------ 00C0
>td 11↵
Ins. P.Addr L.Addr  Code     Mnemonic           BA   HL   IX   IY   SP  BR EP XP YP    SC     CC  Memory
0011 0077FC 00:F7FC                            00FF xxxx xxxx F0E4 xxxx xx xx xx xx 11------ 00C0 MW:[00F7FC]=FF
0012 000180 00:0180                            00FF xxxx xxxx F0E4 xxxx xx xx xx xx 11------ 00C0
0013 0077FD 00:F7FD                            00FF xxxx xxxx F0E4 xxxx xx xx xx xx 11------ 00C0 MW:[00F7FD]=00
0014 000180 00:0180 E7EB     JRS NZ,EBh        00FF xxxx xxxx F0E4 xxxx xx xx xx xx 11------ 00C0
0015 000181 00:0181                            00FF xxxx xxxx F0E4 xxxx xx xx xx xx 11------ 00C0
0016 00016C 00:016C CE3501   CP [HL],#01h      00FF xxxx xxxx F0E4 xxxx xx xx xx xx 11------ 00C0
0017 00016D 00:016D                            00FF xxxx xxxx F0E4 xxxx xx xx xx xx 11------ 00C0
0018 00016E 00:016E                            00FF xxxx xxxx F0E4 xxxx xx xx xx xx 11------ 00C0
0019 000C53 00:0C53                            00FF xxxx xxxx F0E4 xxxx xx xx xx xx 11------ 00C0 MR:[000C53]=01
0020 00016F 00:016F E706     JRS NZ,06h        00FF xxxx xxxx F0E4 xxxx xx xx xx xx 11-----Z 00C0
0021 000170 00:0170                            00FF xxxx xxxx F0E4 xxxx xx xx xx xx 11-----Z 00C0
>
```

(3) During log output

When the command execution result is being output to a log file as specified by the log command, the trace data is displayed in the [Command] window and its contents are also output to the log file.

If the [Trace] window is closed, data is displayed in the same way as in (2) above.

If the [Trace] window is open, its contents are redisplayed. In this case, the same number of lines are displayed in the [Command] window as displayed in the [Trace] window.

(4) Successive display

When you execute the td command, the trace data can be displayed successively by entering the [Enter] key only until some other command is executed.

When you input the [Enter] key, the [Trace] window is scrolled forward one screen.

When displaying data in the [Command] window, 11 lines of data preceding the previously displayed cycle are displayed in the [Command] window (the same number of lines as displayed in the [Trace] window if the command is executed during log output).

The direction of display is such that each time you input the [Enter] key, data on older execution cycles is displayed (FORWARD). This direction can be reversed (BACKWARD) by entering the [B] key. To return the display direction to FORWARD, input the [F] key. If the [Trace] window is open, the direction in which the window is scrolled is also changed.

```
>td 100↵                    ... Started display in FORWARD.
  (Data on cycle Nos. 100 to 110 is displayed.)
>b↵                         ... Changed to BACKWARD.
  (Data on cycle Nos. 99 to 89 is displayed.)
>↵                          ... Continued display in BACKWARD.
  (Data on cycle Nos. 88 to 78 is displayed.)
>f↵                         ... Changed back to FORWARD.
  (Data on cycle Nos. 99 to 89 is displayed.)
>
```

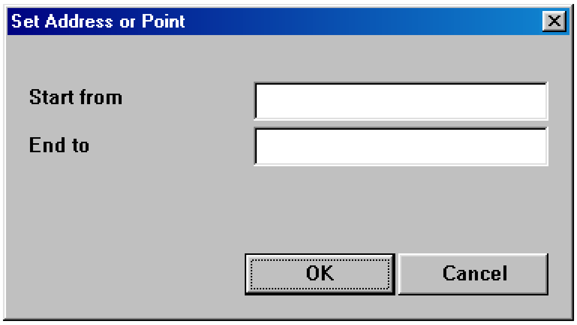
Notes

- Specify the trace cycle No. within the range of 0 to 0x1fff (8,191). An error results if this limit is exceeded.
- The trace memory receives new data until a break occurs. When the trace memory is filled, old data is overwritten by new data.

GUI utility

[Trace | Trace] menu item

When this menu item is selected, the [Trace] window opens and displays the latest trace data.
At the same time, the dialog box shown below appears to specify the cycle number to be displayed.



Enter the display start and end cycle numbers in hexadecimal to the [Start from] and [End to] text boxes, respectively, and then click [OK]. These entries can be omitted, and if [Start from] is omitted, the trace data is displayed from cycle number 0.

[Trace | Setting...] menu item

When this menu item is selected, the [Trace Information Setting] dialog box appears to set trace conditions. See Section 13.8.6, "Trace Function", for details.

ts (trace search)

Function

This command searches trace information from the trace memory under a specified condition. The search condition can be selected from three available conditions:

1. Search by executed address
   In this mode, you can specify a program memory address. The debugger searches the cycle in which the specified address is executed.

2. Search for a specified memory read cycle
   In this mode, you can specify a data memory address. The debugger searches the cycle in which data is read from the specified address.

3. Search for a specified memory write cycle
   In this mode, you can specify a data memory address. The debugger searches the cycle in which data is written to the specified address.

Format

(1) >ts <option> <address>↵ (direct input mode)

(2) >ts↵ (guidance mode)

1. pc address 2. data read address 3. data write address ...? <1 | 2 | 3>↵
Search address ?: <address>↵
(Search result is displayed)
>

<option>: Search condition; pc (= executed address), dr (= data read address), dw (= data write address)
<address>: Search address; hexadecimal or symbol (IEEE-695 format only)
Condition: 0 ≤ address ≤ 0x7fffff (when pc is specified), 0 ≤ address ≤ 0xffffff (when dr/dw is specified)

Examples

The search results are displayed in the [Trace] window if it is opened; otherwise, the results are displayed in the [Command] window in the same way as for the td command.

Format (1)

```
>ts pc 823↵
Searching trace data ... OK!
Ins. P.Addr L.Addr  Code     Mnemonic              BA   HL   IX   IY ...
0006 000823 00:0823                               0006 xxxx xxxx xxxx ...
0007 000823 00:0823 E7FA     JRS NZ,FAh           0006 xx07 xxxx xxxx ...
>
```

Format (2)

```
>ts↵
1.pc address  2.data read address  3.data write address ...? 1↵
Searching trace data ... OK!
Ins. P.Addr L.Addr  Code     Mnemonic              BA   HL   IX   IY ...
0006 000823 00:0823                               0006 xxxx xxxx xxxx ...
0007 000823 00:0823 E7FA     JRS NZ,FAh           0006 xx07 xxxx xxxx ...
>
```

When command execution results are being output to a log file by the log command, the search results are displayed in the [Command] window as well as output to the log file even when the [Trace] window is opened.

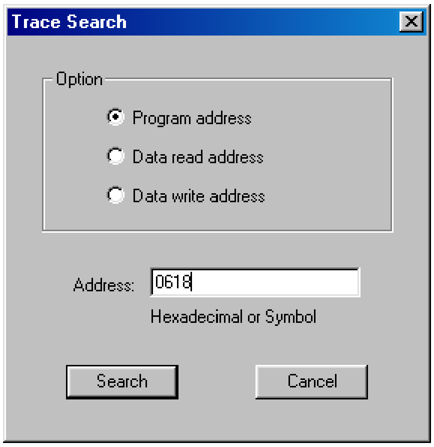
Note

The address specified for search must be within the range of the memory area available for each microcomputer model.
An error results if the limit is exceeded or the input one is not a hexadecimal number or not a valid symbol.

GUI utility

[Trace | Trace Search...] menu item

When this menu item is selected, a dialog appears for setting a search condition.



Trace Search

Option

● Program address

○ Data read address

○ Data write address

Address: 0618

Hexadecimal or Symbol

Select an option using the radio button and enter an address in the text box, then click [OK].

tf (trace file)

Function

This command saves the specified range of the trace information displayed in the [Trace] window by the td or ts command to a file.

Format

(1) >tf <file name> [<cycle1> [<cycle2>]]↵ **(direct input mode)**

(2) >tf↵ **(guidance mode)**

Start index (min 0) ? : <cycle1>↵
End index (max 8191) ? : <cycle2>↵
File Name ? : <file name>↵
>

<file name>: Output file name (path can also be specified)
<cycle1>: Start cycle number; decimal (0 by default)
<cycle2>: End cycle number; decimal (0x1fff by default)
Condition: 0 ≤ cycle1 ≤ cycle2 ≤ 0x1fff

Examples

Format (1)

```
>tf trace.trc↵             ... Saves all trace information extracted by the td command.
8191-8000
8000-7000
    :
1000-   1
OK!
>
```

Format (2)

```
>tf↵
Start index (min 0) ? :0↵
End index (max 8191) ? :100↵
File name   ? :test.trc↵
1000-   1
OK!
>
```

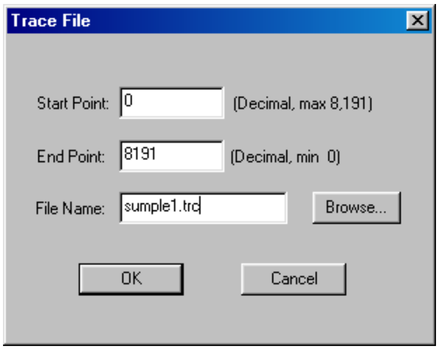
Notes

- If an existing file is specified, the file is overwritten with the new data.
- The default value of <cycle1> is 0, and the default value of <cycle2> is 0x1fff (8191), the latest trace data.

GUI utility

[Trace | Trace File...] menu item

When this menu item is selected, a dialog box appears allowing specification of the parameters.



Start Point: 0 (Decimal, max 8,191)
End Point: 8191 (Decimal, min 0)
File Name: sumple1.trc

Enter a start cycle number, end cycle number and a file name, then click [OK].
To save all the trace information, leave the [Start Point] and [End Point] boxes blank.
The file name can be selected using a standard file selection dialog box that appears by clicking [Browse...].

13.9.12 Coverage

CV (coverage)

Function

This command displays the coverage information (accessed addresses) acquired by the ICE while running the target program.

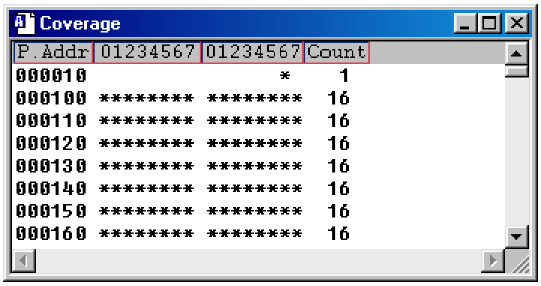
Format

>cv <address1> [<address2>]↵ **(direct input mode)**

<address1>: Start address; hexadecimal or symbol (IEEE-695 format only)
<address2>: End address; hexadecimal or symbol (IEEE-695 format only)
Condition: 0 ≤ address1 ≤ address2 ≤ last memory address (0xffffff)

Examples

(1) When [Coverage] window is opened:



```
P.Addr 01234567 01234567 Count
000010                 *     1
000100 ******** ********    16
000110 ******** ********    16
000120 ******** ********    16
000130 ******** ********    16
000140 ******** ********    16
000150 ******** ********    16
000160 ******** ********    16
```

Coverage information is displayed in a 16 bytes per line format beginning with <address1>. P.Addr indicates the start address (physical address) of each line. The accessed addresses are marked with an asterisk (*) and addresses not accessed are marked with a space " ". The Count value indicates the total addresses accessed (in bytes) among the 16 bytes on each line. All acquired data can be displayed by scrolling the screen.

(2) When [Coverage] window is closed:

If <address2> is omitted when executing the cv command, coverage information from <address1> to the end address is displayed in the [Command] window.
If <address2> is specified when executing the cv command, coverage information from <address1> to <address2> is displayed.

```
>cv 100↵                    ...Shows the executed addresses following 0x000100.
   000100 - 00020e
   000233 - 0002c4
   0004e4 - 0004e9
          :
   00ff40
   00ff54 - 00ff55
   00ff61
   00ff63
>cv 100 1ff↵                ...Shows the executed addresses from 0x000100 to 0x0001ff.
   000100 - 0001ff
>
```

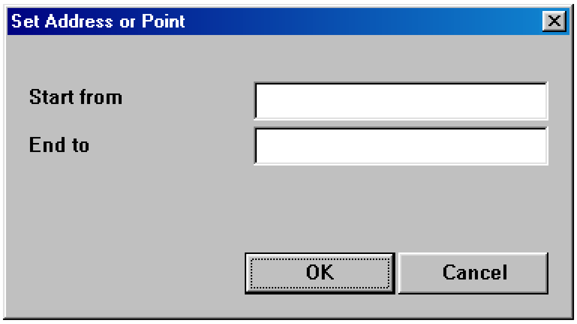
Notes

- Coverage information is recorded according to the acquisition mode (i.e., whether to acquire information from the entire address space or data space only) and acquisition range (selected 64 KB area) specified with the debugger's coverage options. The dialog box displayed by selecting [Setting...] from the [Coverage] menu is used to set the coverage options. For details, see Section 13.8.7, "Coverage".
- The addresses specified here must be within the range of the program memory area available with each microcomputer model.
  An error results if the limit is exceeded or the input one is not a hexadecimal number or a valid symbol.
- An error results if the start address is larger than the end address.

GUI utility

[Coverage | Coverage] menu item

Selecting this menu command opens the [Coverage] window.
At this time, the dialog box shown below appears, allowing you to specify the address from which to start displaying coverage information.



Enter the address in hexadecimal notation from which to start displaying coverage information in the [Start from] text box, then click the [OK] button. To display coverage information in the [Coverage] window, you can leave [End to] blank. Note that the start and end addresses of the selected 64 KB area are assumed if start and end addresses are not entered in these text boxes.

CVC (coverage clear)

Function

This command clears the coverage information.

Format

>cvc↵ (direct input mode)

GUI utility

[Coverage | Coverage Clear] menu item

When this menu item is selected, the cvc command is executed.

13.9.13 Command File

com (execute command file)

Function

This command reads a command file and executes the debug commands written in that file. You can execute the commands successively, or set an 0 to 256 seconds of interval between each command execution in 1-second increments.

Format

(1) >com <file name> [<interval>]↵ **(direct input mode)**

(2) >com↵ **(guidance mode)**

File name ? <file name>↵
Execute commands 1. successively 2. with wait ...? <1 | 2>↵
Interval (0 - 256 seconds) : <interval>↵ (appears only when "2. With wait" is selected)
>*(Display execution progress)*

<file name>: Command file name (path can also be specified)
<interval>: Interval (wait seconds) between each command; decimal (0–256)

Examples

Format (1)

```
>com batch1.cmd↵
>.....                    ... Commands in "batch1.com" are executed successively.
```

Format (2)

```
>com↵
File name   ? test.cmd↵
Execute commands   1. successively   2. with wait   ...? 2↵
Wait time (0 - 256 seconds) : 2↵
>.....                    ... 2 sec. of interval is inserted after each command execution.
```

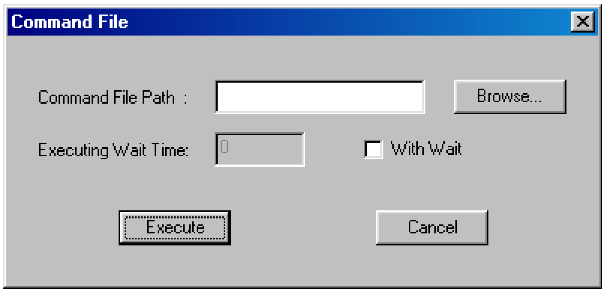
Notes

- Any contents other than commands cannot be written in the command file.
- An error results if the file you specified does not exist.
- Another command file can be read from a command file. However, the nesting of command files is limited to a maximum of 5 levels. An error results if a com (or cmw) command at the sixth level is encountered, the commands in the file specified by that com (or cmw) command will not be executed, but the subsequent execution of the commands in upper level files will be executed continuously.
- If you specify an interval more than 256 seconds, it is set to 256 by default.
- Use the hot key ([CTRL]+[Q]) to stop executing a command file.

GUI utility

[Run | Command File...] menu item

When this menu item is selected, a dialog box appears allowing selection of a command file.



Enter a file name into the [Command File Path] text box, then click [Execute]. The file name can be selected using a standard file selection dialog box that appears by clicking [Browse...].
To specify an interval, select [With Wait] and enter the number of seconds into the [Executing Wait Time] text box.

cmw (execute command file with wait)

Function

This command reads a command file and executes the debug commands written in that file at predetermined time intervals.
The execution interval of each command can be set in a range of 1 to 256 seconds (in 1-second increments) using the md command. In the initial debugger settings, the execution interval is 1 second.

Format

(1) >cmw <file name>↵ **(direct input mode)**

(2) >cmw↵ **(guidance mode)**
File name ? <file name>↵
>*(Display execution progress)*

<file name>: Command file name (path can also be specified)

Examples

Format (1)

```
>cmw batch1.cmd↵
>.....
```

Format (2)

```
>cmw↵
File name   ? test.cmd↵
>.....
```

Notes

- Any contents other than commands cannot be written in the command file.
- An error results if the file you specified does not exist.
- Another command file can be read from a command file. However, the nesting of command files is limited to a maximum of 5 levels. An error results if a cmw (or com) command at the sixth level is encountered, the commands in the file specified by that cmw (or com) command will not be executed, but the subsequent execution of the commands in upper level files will be executed continuously.
- If the cmw command is written in the command file that you want to be read by the com command, all other commands following that command in the file (even when a com command is included) will be executed at predetermined time intervals.
- Use the hot key ([CTRL]+[Q]) to stop executing a command file.

GUI utility

None
However, the same function as the cmw can be executed using [Command File...] in the [Run] menu (see the com command).

rec (record commands to a file)

Function

This command records all the debug commands executed following this command to a specified command file.

Format

(1) >rec <file name>↵ **(direct input mode)**

(2) >rec↵ **(guidance mode)** ...See Examples for guidance.

<file name>: Command file name (path can also be specified)

Examples

(1) First rec execution after debugger starts up

```
>rec↵
File name   ? sample.cmd↵
1. append   2. clear and open   ...? 2↵
>
```

...Displayed if the file is already exists.

(2) rec command input in the second and following sessions

```
>rec↵
Set to record off mode.
.....
>rec↵
Set to record on mode.
```

...Record function toggles when rec is input.

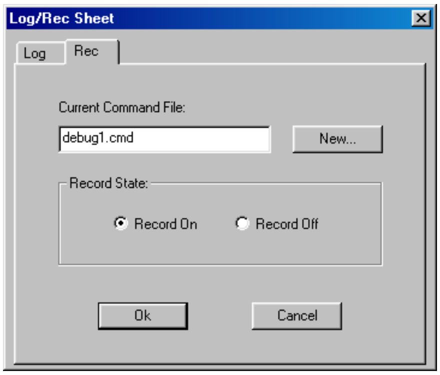
Notes

- In record on mode, besides the commands directly input in the [Command] window, the commands executed by selecting from a menu or with a tool bar button (except the [Help] menu command) are also displayed in the [Command] window, and output to the specified file.
  If you modify the register value or data memory contents by direct editing in the [Register] or [Dump] window, or set breakpoints in the [Source] window by double-clicking the mouse, the corresponding commands are also displayed in the [Command] window, and output to the specified file.
- At the first time, you should specify the file name to which all debug commands following the rec command will be output.
- Once an output command file is opened, the recording is suspended and resumed (toggled) every time you input the rec command. This toggle operation remains effective until you terminate the debugger. If you want to record following commands to another file, you can use format (1) to specify the file name, then current output file is closed and all following commands will be recorded in the newly specified file.
- If you want to execute some commands frequently, you can record them to a file at the first execution, and then use the com or cmw command to execute that command file you made.

GUI utility

[Option | Record...] menu item

Selecting this menu command displays a dialog box for specifying a command file. To specify a new command file, enter the command file name in [Current Command File] or click the [New...] button and select from the list that appears.
If the debugger has already started recording commands, use the [Record State] radio buttons to turn recording on or off.



Log/Rec Sheet

Current Command File: debug1.cmd

Record State: ● Record On ○ Record Off

13.9.14 log

log (log)

Function

This command saves the input commands and the execution results to a file.

Format

(1) >log <file name>↵ **(direct input mode)**

(2) >log↵ **(guidance mode)** ...See Examples for guidance.

<file name>: Log file name (path can also be specified)

Examples

(1) First log execution after debugger starts up

```
>log↵
File name   ? debug1.log↵
1. append   2. clear and open   ...? 2↵        ...Displayed if the file is already exists.
>
```

(2) log command input in the second and following sessions

```
>log↵
Set to log off mode.           ...Logging function toggles when log is input.
.....
>log↵
Set to log on mode.
```

Notes

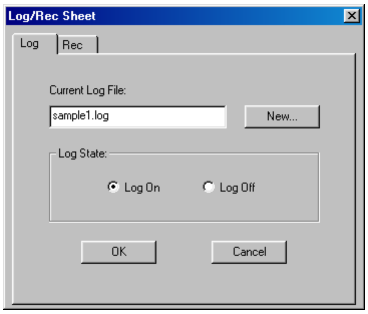
- In log on mode, the contents displayed in the [Command] window are written as displayed directly to the log file.
  The commands executed by selecting from a menu or with a tool bar button are displayed in the [Command] window. However, the [Help] menu and button commands are not displayed. If you modify the register value or data memory contents by direct editing in the [Register] or [Dump] window, or set breakpoints in the [Source] window by double-clicking the mouse, the corresponding commands and the execution results are also displayed in the [Command] window, and output to the specified file.

  The displayed contents of the [Source], [Dump], [Trace] or [Register] window produced by command execution are displayed in the [Command] window as well. The on-the-fly information is also displayed. However, the updated contents of each window after some execution, as well as the contents of each window scrolled by scroll bar or arrow keys, are not displayed.
- At the first time, you should specify the file name to which all following debug commands and execution results will be output.
- Once a log file is open, log output is suspended and resumed (toggled) every time you input the log command. This toggle operation remains effective until you terminate the debugger. If you want to specify a new log file, you can use format (1) to specify the file name, then current log file is closed and following commands and results will be output to the newly specified file.

GUI utility

[Option | Log...] menu item

Selecting this menu command displays a dialog box for specifying a log file. To specify a new log file, enter the log file name in [Current Log File] or click the [New...] button and select from the list that appears.
If the debugger has already started logging commands, use the [Log State] radio buttons to turn logging on or off.


Log/Rec Sheet

Current Log File: sample1.log

Log State: ● Log On ○ Log Off

13.9.15 Map Information

ma (map information)

Function

This command displays the map information that is set by a parameter file.

Format

>ma↵ **(direct input mode)**

Example

After the command is input, the system displays the map information in the internal memory area, external memory area and I/O area.

```
>ma↵
[Internal memory]
  RAM 00F000 - 00F7FF
  STK 00F500 - 00F7FF
  LCD 00F800 - 00F842
  LCD 00F900 - 00F942
  LCD 00FA00 - 00FA42
  LCD 00FB00 - 00FB42
  LCD 00FC00 - 00FC42
  LCD 00FD00 - 00FD42
[External memory]
  ROM 000000 - 00BFFF
  RAM 080000 - 080001
  RAM 100000 - 107FFF
  RAM 180000 - 1801FF
[I/O memory]
        0 1 2 3 4 5 6 7 8 9 A B C D E F
  FF00  * * *
  FF10  * * * *
  FF20  * * * * * *
  FF30  * * * * * * *
  FF40  * * * * * *     * * *
  FF50  * * * * * *
  FF60  * * * *
  FF70  * * * * * * * * *
  FF80
  FF90
  FFA0
  FFB0
  FFC0
  FFD0
  FFE0
  FFF0
```

∗ When displaying the map information of the I/O area, the mapped addresses are marked by the letter "∗".

GUI utility

None

13.9.16 FPGA Operation

xfer (xilinx fpga data erase)

Function

This command erases the contents of the FPGA on the standard peripheral circuit board inserted in the ICE.

Format

>xfer↵ **(direct input mode)**

Example

```
>xfer↵
>
```

After the command is entered, a dialog box appears to select start or cancel erasing.

Notes

- A dialog box appears to show the progress of erasing while executing. To abort erasing, click the [Cancel] button on the dialog box or press the [ESC] key. In this case, the standard peripheral circuit board cannot be used until the FPGA is erased and reprogrammed.
- Erase time is about TBD minutes TBD seconds (max.).

GUI utility

None

xfwr (xilinx fpga data write)

Function

This command writes peripheral circuit data to the FPGA on the standard peripheral circuit board inserted in the ICE.

Format

>xfwr <file name> ;{H | S} [;N]↵ (direct input mode)

<file name>:	FPGA data file (.mot: Motorola S, .mcs: Intel HEX)
H:	Load Intel HEX file
S:	Load Motorola S file
N:	Skip erasing before writing data

Examples

```
>xfwr ..\ice\fpga\c88xxx.mot ;S↵
>
```

In this example, the main FPGA is erased and then data in the c88xxx.mot file (Motorola S format) is written to it.

```
>xfwr ..\ice\fpga\c88xxx.mot ;S ;N↵
>
```

In this example, erasing before writing is skipped. However, the main FPGA must be erased beforehand.

Notes

- Use the file provided by Seiko Epson as the data to be written without modifying the contents. Also the file extension cannot be changed as it is .mot (Motorola S) or .mcs (Intel HEX). Specifying an illegal file results in an error and data cannot be written.
- The N option can be specified when the FPGA has been erased completely using the xfer command. When writing data to the FPGA that has not been erased, do not specify the N option.
- A dialog box appears to show the progress while executing. To abort execution, click the [Cancel] button on the dialog box or press the [ESC] key. In this case, the standard peripheral circuit board cannot be used until the FPGA is erased and reprogrammed.
- Process time including erase is about TBD minutes (max.).

GUI utility

None

xfcp (xilinx fpga data compare)

Function

This command compares the contents between the FPGA and the specified file.

Format

>xfcp <file name> ;{H | S}↵ (direct input mode)

<file name>: FPGA data file (.mot: Motorola S, .mcs: Intel HEX)
H: Intel HEX file
S: Motorola S file

Examples

```
>xfcp ..\ice\fpga\c88xxx.mot ;S↵
>                                      ...No error has occurred.

>xfcp ..\ice\fpga\c88yyy.mot ;S↵
Warning : Verify error                 ...Verify error has occurred.
0X00000  0XFF                          ...Error addresses and data in the FPGA are displayed.
0X00001  0X84
0X00002  0XAB
   :      :
>
```

Notes

- Data is verified only within the valid address range in the specified file. If the FPGA contains data outside the range, it is not verified.
- Use the file provided by Seiko Epson as the data to be compared without modifying the contents. Also the file extension cannot be changed as it is .mot (Motorola S) or .mcs (Intel HEX). Specifying an illegal file results in an error.
- A dialog box appears to show the progress while executing. To abort execution, click the [Cancel] button on the dialog box or press the [ESC] key.

GUI utility

None

xdp (xilinx fpga data dump)

Function

This command displays the content of the FPGA on the standard peripheral circuit board to the [Command] window in a 16 words/line hexadecimal dump format.

Format

>xdp <address1> [<address2>]↵ (direct input mode)

<address1>: Start address to display; hexadecimal
<address2>: End address to display; hexadecimal
Condition: 0 ≤ address1 ≤ address2 ≤ FPGA end address

Examples

If only <address1> is defined, the debugger displays data for 256 words from <address1>.

```
>xdp 0↵
Addr   +0 +1 +2 +3 +4 +5 +6 +7  +8 +9 +A +B +C +D +E +F
00000: FF 84 AB EF F9 D8 FF BB  FB BB BF FB BF BF FB BF
00010: BB FB BB BF BB BF FB BB  BF BF FB BB FF EE FF EE
00020: EF FE D7 FB FE EE EF EF  EE EE FE EE FB FE EF EF
  :               :                        :
000E0: FF FF FF FF FB FF FF FF  BD DF FB FD DF FF FF FF
000F0: FF FF BF FF FF FF FF F9  FF FF FF FF FF FF FF FF
>
```

If both <address1> and <address2> are defined, the debugger displays data from <address1> to <address2>.

```
>xdp 100 100↵
Addr   +0 +1 +2 +3 +4 +5 +6 +7  +8 +9 +A +B +C +D +E +F
00100: FF
>
```

Notes

- An error results if the specified address is not a hexadecimal number.
- An error results if the start address is larger than the end address.

GUI utility

None

13.9.17 Quit

q (quit)

Function

This command quits the debugger.

Format

>q↵ (direct input mode)

GUI utility

[File | Exit] menu item

Selecting this menu item terminates the debugger.

13.9.18 Help

? (help)

Function

This command displays the input format of each command.

Format

(1) ? (direct input mode)
(2) ? <n> (direct input mode)
(3) ? <command> (direct input mode)

<n>:	Command group number; decimal
<command>:	Command name
Condition:	1 ≤ n ≤ 6

Examples

When you input the command in Format 1 or 2, the system displays a list of commands classified by function. Use the command in Format 3 if you want to display the input format of each individual command.

```
>?↵
group 1: data & register ............ dd,de,df,dm,ds/rd,rs
group 2: execution & break .......... g,gr,s,n,se,rst/bp,bpa,bpr,bc(bpc),bas,ba,bar,bd,bdr,bl,bac
group 3: source & symbol ............ u,sc,m/sy/w
group 4: file & flash rom ........... lf, par/xfer,xfwr,xfcp,xdp
group 5: trace & coverage ........... td,ts,tf/cv,cvc
group 6: others ..................... par/com,cmw,rec/log/ma/q/?
 Type "? <group #>" to show group or "? <command>" to get usage of the command.
>? 1↵
group 1: data & register
dd (data dump), de (data enter), df (data fill), dm (data move), ds (data search)
rd (register display), rs (register set)
 Type "? <command>" to get usage of the command.
>? dd↵
dd (data dump): dump memory content with hexadecimal format
usage: dd [addr1] [addr2] [unit]   ... dump from 0x0 in byte unit if without parameter
       dd [addr1] [@size] [unit]   ... dump from 0x0 in byte unit if without parameter
unit: display unit (-B (default) / -W / -L / -F / -D)
```

GUI utility

None

13.10 Error Messages

Debugger error messages

Error message	Description
Error : Address out of range : use 0x000000 - 0xffffff	The specified address is outside the valid range.
Error : Address out of range, use 0 - 0x7FFFFF	The address specified here is outside the program memory area.
Error : Address out of range, use 0 - 0xFFFFFF	The address specified here is outside the data memory area.
Error : Cannot open device(ICE88UR)	Failed to connect to the ICE.
Error : Cannot open file	Cannot open the file.
Error : Checksum error	Checksum resulted in an error.
Error : Coverage mode is off or the coverage mode is not supported	Coverage mode is turned off or the ICE being used does not support coverage mode.
Error : Data out of range, use 0 - 0xFF	The specified value is outside the valid range of data.
Error : DLL Initialization error	Failed to initialize DLL.
Error : End address < start address	The end address specified here is smaller than the start address.
Error : End index < start index	The end cycle specified here is smaller than the start cycle.
Error : Error file type (extension should be CMD)	The specified file extension is not effective as a command file.
Error : Error file type (extension should be PAR)	The specified file extension is not effective as a parameter file.
Error : Failed ICE88UR initialization	Failed to initialize the ICE.
Error : Failed to initialize DLL : %s	Failed to initialize DLL.
Error : Failed to Load DLL	Failed to load DLL needed to start DB88.
Error : Failed to open : %s	Could not open the file.
Error : Failed to read BA	Error occurred when reading the BA register.
Error : Failed to read BR	Error occurred when reading the BR register.
Error : Failed to read CB	Error occurred when reading the CB register.
Error : Failed to read CC	Error occurred when reading the CC register.
Error : Failed to read EP	Error occurred when reading the EP register.
Error : Failed to read file : %s	Error occurred when reading the file.
Error : Failed to read HL	Error occurred when reading the HL register.
Error : Failed to read NB	Error occurred when reading the NB register.
Error : Failed to read PC	Error occurred when reading the PC register.
Error : Failed to read SC	Error occurred when reading the SC register.
Error : Failed to read SP	Error occurred when reading the SP register.
Error : Failed to read X	Error occurred when reading the X register.
Error : Failed to read Y	Error occurred when reading the Y register.
Error : Failed to road DLL : %s	Failed to load DLL.
Error : Failed to write BA	Error occurred when writing to the BA register.
Error : Failed to write BR	Error occurred when writing to the BR register.
Error : Failed to write CB	Error occurred when writing to the CB register.
Error : Failed to write CC	Error occurred when writing to the CC register.
Error : Failed to write EP	Error occurred when writing to the EP register.
Error : Failed to write HL	Error occurred when writing to the HL register.
Error : Failed to write NB	Error occurred when writing to the NB register.
Error : Failed to write PC	Error occurred when writing to the PC register.
Error : Failed to write SC	Error occurred when writing to the SC register.
Error : Failed to write SP	Error occurred when writing to the SP register.
Error : Failed to write X	Error occurred when writing to the X register.
Error : Failed to write Y	Error occurred when writing to the Y register.
Error : ICE88UR Diagnostic error	Detected an error during ICE self-diagnostic processing.
Error : Ice88ur Initialization failed	Failed to initialize the ICE.
Error : Ice88ur is already running	ICE88UR.EXE is up and running. (DB88 and ICE88UR cannot be started at the same time.)
Error : ICE88UR is turned off	Power to the ICE is turned off.
Error : Illegal initialization packet data	Initialization packet data is in error.
Error : Incorrect number of parameters	The number of parameters for the command is illegal.
Error : Incorrect r/w option, use r/w/*	The R/W option specified here is invalid.
Error : Incorrect register name, use PC/SP/IX/IY/A/B/HL/BR/CB/EP/XP/YP/SC	The register name specified here is invalid.
Error : Index out of range, use 0 - 8191	The specified trace cycle number is outside the valid range.
Error : Initialization failed! Please quit and restart!	Failed to initialize DB88. Please restart DB88.
Error : Input address does not exist	The address specified here has no breakpoints set.
Error : Invalid command	The command entered here is invalid.

Error message	Description
Error : Invalid data pattern	The data pattern entered here is invalid.
Error : Invalid display unit, use -B/-W/-L/-F/-D	The display unit specified here is invalid.
Error : Invalid DLL ModuleID	DLL identification error
Error : Invalid file name	The specified file extension is not effective as a program file or function option file.
Error : Invalid fsa file	The FSA file is invalid.
Error : Invalid hexadecimal string	This is an invalid hexadecimal string.
Error : Invalid value	The value entered here is invalid.
Error : Maximum nesting level(5) is exceeded, cannot open file	Command files have been nested exceeding the nesting limit.
Error : Memory ranges in %s are invalid or the file is not exist	The memory range of the CPU INI file is invalid.
Error : No symbol information	No symbol information is found. (No symbol files have been loaded.)
Error : Number of steps out of range, use 0 - 65535	The specified number of steps exceeds the limit.
Error : The Memory Area cannot include the boundary between 0x00FFFF and 0x010000	The specified area overlaps the 0x00FFFF–0x010000 address boundary.
Error : The Memory Area must be above 0x10000, and longer than 256 bytes	Any memory area specified above 0x010000 must be greater than 256 bytes in size.
Error : This command is not supported in current mode	The trace and coverage commands are not effective when trace or coverage is turned off.
Error : Unable to get the coverage area number	Failed to get the coverage area number.
Error : Unable to get the coverage mode	Failed to get coverage information.
Error : Unable to set SelfFlash check function	Could not set the SelfFlash check function.
Error : Unable to set the coverage area number	Failed to set the coverage area number.
Error : Unable to set the coverage mode	Failed to set coverage mode.
Error : Wrong Command line parameter	The startup parameters are incorrect.
Please load the selfflash library program	Please load the SelfFlash library program. (When the SelfFlash function is enabled, a library program must be loaded in the ICE.)
Warning : 64 break addresses are already set	The total number of breakpoints specified here exceeds 64.
Warning : Break address already exists	The specified address has a breakpoint already set.
Warning : Identical break address input	Two or more instances of the same address are specified on the command line.
Warning : Memory may be modified by SelfFlash	Memory contents may have been modified by the SelfFlash program.
Warning : SelfFlash program area is out of the current software pc break area. Please clear the break point(Address)	The SelfFlash program area does not match the currently set software break area. Please clear the breakpoint set at (Address). (If this breakpoint is not cleared, the program may stop at an unexpected location.)

ICE hardware error messages

Error message	Description
Error : Cannot be run in Free-Run mode	The ICE is operating in free-run mode.
Error : Cannot fine specified data	The specified data could not be found. (The search failed to find matching data.)
Error : ICE88UR is still keep a conservative mode	The ICE is operating in maintenance mode.
Error : ICE88UR power off execution abort	Power to the ICE main unit is off. Execution was aborted. (Power to the main unit has been shut off while running the program.)
Error : Insufficient memory for loading program	Failed to allocate memory for the program. (Windows system resources may be insufficient. Check available resources and quit unnecessary applications.)
Error : Vdd down or no clock	The power supply voltage for the target system is low, the target system is not powered on, or no clocks are supplied to the target system. (Effective only when Vdddown is set to 1 in the parameter file.)
Error : Verify error	A verify error occurred.
ICE88UR system error : ?? illegal packet	Detected an illegal packet.
ICE88UR system error : Command timeout	Detected a command time-out.
ICE88UR system error : Firmware packet error	Detected an error in EB: Firmware packet.
ICE88UR system error : Master reset	Detected MR: master reset.
ICE88UR system error : Not connected	The ICE is not connected or powered on.
ICE88UR system error : Not ready	The ICE is not ready.
Internal error : ICE88UR does not support this command version	The current version of the ICE does not support this command. (Please shut down the DB88 debugger immediately.)
Internal error : Illegal error code fetched. System crash possible	Nonexistent error code has been encountered. (Please shut down the ICE88UR debugger immediately.)
Processing terminated by hitting ESC-key	Processing terminated because the ESC key was pressed.

APPENDIX A ASSEMBLER (Sub tool chain)

A.1 Outline of Package

A.1.1 Introduction

The "S1C88 Family Assembler" is one of the software development tools of the CMOS 8-bit single chip microcomputer S1C88 Family. It consists of a cross assembler, linker and utilities to create programs. This package can commonly be used for all S1C88 Family models and allows for development of programs with macro function.

A.1.2 Outline of Software Tools

Figure A.1.2.1 shows the flow of software development using the structured assembler.

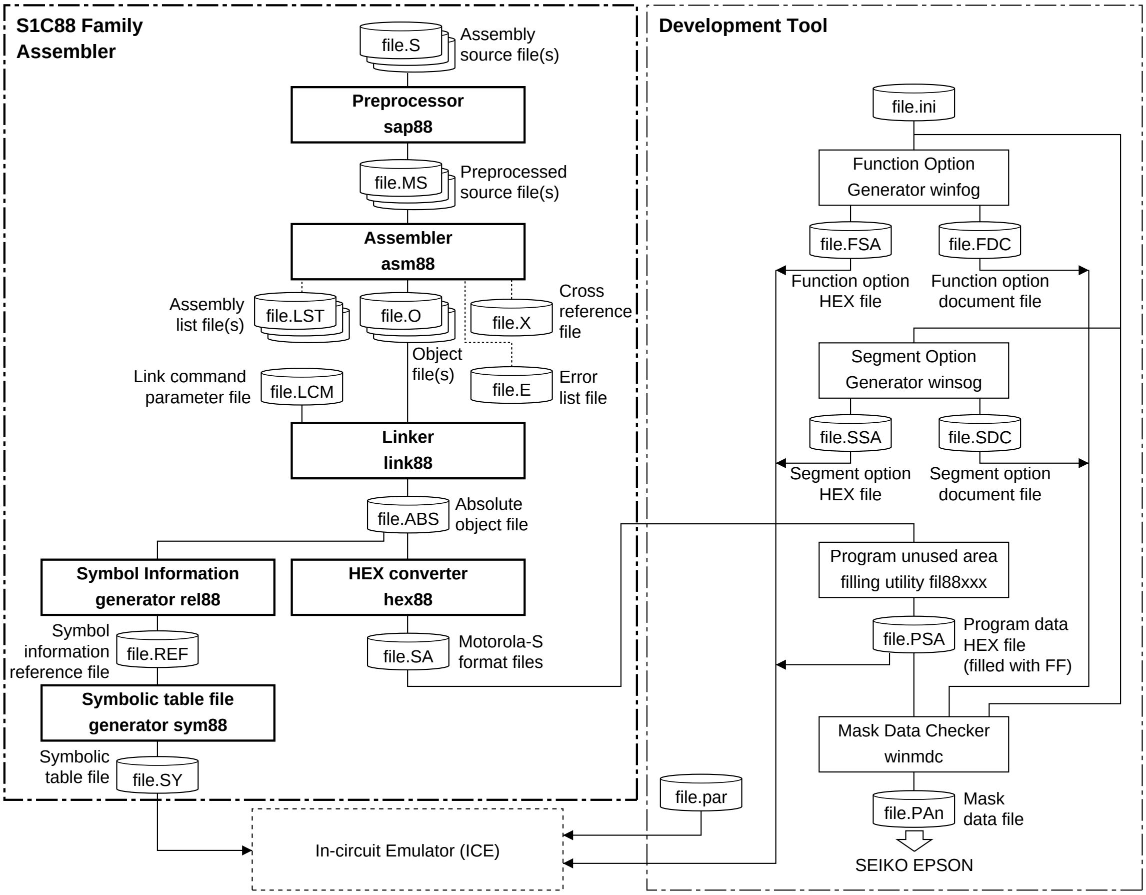


Fig. A.1.2.1 Software development flow using structured assembler

The basic functions of each program are as follows.

Structured preprocessor <sap88>

The sap88 structure preprocessor is a preprocessor used to add the macro function on the cross assembler asm88.
First create assembly source files including macro functions and process them with the sap88 to create the source files (in which macros are expanded into the S1C88 instructions) that can be assembled with the asm88.

Cross assembler <asm88>

The asm88 cross assembler assembles the program source file described by the S1C88 instruction set and pseudo-instruction and converts it into machine language.
The asm88 is compatible with the relocatable assembly for development by module, and creates relocatable object files used to link other modules via the linker.

Linker <link88>

The relocatable object file created with the asm88 is linked if there is more than one present and then converted into absolute (binary form) object file.

Other utilities

This package contains the following utility programs in addition to the earlier mentioned major programs.

Symbol information generator <rel88>

This is a program that obtains symbolic table information of the relocatable object file.
This utility is used for preprocessing of symbolic table generations.

Binary/HEX converter <hex88>

Converts the binary file into a Motorola S2 format HEX file (ASCII file).
This is basically used to convert the absolute object file output from the link88 linker into a HEX program file. The converted program data HEX file allows for debugging through hardware tools and creation of mask data.

Symbolic table file generator <sym88>

The sym88 symbolic table file generator converts a symbolic information file generated in file redirect with the rel88 symbol information generator to a symbolic table file that can be referenced in the ICE.
Loading the symbolic table file and the corresponding relocatable assembly program file in the ICE makes symbolic debugging possible.

Batch files

Batch files are included to automatically process basic tools and operations to promote efficient program development. Customize the file accordingly.

- ra88.bat: Batch file for relocatable assembly
- lk88.bat: Batch file for linking

Details on the batch file and how to create customized files will be explained in Section A.2, respectively under their titles.

A.2 Program Development Procedures

This section will start off by explaining the flow involved in program development and then give details on how each software tool of this package is used, in accordance with the development flow. Each software tools will be explained of its basic processing procedures and the flag settings (start-up command flag) required for the tools in terms of batch file commands. Refer to Appendix C, "Assembly Tool Reference" for more information on other flags, etc.

A.2.1 Development Flow

The following shows the program development procedure using the asm88 cross assembler.

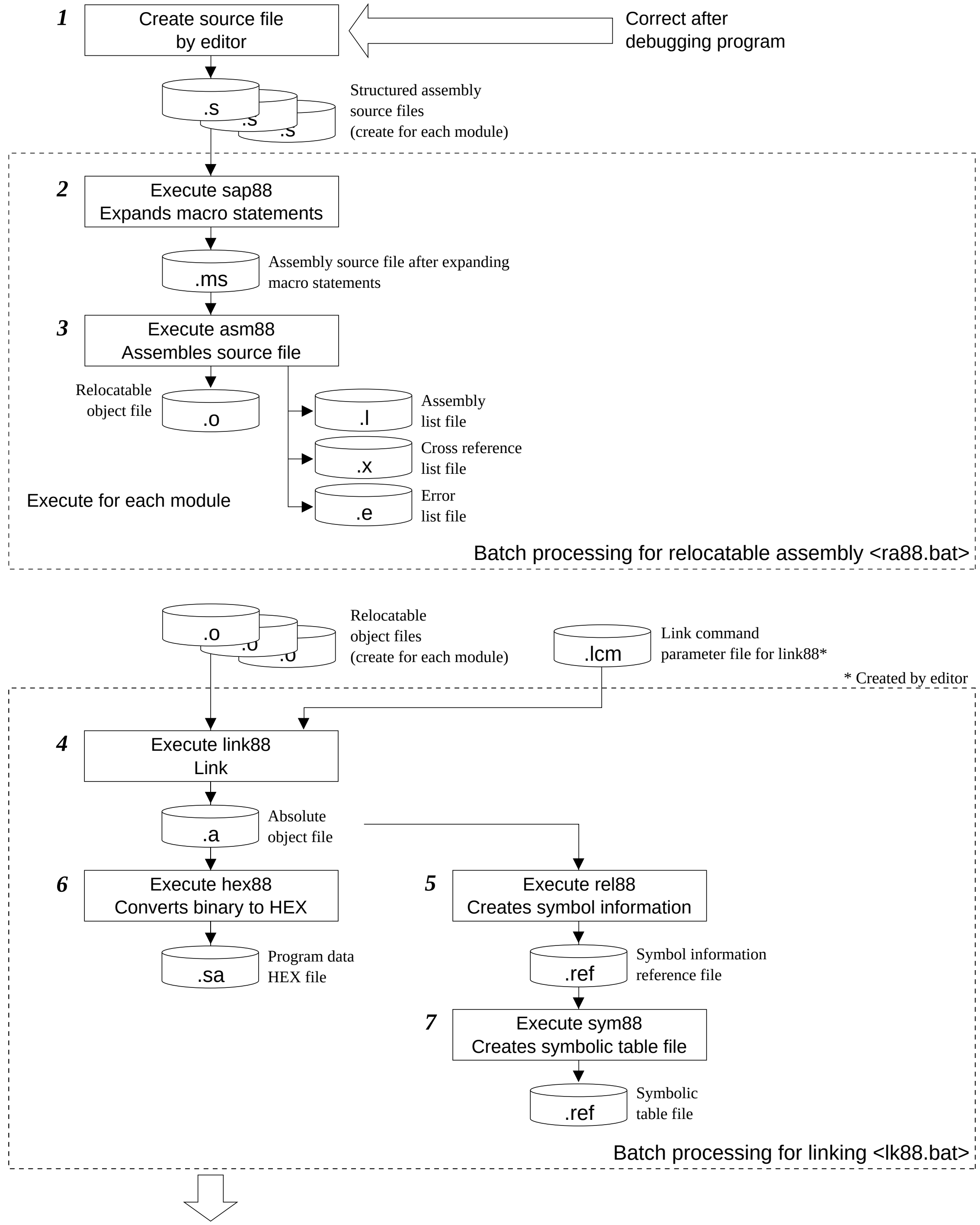
<Relocatable assembly and link>

– Create the entire program as a multiple module (development by module) –

Relocatable assembly refers to the assembling method in which programs are allocated into several parts (each allocated part is referred to as a module) according to the processing contents and then undergoing development procedures by each module.
The cross assembler can input assembly source files created with an editor and the files in which macros are expanded by the sap88.
Each module (relocatable object file) is linked via the linker after assembling and then consolidated into one program. The program memory address that allocates each module is determined through the link. Therefore, the developmental process in which the source program is created can be performed without regards to the address.
Debugging efficiency is boosted since this method allows for debugging by modules that have been allocated in small programs.

Figure A.2.1.1 shows the flow of program development upon using the relocatable assembly. This package contains "ra88.bat" and "lk88.bat" that are batch files containing basic processing tools. Customize accordingly. (Refer to Sections "A.2.3.4 Batch processing for relocatable assembly (ra88.bat)" and "A.2.4.5 Batch processing for linking (lk88.bat)" for more information on "ra88.bat" and "lk88.bat".)

Note: Prepare each relocatable module under 32K bytes so that they fit in one bank. Modules exceeding this capacity will result in an error message during linking. Thus, it will be necessary to allocate the program so that it is under 32K bytes. Similarly, the data size must be under 64K bytes so that it fits in one page.
The modules cannot be reallocated so that they span across both banks. In this case, the modules will be allocated so that it starts from the head of the next bank. The program memory (usable area) will be wasted if all modules are too large. Give consideration to each module size to prevent this.


1
Create source file by editor
Correct after debugging program
.s
.s
.s
Structured assembly source files (create for each module)
2
Execute sap88
Expands macro statements
.ms
Assembly source file after expanding macro statements
3
Execute asm88
Assembles source file
Relocatable object file
.o
.l
Assembly list file
.x
Cross reference list file
.e
Error list file
Execute for each module
Batch processing for relocatable assembly <ra88.bat>
.o
.o
.o
Relocatable object files (create for each module)
.lcm
Link command parameter file for link88*
* Created by editor
4
Execute link88
Link
.a
Absolute object file
6
Execute hex88
Converts binary to HEX
.sa
Program data HEX file
5
Execute rel88
Creates symbol information
.ref
Symbol information reference file
7
Execute sym88
Creates symbolic table file
.ref
Symbolic table file
Batch processing for linking <lk88.bat>

- System code setting and FF filling in unused program area by fil88XXX.
- Program debugging using ICE.
- Creating mask data of program.

Fig. A.2.1.1 Relocatable assembly development flow

A.2.2 Creating Source File

***Software used:* Editor**

Create the source file using an editor.
Small applications can be created solely in assembler language with the entire program as a single module.
What's more, source files for single module can also be allocated by using the INCLUDE pseudo-instruction of the sap88 structured preprocessor.
Generally, debugging requires appropriate consideration to module allocation since source files are each created for respective modules.

Create source files for assembler modules by using the S1C88 CPU instruction set or assembler pseudo-instructions.
Specify the assembly source file name with a ".s" on the extension.
Each source program statement basically comes in the following form.

Symbol field	Mnemonic field	Operand field	Comment field

- **Symbol field:**
  This field indicates the symbol. Always put a colon (:) immediately after the symbol, other than for EQU or SET command statements.
- **Mnemonic field:**
  This field indicates the operation code and pseudo-instruction.
- **Operand field:**
  This field indicates the operand, constant, variable, defined symbol, symbol that indicates the memory address and formula of each instruction.
- **Comment field:**
  A semi-colon (;) at the beginning of this field, then continued with a comment.

Refer to Appendix B of this manual for more information on how to create a source file.
Macro statement offered by the sap88 structured preprocessor and various pseudo-instructions of the asm88 cross assembler can be used for this assembler.
The following indicates an outlines of these statements and instructions.

<Instruction set>

All S1C88 Family models employs a S1C88 in the core CPU. Therefore, instructions are common for all models other than for CPU MODELS and mode limitations. Refer to the "S1C88 Core CPU Manual" for more information on the instructions, and refer to the "S1C88xxx Technical Manual" for control program examples of the peripheral circuit incorporated in each model.
The asm88 cross assembler is capable of converting all mnemonic instruction settings of the S1C88 into machine language.

<Macro statement>

Macro is used to priorly define a processing (sequence of instructions) frequently used in the program with a voluntary name to allow for it to be called out under that specific name. As a result, the need for routine procedures can be eliminated. (For more information refer to Appendix B.)
Macro statements are offered as pseudo-instructions of the sap88 and by putting it through the sap88 it is applied in the macro call-out portion in mnemonic form that can be assembled.

Example of macro definition

Before expanding

```
     subtitle       "example"
     public         main,work
     external       src_address,dst_address,counter
;
abc  equ            0ffh
;
     data
work:db             [1]
;
     code
;*************************************************
;**     * macro define *                        **
;*************************************************
nop3 macro
     nop
     nop                                                  Macro definition
     nop
     endm
;*************************************************
;**     * example *                             **
;*************************************************
main:
     ld   a,#abc
     lb   b,[work]
     nop3                      ; macro call ***           Macro call
     ld   ix,#src_address
     ld   iy,#dst_address
     ld   hl,[counter]
;***
     end
```

After expanding

```
     subtitle       "example"
     public         main,work
     external       src_address,dst_address,counter
;
abc  equ            0ffh
;
     data
work:db             [1]
;
     code
;*************************************************
;**     * macro define *                        **
;*************************************************
;*************************************************
;**     * example *                             **
;*************************************************
main:
     ld   a,#abc
     lb   b,[work]
     nop                                                  Macro statement expanded into
     nop                                                  mnemonics
     nop
     ld   ix,#src_address
     ld   iy,#dst_address
     ld   hl,[counter]
;***
     end
```

<Pseudo-instruction>

Pseudo-instruction by function	Description
Section setting pseudo-instructions (CODE, DATA)	Use to specify sections. * Specifies the program area and data area. (For more details refer to "A.2.3.2 Cross assembler (asm88)".)
Data definition pseudo-instructions (DB, DW, DL, ASCII, PARITY)	Specifies various data within the program memory.
Symbol definition pseudo-instructions (EQU, SET)	Allocates constant to symbols (voluntary name) used within the source program.
Location counter control pseudo-instruction (ORG)	Sets the program counter.
External definition and reference pseudo-instructions (EXTERNAL, PUBLIC)	Allows for symbols and labels to be referenced between modules.
Source file insertion pseudo-instruction (INCLUDE) sap88 only	Inserts contents of other source files in voluntary places.
Assembly termination pseudo-instruction (END)	Specified the assembly end point.
Macro related pseudo-instructions (MACRO–ENDM, DEFINE, LOCAL, PURGE, UNDEF, IRP–ENDR, IRPC–ENDR, REPT–ENDR) sap88 only	Defines the macro statement.
Conditional assembly pseudo-instructions (IFC–ENDIF, IFDEF–ENDIF, IFNDEF–ENDIF) sap88 only	Assembly or skip can be set according to the definition of the symbol.
Output list control pseudo-instructions (LINENO, SUBTITLE, SKIP, NOSKIP, LIST, NOLIST, EJECT)	Controls the output to the assembly list file.

Unlike CPU instructions, pseudo-instructions do not directly compose of application programs upon executing control instructions to the sap88 and asm88.
The pseudo-instructions that can be used with this assembler are indicated above according to their functions. (Refer to Appendix B for more details.)

A.2.3 Assembly

This section will explain the method to assemble the assembly source file and the relocatable object file created by the process.

***Software used:* sap88, asm88**

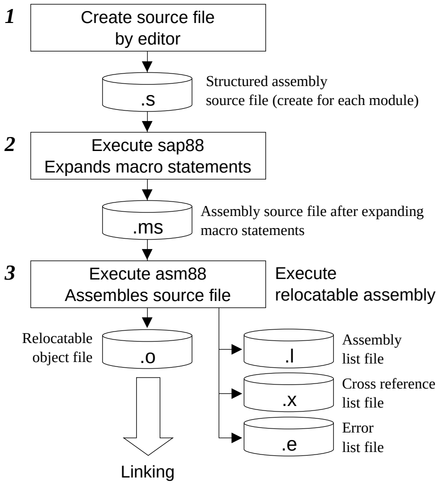

1
Create source file by editor
.s
Structured assembly source file (create for each module)
2
Execute sap88 Expands macro statements
.ms
Assembly source file after expanding macro statements
3
Execute asm88 Assembles source file
Execute relocatable assembly
Relocatable object file
.o
.l
Assembly list file
.x
Cross reference list file
.e
Error list file
Linking

Fig. A.2.3.1 Flowchart of relocatable assembly

A.2.3.1 Structured preprocessor (sap88)

This assembler system is composed of the sap88 structured preprocessor and asm88 cross assembler.

As indicated in Section A.2.2, the sap88 is responsible in putting the macro statement in mnemonic form. Since the asm88 cannot read the macro statement, assembly source files included these documents can not be directly input in the asm88 as a file.
The asm88 is the actual assembler responsible in converting the mnemonic language into machine language and assembling cannot be performed with sap88.
Therefore, there is a need to used both sap88 and asm88 for the structured assembly. It is advisable to process it through the sap88 even if the structured assembly is not required, since the process will not effect the source file.

The sap88 inputs an assembly source file with a ".s" extension and expands the macro statements. After that, the sap88 outputs a file for assembly. The name of the extension of the output file should be set as ".ms".

A.2.3.2 Cross assembler (asm88)

The asm88 cross assembler assemble the S1C88 Family CPU instructions and the pseudo-instructions of the asm88 and converts it into machine language.
The asm88 is compatible with the relocatable assembly.
The relocatable assembly creates relocatable object files (".o") that will be linked with other modules using a linker. The asm88 can input several assembly source files and thus allows for simultaneously assembly of several relocatable modules.
The asm88 can also output three lists, i.e., assembly list (".l"), error list (".e") and a cross reference list (".x") for the programmer.
The assembly list consists of the line number, target address, code that corresponds to the source and source statements. The line number is output in decimals, while the address and code are output in hexadecimals.
If in case an error takes place during assembling, an error list file containing the source file name, line number in which the error took place, error level and error message will be created. What's more, the assembly list file will also note the line in which the error took place with an asterisks "*" beside the line number. Processing will be continued regardless of an error message unless the error is fatal.
The relation of the symbol definition and reference within the file has been prepared to foster easy understanding depending on the cross reference list.
File management has been enhanced since they are prepared as separate files.

<Control of program and data memory>

This section will explain how to control the memory of the program and data.
The S1C88XXX memory map can be categorized in the program memory (ROM) for the program code and RAM and I/O memory for the data.
For example, even if a certain symbol is noted in a voluntary position in the assembly source file, the asm88 is not capable of determining whether this is within the program memory or data memory.
For this reason, there is a need to clarify which memory each line comes under by prior instruction through the section setting pseudo-instructions.
The following explains the section set methods for the relocatable assembly, and the asm88 process corresponding to the method.

Setting sections

The absolute address allocated within each module of the relocatable assembly will be specified or determined upon liking. Therefore, an absolute address cannot be specified within the assembly source file. A relative address specification can be made using an ORG pseudo-instruction, however, in this case, a standard for a relative address will be required. What's more, there is also a need to specify the segments of the program and data area for the asm88.
The entire program for this assembler is categorized into CODE and DATA. These basically indicate the following areas.

CODE section: Program data area written in the ROM
DATA section: Data memory area other than ROM

The asm88 is complete with a CODE and DATA pseudo-instruction to specify the section. The area can be set through descriptions in the assembly source file.

Specifying the CODE section

If a CODE pseudo-instruction is described within an assembly source file, the asm88 will assemble it to be allocated to the CODE section until the next DATA pseudo-instruction appears. The CODE pseudo-instruction can be used in several places within one module. The asm88 assumes the head of the CODE section within the module as relative address 0000H and will continuously realign them in the order that the CODE pseudo-instruction appears to consolidate it into one block. In other words, a CODE specification range of one module will be handled as one CODE section. (Refer to Figure A.2.3.2.1.)
The CODE section of each module is further consolidated as a whole by the linker. The linker will link in sectional units in accordance with the bank control within the program memory area.
The CODE section consists of CODE sections with one or multiple modules and the maximum size is limited to 32K bytes as one bank is. (Details on section control will be explained in "A.2.4.2 Section control".) Therefore, the programmer must be careful not to use more than 32K bytes in the code when creating a module. The capacity of the CODE section can be verified by using the -ROM# flag when starting-up the asm88. Use of this feature is advised. For example, when flag specification for "-ROM 32768" is performed, an error message will be displayed if a CODE section of one module exceeds 32K bytes.

Specifying the DATA section

If a DATA pseudo-instruction is described within an assembly source file, the asm88 will assemble it to be allocated to the DATA section until the next CODE pseudo-instruction appears. The DATA pseudo-instruction can be used in several places within one module. The asm88 assumes the head of the DATA section within the module as relative address 0000H and will continuously realign them in the order that the DATA pseudo-instruction appears to consolidate it into one block. In other words, a DATA specification range of one module will be handled as one DATA section. (Refer to Figure A.2.3.2.1.)
The DATA section of each module is further consolidated as a whole by the linker. The linker will link in sectional units in accordance with the page control within the data memory area.

The DATA section consists of DATA sections with one or multiple modules and the maximum size is limited to 64K bytes as one page is. (Details on section control will be explained in "A.2.4.2 Section control".) Therefore, the programmer must be careful not to use more than 64K bytes in the code when creating a module. The capacity of the DATA section can be verified by using the -RAM# flag when starting-up the asm88. Use of this feature is advised.
For example, when flag specification for "-RAM 65535" is performed, an error message will be displayed if a DATA section of one module exceeds 64K bytes.

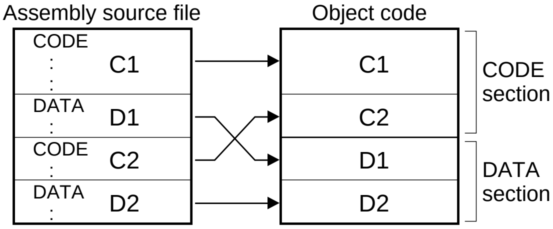


Fig. A.2.3.2.1 CODE section and DATA section

Note
If either the CODE pseudo-instruction or DATA pseudo-instruction is missing during relocatable assembling the operation will result in an error. For this reason, it is important that the CODE pseudo-instruction is used for the program memory and the DATA pseudo-instruction is used for the data memory.

A.2.3.3 Starting sap88 and asm88

<sap88 operation procedure>

(1) Set the directory in which the structured assembly source file (.s) is presented as the current drive.

(2) Start-up the sap88 with the next format.

sap88_[flag]_input file↵

_ indicates a space key input.
↵ indicates a return key input.

The following indicates the flag used for batch processing of relocatable assembly (ra88.bat).

Flag	Description
-o <file name>	Specify the file name that is output. (Specify ".ms" as the extension of the file to be output.) If this flag is omitted it will be processed as a standard output.

Refer to Appendix C for information on other flags.

Example: `C:\USER>c:\EPSON\sap88 -o sample.ms sample.s↵`

Inputs the assembly source file "sample.s" created in the sub-directory USER of drive C and then creates assembly source file "sample.ms" to be input in asm88 in the same directory as the input file. If the PATH to sap88 is set, then there is not need to specify the path before sap88.

Refer to Section "A.2.3.9 Example of assembly execution" for more information on I/O files and messages displayed.

<asm88 operation procedure>

(1) Set the directory in which the assembly source file (.ms) created with the sap88 exists as the current drive.

(2) Start-up the asm88 with the next format.

asm88_[flag]_input file↵

_ indicates a space key input.
↵ indicates a return key input.
Flag can be omitted.

The following indicates the flags used for batch processing of relocatable assembly (ra88.bat).

Flag	Description
-ROM#	Specify the ROM capacity in byte units. It is especially useful during relocatable assembling and is used to verify the size of the CODE area.
-RAM#	Specify the RAM capacity in byte units. It is especially useful during relocatable assembling and is used to verify the size of the DATA area.

Refer to Appendix C for more information on other flags.

Example 1: When continuously assembling several assembly source files through relocatable assembly.

```
C:\USER>c:\EPSON\asm88 sample1.ms sample2.ms↵
```

Inputs the assembly source files "sample1.ms" and "sample2.ms" created in the sub-directory USER of drive C and starts the relocatable assembly process. Then creates the relocatable object files "sample1.o" and "sample2.o" in the same directory as the input file.
At the same time, the assembly list files "sample1.l" and "sample2.l", cross reference list files "sample1.x" and "sample2.x", and error list files "sample1.e" and "sample2.e" will also be created in the same directory.
If the PATH to asm88 is set, then there is not need to specify the path before asm88.

Example 2: Assembling with the relocatable assembler, including the verification of the ROM and RAM capacity.

```
C:\USER>c:\EPSON\asm88 -ROM 32768 -RAM 65536 sample.ms↵
```

Inputs assembly source file "sample.ms" created within the sub-directory USER of drive C and starts relocatable assembly. Then creates the relocatable object file "sample.o" in the same directory as the input file.
At the same time, creates the assembly list file "sample.l", cross reference list file "sample.x" and error list file "sample.e" in the same directory.
The capacity of the CODE and DATA sections will be verified during assembling with the -ROM and -RAM flags. An error will result in this case when the CODE exceeds 32K bytes and the DATA exceeds 64K bytes.
If the PATH to asm88 is set, then there is not need to specify the path before asm88.

Refer to Section "A.2.3.9 Example of assembly execution" for more information on I/O files and messages displayed.

A.2.3.4 Batch processing for relocatable assembly (ra88.bat)

The start-up procedures for sap88 and asm88 were already discussed in the earlier section, however, it must be further noted that these can be batch processed by consolidating them into a batch file.
The batch file can voluntarily created by the user, however, since this package contains batch file, i.e., ra88.bat for relocatable assembly, the following will introduce the contents of the batch file and how to use them. This batch file can be used for general processing purposes. Use it advantageously by customizing the flag settings, etc. as needed.
Figure A.2.3.4.1 shows the ra88.bat processing flow.

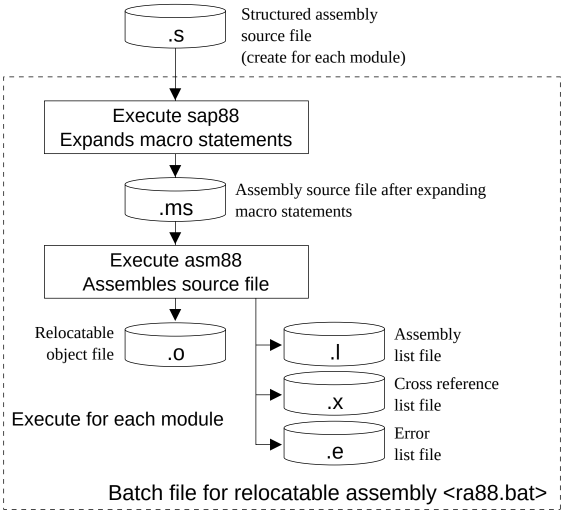

.s
Structured assembly source file (create for each module)
Execute sap88
Expands macro statements
.ms
Assembly source file after expanding macro statements
Execute asm88
Assembles source file
Relocatable object file
.o
.l
Assembly list file
.x
Cross reference list file
.e
Error list file
Execute for each module
Batch file for relocatable assembly <ra88.bat>

Fig. A.2.3.4.1 ra88.bat processing flow

<Outline of process>

The ra88.bat inputs the specified assembly source file and then executes sap88 and asm88, respectively to perform relocatable assembly to create a relocatable object file. Since the sap88 does not permit input of multiple assembly source files, it is limited to assembly per module other than when several structured assembly source files are read with the INCLUDE pseudo-instruction of the sap88.

<Input/output files>

The following indicates the input / output files of the ra88.bat.

Input file

Structured assembly source file (relocatable): file_name.s
This is a structured assembly source file (relocatable) created with an editor .

Output files

1. **Assembly source file: file_name.ms**
   An assembly source file in which macros are expanded will be output.
2. **Relocatable object file: file_name.o**
   This is a binary file that has been converted in machine language that can be reallocated through relocatable assembly. (This is also the file that inputs the lk88.bat batch file to perform linking.)
3. **Assembly list file: file_name.l**
   This is the file output as a list that corresponds to each source statement when the machine language and the relocatable address (the head of the CODE or the DATA section is assumed as relative address 000000H) converted with the assembler.
4. **Cross reference list file: file_name.x**
   This is the address list that contains the definition and references of symbols.
5. **Error list file: file_name.e**
   This is the list of error taking place during assembling.

<Operation procedure>

(1) Set the directory in which the structured assembly source file (.s) is presented as the current drive.

(2) Start-up the ra88.bat with the next format.

`ra88`_file name↵

_ indicates a space key input.
↵ indicates a return key input.

Do not input the extensions of file name. It is fixed on the ".s" extension.

Example: `C:\USER>c:\EPSON\ra88 sample`↵

Inputs structured assembly source file "sample.s" created within the sub-directory USER of drive C and starts relocatable assembly. Then creates the following files in the same directory as the input file.

```
sample.ms, sample.o, sample.l, sample.x, sample.e
```

If the PATH to ra88 is set, then there is not need to specify the path before ra88.

Refer to Section "A.2.3.9 Example of assembly execution" for more information on I/O files and messages displayed.

Customizing ra88.bat

*<**Customizing ra88.bat execution parameters**>*

Since the ra88.bat controls the program execution, it has a execution parameter customization field within it. General parameters are temporarily described in the default position, however, it is advised that the program is customized in accordance with the user's development method.

1. **Setting the ROM capacity (Verification of the size of the CODE section)**

set rom = 32768 : The capacity of the ROM of the CODE section that locates errors will be specified in bytes. (default capacity 32768 = 32K bytes)

2. **Setting the RAM capacity (Verification of the size of the DATA section)**

set ram = 65536 : The capacity of the RAM of the DATA section that locates errors will be specified in bytes. (default capacity 65536 = 64K bytes)

Note: There are basically no error checks made on these parameter settings, therefore, do not set the parameter with settings other than those specified.

*<**Customizing ra88.bat execution command**>*

The ra88.bat has the following command line upon execution of the program. Customize these command lines if a flag without a default setting is to be used.

sap88

```
%drv%sap88 -o %1.ms %1.s
```

asm88

```
%drv%asm88 -ROM %rom% -RAM %ram% %1.ms
```

The %drv% is a path that locates the execution command of the ra88.bat. For this reason, it can not be altered and neither can the SET statement that is defined be altered. The %1 is a file name that is input from the command line.

The following indicates the ra88.bat program source list and the message list of the ra88.bat. Refer to it upon customizing the program.

ra88.bat program source list

```
echo off
rem **************************************************************************
rem *E0C88 Family Auto Relocatable Assemble Execution Utility
rem *                                  (Ver. X.XX)
rem *                            Copyright(C) SEIKO EPSON CORP. 1993–1996
rem **************************************************************************
rem * customized parameter information
rem *rom=*  * : rom capacity(32768 max.)
rem *ram=*  * : ram capacity(65536 max.)
rem **************************************************************************
rem ********** customized parameter area (default) **********
rem *  caution : customized parameters value do not check,therefore
rem *            please be carefully when you set
rem **********
set rom=32768                         ← Setting the capacity of the ROM
set ram=65536                         ← Setting the capacity of the RAM

rem ********** command searching path **********
rem set drv=c:\
rem ********************************************* ****************************
rem *main program
rem *      if you want to use another option(s), please append
rem *      option flag(s) at command line.
rem **************************************************************************
:start
     echo E0C88 Family Auto Relocatable Assemble Execution Utility Ver. X.XX
     echo Copyright (C) SEIKO EPSON CORP. 1993–1996

           if "%1"=="" goto usage
:error_chk
           if not exist %drv%nul goto exit04
           if not exist %1.s goto exit05
           if not exist %drv%sap88.exe goto exit06
           if not exist %drv%asm88.exe goto exit07

rem (sap88)
:sap88
%drv%sap88 -o %1.ms %1.s                                  ← Start-up command of sap88
           if errorlevel 1 goto exit01

rem (asm88)
:asm88
%drv%asm88 -ROM %rom% -RAM %ram% %1.ms                    ← Start-up command of asm88
           if errorlevel 1 goto exit02
                 goto end

     :usage
     echo usage : ra88 needs [input file_name]
                 goto skip
     :exit01
     echo Error stop at %drv%sap88.exe
                 goto skip
     :exit02
     echo Error stop at %drv%asm88.exe
                 goto skip
     :exit03
     echo Cannot find %drv% installed E0C88 dev. tools directory
                 goto skip
     :exit04
     echo Cannot find input file
                 goto skip
     :exit05
     echo Cannot find %drv%sap88.exe
                 goto skip
     :exit06
     echo Cannot find %drv%asm88.exe
                 goto skip
     :end
     echo ra88.bat utility has been successfully executed.
     :skip
     set rom=
     set ram=
     set drv=
```

User customization field

Note: There are basically no error checks made on these parameter settings, therefore, do not set the parameter with settings other than those specified.

The drv is a path that locates the execution command of the ra88.bat. It is set to root directory by default. Customize it if necessary.

Message list

1. Start-up message

```
E0C88 Family Auto Relocatable Assemble Execution Utility Ver. X.XX
Copyright (C) SEIKO EPSON CORP. 1993–1996
```

2. Message when terminated normally

```
ra88.bat utility has been successfully executed.
```

3. Error message

Error message	Explanation
usage : ra88 needs [input file_name]	Usage output.
Error stop at [drive and path name] sap88.exe	Error occurred in sap88.
Error stop at [drive and path name] asm88.exe	Error occurred in asm88.
Cannot find [drive and path name] installed E0C88 dev. tools directory	Cannot find [drive or path] in which the S1C88 Family software tools is installed.
Cannot find input file	Cannot find aa88.bat input file (.s).
Cannot find [drive and path name] sap88.exe	Cannot find sap88.
Cannot find [drive and path name] asm88.exe	Cannot find asm88.

Note: The following operations will be stopped when an error occurs.

<Precautions upon using the batch file>

(1) Some of the messages displayed during batch processing is automatically generated through the MS-DOS / PC-DOS batch processing function and command. For this reason, it may be placed under MS-DOS / PC-DOS control when an error occurs and thus force the batch processing to be interrupted.

(2) When an error occurs, the following procedures do not automatically continue. However, it may not be controllable as noted in reason (1) indicated above.

(3) The ra88.bat and the lk88.bat (mentioned hereafter) employ the MS-DOS / PC-DOS COPY command in addition to S1C88 Family tools.
For this reason, it is requested that the COPY command is operable, by setting the PATH, when executing the batch file.

(4) The execution parameters (user customization field) of the batch file basically do not locate parameter setting errors. Therefore, do not set the parameters other than specified.

(5) An MS-DOS / PC-DOS environment variable will be used to execute the batch file, therefore, the size of the environment variable should be allocated with as much space as possible using the CONFIG.SYS.

A.2.3.5 Relocatable object file

The relocatable object file is a binary file that is created through the relocatable assembly of the asm88. Other than when -o flag is specified the file name that is created will be the same file name input with the asm88 and the extension will be ".o".
This file consists of header information and symbol tables required for reallocation using the linker, in addition to the object (machine language) code.

A.2.3.6 Assembly list file

The assembly list file is an ASCII file added with an object code (hexadecimal) and code address (hexadecimal) in the assembly source file input in the asm88. It is created through asm88 assembly. Each page will have a header with the file name and date that the file is created.

The file name that is created will be the same as the file name input via the asm88 other than when -o flag is specified. The extension will be ".l".
The assembly list file consists of the following items:
LINE .... The consecutive line number from the beginning.
ADDRESS .... This refers to the target address of the object code.
CODE.............................. This is the object (machine language) code that corresponds to the source statement in the same line.
SOURCE STATEMENT .. This is the assembly source input in the asm88.

When relocatable assembly is performed, the code address will be a relative address from the beginning of the CODE section. Similarly, the address of the data area is a relative address from the beginning of the DATA section.
If an error is occurred, an asterisks "*" will be placed at the beginning of the line in which the error occurred.

The output of assembly list file can be controlled with the following asm88 pseudo-instructions and flag specifications upon start-up.

Output list control pseudo-instructions

Pseudo-instruction	Description
LINENO	Changes the line number (LINE) to the voluntary value.
SUBTITLE	Inserts the subtitle line that is voluntarily set after the column explanation line.
SKIP	If any line of the code exceeds 5 bytes through ASCII, DB or DW data settings, the exceeding portion will not be output. (default setting.)
NOSKIP	Outputs all codes by canceling the SKIP setting.
LIST	The following lines are output in a list when the NOLIST setting is canceled.
NOLIST	Prevents output of the list from the line after the pseudo-instruction.
EIECT	Adds a involuntary page break.

Refer to Appendix B for details of the pseudo-instructions.

Start-up flag

Refer to Appendix C for details of the flag.

Flag	Description
`-l`	Prevents creation of an assembly list file.

A.2.3.7 Cross reference list

The cross reference list file is created through asm88 assembly with an ASCII file. This ASCII file is defined within the module or contains a list of reference symbols.
The name of the file created will be the same as the file name input with the asm88 other than when specifying -o flag. The extension will be ".x".

The output format of the cross reference list file is as follows.

R SYMBOL A VALUE LINE No. INFORMATION

R Reference definition
G: Global
L: Local

SYMBOL Symbol name (maximum 15 characters)

A Attribute
L: Label
C: Constant
V: Variable
U: Undefined within the module
VALUE Symbol value (6 digit, hexadecimal expression)

LINE No. INFORMATION

This is a list in which the symbol is defined or referenced line numbers. They are output as follows.
lineno* lineno lineno . . . . lineno
lineno*: The line number in which the target symbol is defined.
lineno: The line number in which the target symbol is referenced.

The LINE No. INFORMATION can consist up to a maximum of 12 line numbers.

The following page header will be output at the head of each page.

The numeric labels are temporary labels. The same name can be used if they are outside the range defined by the general label. It will not be output on the cross reference list. (Refer to Appendix B for the numeric labels.)

The cross reference list file can prohibit output using the -x flag of the asm88.

Example of cross reference list

```
CROSS REFERENCE TABLE OF asm88  error.x  1993-06-07  17:28    PAGE  1

L  delay          L  000100H       5*     14     15
L  delay_00       L  000103H       7*      9
L  delay_3times   L  000107H      13*
```

A.2.3.8 Error list

The errors generated during asm88 assembling will be output as an error list file.
The name of the file created will be the same as the file name input with the asm88 other than when specifying -o flag. The extension will be ".e".

The output format of the error list is as indicated below.

SOURCE FILE LINE No.: ERROR LEVEL: ERROR MESSAGE

SOURCE FILE Source file name

LINE No. Line number in which the error occurred

ERROR LEVEL Level of error
Warning This is a warning and does not affect the output object.
Severe This is a general error. The output object will be invalid.
Fatal This is a fatal error. Assembly will be interrupted. Fatal errors are displayed on the CRT without output of an error list file.

ERROR MESSAGE Error content

Refer to Appendix C for the error messages of the asm88.

Example of error list

```
error.s 16:  Severe:   delay not defined
```

When an error is not generated, nothing will be output in the error list file.

A.2.3.9 Example of assembly execution

The following shows example of the assembly execution.

Messages when ra88.bat (relocatable assembly) is executed

```
C:\USER>c:\EPSON\ra88 sample↵

C:\USER>echo off
E0C88 Family Auto Relocatable Assemble Execution Utility Ver. X.XX
Copyright (C) SEIKO EPSON CORP. 1993–1996
sap88 Structured Assembler Preprocessor Version X.XX

Copyright (c) 1993 by Advanced Data Controls, Corp.
Licenced to SEIKO EPSON CORP.
asm88 Cross Assembler Version X.XX

Copyright (c) 1993 by Advanced Data Controls, Corp.
Licenced to SEIKO EPSON CORP.

   9 Symbol(s) Used

   0 Warning Error(s)
   0 Severe Error(s)
ra88.bat utility has been successfully executed.
C:\USER>
```

A.2.4 Link

This section will explain the linking operations of relocatable modules.

***Software used:* link88**

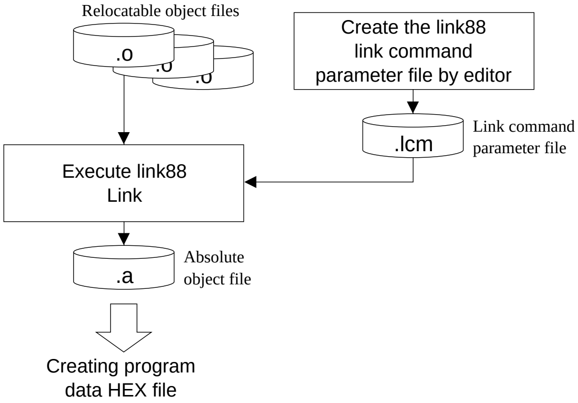


Fig. A.2.4.1 Link processing flow

A.2.4.1 Linking modules

The object codes of each module created with the relocatable assembly of the asm88 is not specified to be located in a certain portion of the ROM. The allocation address is determined by how each modules are linked. The link88 linker is the tool used for linking operations.

When linking is successfully performed the relative address for the external reference label that was undeclared up to this point will be declared and thus, create an absolute object file (.a) that consolidates all modules into one file. By processing this absolute object file with the binary / HEX converter hex88, as indicated in Section A.2.5, the program data HEX file to be used to create the program mask data or to debug the hardware will be created.

A.2.4.2 Section control

The S1C88 Family has a 24-bit width address space (maximum of 16M bytes). By using the topmost 8-bit for register control using the code bank register (CB), expand page register (EP, XP, YP) and others, the address space can be allocated into a 32K-byte bank (CODE) or 64K-byte page (DATA) unit. Access performance can be improved within those ranges. By rewriting the content of the register, the user will have access of a voluntary bank or page from a voluntary bank. As a result, large programs and data bases can easily be controlled. However, the bank and page will not automatically be changed with the execution of the program and thus it must be set in accordance with the program specifications.
Therefore a program as described in linear programs can not be created in the 16M-byte address space. This indicates that multiple modules can not simply be linked.
For this reason, the link88 employs a multi-section method to resolve this problem by allocate voluntary modules in voluntary addresses.
Allocation in this method is undertaken by making it possible to specify addresses for block units referred to as sections.
The section is categorized into a CODE section in which the allocation site is the ROM and the DATA section which is the data memory. To resolve the aforementioned bank and page problems, the size of one CODE section can consist of up to 32K bytes and the size of one DATA section is limited to 64K bytes. It is important to note that this size is based on the fact that they are not allocated over the bank or page limit. If in case they are allocated in the middle of a bank or page, the size will be limited to the remaining size.

To create an object code for the desired multi-section using the section method, the user must define the section and supply address information on the allocation of the section to allocate the address.
The section is defined by using the linker's secondary flag (flag used to define section) +code and +data and the -p flag is used to allocate the address.
Up to a maximum of 255 sections can be defined with one link.

<Example of section definition>

Let's look at the section definition procedures through a simple example.
First, the method to actualize a memory mapping as indicated in Figure A.2.4.2.1 will be explained.
It will be assumed that "prg1.s" describing C1 and D1, "prg2.s" describing C2 and "prg3.s" describing C3 is assembled and then each respective relocatable object file "prg1.o", "prg2.o" and "prg3.o" is created.
In this case, C indicates the CODE section and D indicates the DATA section.

The flag to link88 can be specified through input redirect operations.
When the following flag specification is performed and a link command parameter file (filename.1cm) that is used to allocate the address and define the section is created following by executing link88<filename.lcm, a memory mapping as indicated in Figure A.2.4.2.1 will be created.

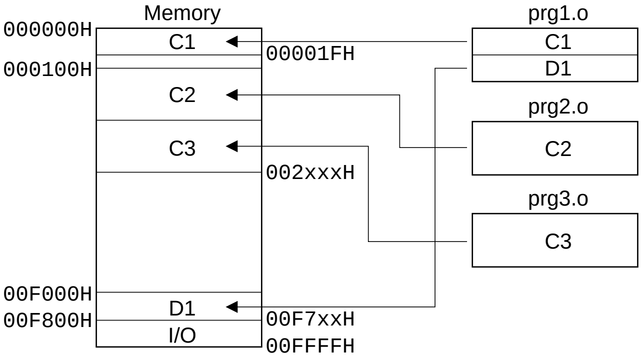


Fig. A.2.4.2.1 Memory mapping example

Contents of the file transferred to link88 (link88<filename.lcm)

```
-o prg.a                 ...(1)
+code -p0x000000         ...(2)
+data -p0x00f000         ...(3)
prg1.o                   ...(4)
+code -p0x000100         ...(5)
prg2.o prg3.0            ...(6)
```

(1) Specifies the absolute object file that is output with the -o flag.
(2) Defines the CODE section that starts with a physical address from 000000H.
(3) Defines the DATA section that starts with a physical address from 00F000H.
(4) Allocates "prg1.o" to the sections defined in (2) and (3) indicated above.
In this case, the contents of the CODE section C1 in "prg1.o" will be allocated from the beginning of the CODE section defined in (2) and the contents of the DATA section D1 will be allocated at the head of the DATA section defined in (3).
(5) Defines the CODE section that starts with a physical address from 000100H. This CODE section is different from the CODE section defined in (2). The CODE section (2) will be completed when a new section is defined at this point.
(6) The "prg2.o" CODE section of C2, and "prg3.o" CODE section C3 will be continuously be allocated in respective order.
In this example, "prg2.o" and "prg3.o" does not have a DATA section. However, if there is a DATA section then it will be allocated from the address following D1 of the DATA section defined in (3).

There are three sections defined and linked in this example as indicated above. When the link is successful an absolute object file named "prg.a" will be created.
Multiple modules can be allocated in these sections defined as long as it is within the allowable capacity limit. What's more, multiple sections can be allocated within one bank as well.

<Allocation address and relocation of section>

As indicated in the earlier example, the -p flag determines the physical start address of the section defined immediately before operations.
Let's say, for example, the following settings are made for a certain section.

```
-p 0x10000
```

The start address of this section will physically be 10000H. The CODE section will be specified at the head of bank 2 and the DATA section will be specified at the head of page 1.
The following allocation (reallocation of address information) will be performed for a symbol if a symbol is defined to be positioned from the head of this section to the 1234H offset and that symbol is used to reference that address.

(1) When handled as data memory (symbol name will be indicated as "SYMBOL".)

Operand		*Relocate value*
#SYMBOL	→	#1234H
[SYMBOL]	→	[1234H]
#POD SYMBOL	→	01H
#LOD SYMBOL	→	1234H
#HIGH SYMBOL	→	12H
#LOW SYMBOL	→	34H
[BR:LOW SYMBOL]	→	[BR:34H]

(2) When handled as program memory (symbol name will be indicated as "LABEL".)

Operand		*Relocate value*
#BOC LABEL	→	02H
#LOC LABEL	→	9234H

A relative valued in accordance with the address that allocated by the branch instruction will be calculated and set for PC relative branch instructions like "JRL LABEL".

The section start address, in the above example, was specified at the head of the bank or page, however, specifications can be made for it to start in the middle of a bank or page, as indicated below.

```
-p 0x15000
```

In this case the start address will physically be 15000H and have a 5000H offset from the head of the bank or page. The link88 relocates each symbol based on the physical address, therefore, such offsets will also be properly processed.
All symbol information after reallocation will be recorded in the absolute object file. A list of these symbols can be created using the rel88 symbol information generating utility. Refer to Section A.2.6.1, "Creating symbol information (rel88)" for more information on rel88 operations.

A.2.4.3 Module allocation information

As indicated in the example of section definition mentioned earlier, section definitions and command lines that specify files can be handed over to the link88 through the input redirect function.
The number of modules are limited and the link is simple, as indicated in the example, it will be possible to create a file similar to that indicated in the example and directly input into the link88.
There will be need to be conscious about the memory efficiency when increasing the number of modules. One CODE section is limited to 32K bytes and the DATA section is limited to 64K bytes. Thus, it will be necessary to allocate each module so that it does not exceed the limit. It will be necessary to give consideration to the combination of modules in each section upon allocation. Otherwise, there will be more unused memory area and thus, require unnecessary memory extension.

A.2.4.4 Starting link88

<Operations of link88>

(1) Set the directory in which the relocatable object files (.o) to be linked and the link command parameter file (.lcm) including link88 command line created with the editor are existed as the current drive.

(2) Start-up the link88 with the next format.

link88_<_link command parameter file name↵

_ indicates a space key input.
↵ indicates a return key input.

Regardless of the input redirect function, the link command parameter file can directly be input in the command line. The procedures will be omitted since it is not practical. Refer to Appendix B for more information on formatting.
Details on the flags that compose the command line will also be omitted.
Refer to Appendix B for details of the flags.

Example: Performing linking through the link command parameter file (.lcm)

```
C:\USER>c:\EPSON\link88 < sample.lcm↵
```

Use the link command parameter file "sample.lcm" created in the USER of the sub-directory of drive C as the input redirect function to start-up link88 and perform linking. The name of the absolute object file specified in the link command parameter file will be created in the same directory as the input file. If the PATH to link88 is set, then there is not need to specify the path before link88.

Refer to Section A.2.4.2 for the link command parameter file.

A.2.4.5 Batch processing for linking (lk88.bat)

As so with the assembler, this package contains the lk88.bat batch file for linking.
This batch file is prepared so that it can process the procedures from linking to creation of the program data HEX file. (Details on processing procedures after linking will be noted later.)
Figure A.2.4.5.1 shows the processing flow of lk88.bat.

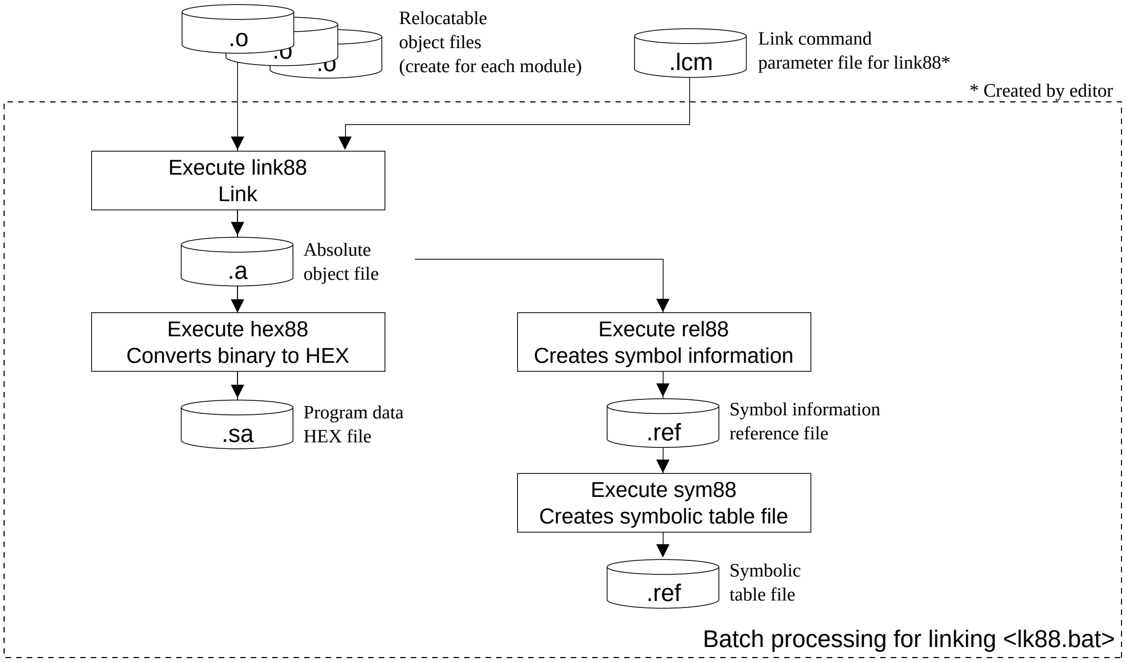


Fig. A.2.4.5.1 lk88.bat processing flow

<Outline of processing procedures>

The lk88.bat reads the link command parameter file for the link88 and executes linking operations. When an absolute object file is created using the link88, it will then use the rel88 symbol information generator. After reallocation operations are complete a symbolic table information file will be created. After that, the sym88 will be executed to generate a symbolic table file that is necessary for symbolic debugging using the ICE.
Then a program data HEX file will be created with the hex88 binary / HEX converter from the absolute object file.

<Input/output files>

Input files

1. **Link command parameter file: file_name.lcm**
   This is a command parameter file for the link88. It indicates the information to reallocate the relocatable object of the S1C88 memory space.

2. **Relocatable object file: file_name.o**
   This is a relocatable file in machine language that can be output through relocatable assembly with the cross assembler.

Output files

1. **Absolute object file: file_name.a**
   This is the multi-section object file created with the linker.

2. **Program data HEX file: file_name.sa**
   This is a Motorola S2 format ASCII record file consisting of an absolute object file that was converted with the binary / HEX converter.

3. **Symbol information reference file: file_name.ref**
   This is the symbol information reference file of the absolute object file that was reallocated by the physical address.

4. **Symbolic table file: file_name.sy**
   This file contains symbol names and the address list information for symbolic debugging.

<Operation procedure>

(1) Set the directory including the relocatable object files (.o) to be linked as the current drive.
Put the command parameter file handed over to the link88 in the same directory.

(2) Start-up the lk88 with the next format.

```
lk88↵
```

↵ indicates a return key input.

Example: `C:\USER>c:\EPSON\lk88↵`

Use the link command parameter file "sample.lcm" created in the USER of the sub-directory of drive C to start batch processing.
Batch processing will create the absolute object file (.a), symbol information reference file (.ref), program data HEX file (.sa) and symbolic table file (.sy) in the same directory as the input file.
If the PATH to lk88 is set, then there is not need to specify the path before lk88.

Customizing lk88.bat

<*Customizing lk88.bat execution parameters*>

Since the lk88.bat controls the program execution, it has a execution parameter customization field within it. General parameters are temporarily described in the default position. Always customize the batch files according to your development method since the parameter will vary depending on your application style.

1. Parameter file name to be input

set parfn = file_name : Link command parameter file name (.lcm) input to link88

2. Output file name

set outfn = file_name : File name of absolute object file and program data HEX file

3. Use of the symbol information generator (rel88)

set rel88 = y : rel88 is used (default)
A symbol information reference file (.ref) will be created.
= n : rel88 is not used.

4. Use of +sec flag (information on individual section) of the symbol information generator (rel88)

set secf = y : +sec flag is added to rel88 (default)
= n : +sec flag is not added to rel88

Note: This parameter will be ignored when rel88 is not used.

Note: There are basically no error checks made on these parameter settings, therefore, do not set the parameter with settings other than those specified.

<*Customizing lk88.bat execution command*>

The lk88.bat has the following command line upon execution of the program. Customize these command lines if a flag without a default setting is to be used.

link88

```
%drv%link88<%parfn%.lcm
```

rel88 (when +sec flag is used)

```
%drv%rel88 -v +sec
%outfn%.a>%outfn%.ref
```

rel88 (when +sec flag is not used)

```
%drv%rel88 -v %outfn%.a>%outfn%.ref
```

hex88

```
%drv%hex88 -o %outfn%.sa %outfn%.a
```

sym88

```
%drv%sym88 %outfn%.ref
```

The %drv% is a path that locates the execution command of the lk88.bat. For this reason, it can not be altered and neither can the SET statement that is defined be altered.
Use the same name for the customized parameter outfn as the name described in the link command parameter (.lcm).

The following indicates the lk88.bat program source list and the message list of the lk88.bat. Refer to it upon customizing the program.

lk88.bat program source list

```
echo off
rem **************************************************************************
rem *      E0C88 Family Auto Link Execution Utility
rem *                                   (Ver. X.XX)
rem *                             Copyright(C) SEIKO EPSON CORP. 1993–1996
rem **************************************************************************
rem * customized parameter information
rem *  parfn=         :  input parameter file_name
rem *                    (file_name_lcm) for link88.exe     i.e. c8316xxx.lcm
rem *  outfn=         :  output file_name which is written
rem *                    in the input parameter file_name   i.e. c8316xxx
rem *  rel=y  y : use rel88 for absolute symbol map generation
rem *     =n  n : do not use rel88
rem *
rem *  secf=y y : show physical address and module size with absolute
rem *             symbolic table after link procedure
rem *      =n n : do not show physical address and module size just
rem *             symbolic table after link procedure
rem **************************************************************************
rem ********** customized parameter area (default) **********              User customization field
rem *  caution : customized parameters value do not check, therefore
rem *        please be carefully when you set                              Note: There are basically no
rem *********                                                             error checks made on these
set parfn=sample         ←  Name of link command parameter file to be input   parameter settings, therefore,
set outfn=sample         ←  Name of file to be output                          do not set the parameter with
set rel=y                ←  Use of not of rel88                                settings other than those
set secf=y               ←  Use or not of the rel88 + sec flag                 specified.

rem ********** command searching path **********   The drv is a path that locates the execution command
rem set drv=c:\                                    of the lk88.bat. It is set to root directory by default.
                                                   Customize it if necessary.
rem **************************************************************************
rem * main program
rem *       if you want to use another option(s), please append
rem *       option flag(s) at command line
rem **************************************************************************
:start
     echo E0C88 Family Auto Link Execution Utility Ver. X.XX
     echo Copyright (C) SEIKO EPSON CORP. 1993–1996

:error_chk
          if not exist %drv%nul goto exit05
          if not exist %parfn%.lcm goto exit06
          :chk00
          if not exist %drv%link88.exe goto exit07
          if not exist %drv%rel88.exe goto exit08
          if not exist %drv%hex88.exe goto exit09
          if not exist %drv%sym88.exe goto exit10

:link88
%drv%link88<%parfn%.lcm                                ←  Start-up command of link88
          if errorlevel 1 goto exit01

rem (rel88 no sec option)
:rel88_01
          if "%rel%"=="n" goto hex88
          if "%secf%"=="y" goto rel88_02
%drv%rel88 -v %outfn%.a>%outfn%.ref                    ←  Start-up command of rel88 (no +sec flag)
          if errorlevel 1 goto exit02
                goto hex88
rem (rel88 with sec option)
:rel88_02
%drv%rel88 -v +sec %outfn%.a>%outfn%.ref               ←  Start-up command of rel88 (with +sec flag)
          if errorlevel 1 goto exit02

:hex88
%drv%hex88 -o %outfn%.sa %outfn%.a                     ←  Start-up command of hex88
          if errorlevel 1 goto exit03
```

```
:sym88
%drv%sym88 %outfn%.ref                                  ← Start-up command of sym88
          if errorlevel 1 goto exit04
                goto end

     :exit01
     echo Error stop at %drv%link88.exe
          goto skip
     :exit02
     echo Error stop at %drv%rel88.exe
          goto skip
     :exit03
     echo Error stop at %drv%hex88.exe
          goto skip
     :exit04
     echo Error stop at %drv%sym88.exe
          goto skip
     :exit05
     echo Cannot find %drv% installed E0C88 dev. tools directory
          goto skip
     :exit06
     echo Cannot find %parfn% input parameter file
          goto skip
     :exit07
     echo Cannot find %drv%link88.exe
          goto skip
     :exit08
     echo Cannot find %drv%rel88.exe
          goto skip
     :exit09
     echo Cannot find %drv%hex88.exe
          goto skip
     :exit10
     echo Cannot find %drv%sym88.exe
:end
     echo lk88.bat utility has been successfully executed.
:skip
     set parfn=
     set outfn=
     set rel=
     set secf=
     set drv=
```

Message list

1. Start-up message

```
E0C88 Family Auto Link Execution Utility Ver. X.XX
Copyright (C) SEIKO EPSON CORP. 1993–1996
```

2. Message when terminated normally

```
lk88.bat utility has been successfully executed.
```

3. Error message

Error message	Explanation
Error stop at [drive and path name] link88.exe	Error occurred in link88.
Error stop at [drive and path name] rel88.exe	Error occurred in rel88.
Error stop at [drive and path name] hex88.exe	Error occurred in hex88.
Error stop at [drive and path name] sym88.exe	Error occurred in sym88.
Cannot find [drive and path name] installed E0C88 dev. tools directory	Cannot find [drive or path] in which the S1C88 Family software tools is installed.
Cannot find [file_name] input parameter file	Cannot find input parameter file (.lcm) that is used with the lk88.bat.
Cannot find [drive and path name] link88.exe	Cannot find link88.
Cannot find [drive and path name] rel88.exe	Cannot find rel88.
Cannot find [drive and path name] hex88.exe	Cannot find hex88.
Cannot find [drive and path name] sym88.exe	Cannot find sym88.

Note: The following operations will be stopped when an error occurs.

<Precautions upon using the batch file>

(1) Some of the messages displayed during batch processing is automatically generated through the MS-DOS / PC-DOS batch processing function and command. For this reason, it may be placed under MS-DOS / PC-DOS control when an error occurs and thus force the batch processing to be interrupted.

(2) When an error occurs, the following procedures do not automatically continue. However, it may not be controllable as noted in reason (1) indicated above.

(3) The execution parameters (user customization field) of the batch file basically do not locate parameter setting errors. Therefore, do not set the parameters other than specified.

(4) An MS-DOS / PC-DOS environment variable will be used to execute the batch file, therefore, the size of the environment variable should be allocated with as much space as possible using the CONFIG.SYS.

A.2.4.6 Absolute object file

The absolute object file is a binary file created by link88.
The name of the file name created will be the same as that specified with the -o flag.
The files come in a multi-section object format.
This file is composed of an object (machine language) code and various reallocation information.

A.2.4.7 Execution example of linking

The following shows examples of the lk88 execution.

```
C:\USER>c:\EPSON\lk88

C:\USER>echo off
E0C88 Family Auto Link Execution Utility Ver. X.XX
Copyright (C) SEIKO EPSON CORP. 1993–1996
link88 Linker Version X.XX

Copyright (c) 1993 by Advanced Data Controls, Corp.
Licenced to SEIKO EPSON CORP.
lk88.bat utility has been successfully executed.
C:\USER>
```

A.2.5 Creating Program Data HEX File

This section will explain the program data HEX file and how they can be created using the hex88 binary / HEX converter.

***Software used:* hex88**

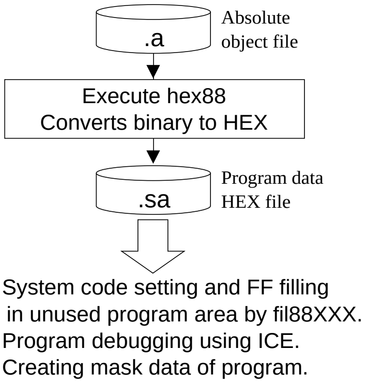

.a
Absolute object file
Execute hex88
Converts binary to HEX
.sa
Program data HEX file
• System code setting and FF filling in unused program area by fil88XXX.
• Program debugging using ICE.
• Creating mask data of program.

Fig. A.2.5.1 Program data HEX file generation flow

A.2.5.1 Program data HEX file

The program data HEX file is an ASCII file in which the binary object codes were converted in HEX data. The Motorola S2 format is generally employed at the HEX file format since the S1C88 Family has a 16M-byte address space. (Refer to Section A.2.5.3 for more information.)
This file will be required to mask program data or to debug program with the ICE.
When development is undertaken for modules according to relocatable assembly, the absolute object file created by the linker will be converted into HEX data through the hex88 binary / HEX converter and then create a program data HEX file.
The program data HEX file created through such procedures will set system codes according to each model and fill FF of the unused built-in ROM area. This is done with the fil88XXX software tool according to the model.

A.2.5.2 Creating program data HEX file using hex88

The following indicates the direction in creating a program data HEX file using the hex88.

(1) Set the directory in which the absolute object file (.a) is presented as the current drive.

(2) Start-up the hex88 with the next format.

hex88_[flag]_file name↵

_ indicates a space key input.
↵ indicates a return key input.

The following indicates the flag employed during batch processing (lk88.bat) of links.

Flag	Description
-o <file name>	Specify the file name that is output. (Specify ".sa" as the extension of the file to be output.) If this flag is omitted it will be processed as a standard output.

Example: Converting sample.a to create program data HEX file

```
C:\USER>c:\EPSON\hex88 -o sample.sa sample.a↵
```

"sample.sa" will be created in the same directory as the input file by inputting the absolute object file "sample.a" created in the USER of the sub-directory of drive C and converting it into HEX data format.
If the PATH to hex88 is set, then there is not need to specify the path before hex88.

The batch file can allow for hex88 to be executed after linking. Refer to Section "A.2.4.5 Batch processing for linking (lk88.bat)" for more details on such batch processing methods.

A.2.5.3 Motorola S2 format

The HEX file in the Motorola S2 format is a collection of records composed of fields like the following.

<S FIELD><COUNT><ADDR><DATA BYTES><CHECKSUM>

All information will be indicated in hexadecimal pairs and each pair will indicate a 1-byte value.

<S FIELD> Indicates the format of that line. "S2" will appear in this field.

<COUNT> Indicates the total number of bytes of <ADDR>, <DATA BYTES> and <CHECKSUM> in hexadecimal form.

<ADDR> Indicates the address of the first data byte of that line.
The <ADDR> field in S2 format is 3-byte.

<DATA BYTES> Data will be allocated in 1 byte units in order of the increase in address. This field generally includes the 32-byte (maximum) data.

<CHECKSUM> This is the complement of 1 of the total number of bytes allocated to that line (excluding S field).

Motorola S2 format

```
S224000380788812CF7C8812CFC0CFC1CFC2CFC3CFC4CFC5CFC6CFC7CFD0CFD1CFD2CFD3CF7C
S2240003A0D4CFD5CFD6CFD7CFD8CFD9CFDACFDBCFDCCFDDCFDECFDFCFE0CFE1CFE2CFE3CF90
S2240003C0E4CFE5CFE6CFE7CFE8CFE9CFEACFEBCFECCFEDCFEECFEFCFF0CFF1CFF2CFF3CE71
S2240003E0F4CEF5CEF8CEF9CFFACFFEDD8812C8C8C9C9CACACCCCCCCCDCDA8A9AAABACAD28
S224000400AEAFCFB4CFB5CFB6CFB7CFBCCFBDA0A1A2A3A4A5A6A7CFB0CFB1CFB2CFB3CFB8AC
S224000420CFB9F6F7CE94CE95CE9688CE97CE90CE91CE9288CE93CE9CCE9DCE9E88CE9FCE22
S22400044098CE99CE9A88CE9BCE80CE81CE8288CE83CE84CE85CE8688CE87CE88CE89CE8A9E
S22400046000CE8BCE8CCE8DCE8E88CE8FE438E536E634E732CEE02FCEE12CCEE229CEE326CE
```

<ADDR>
<DATA BYTES>
<CHECKSUM>
<COUNT> 32-byte
<S FIELD>

A.2.6 Symbol Information

A.2.6.1 Creating symbol information (rel88)

The rel88 is a utility used to create symbol information. It will obtain symbol information from the specified object file and then create its list. The target object files are the relocatable object file created with asm88 and the absolute object file created with link88.

Generally, this tool is used for two purposes: one for checking the symbol list after linking and second for generating a file to be input to the sym88.

The rel88 outputs a list in accordance with the standard output.
The following explains the operations to obtain the symbol list of an absolute object file.

<rel88 operation procedure>

When creating a symbol list for the absolute object file

(1) Set the directory in which the absolute object file (.a) is presented as the current drive.

(2) Start-up the rel88 with the next format.

rel88_[flag]_input file name_>_output file name↵

_ indicates a space key input.
↵ indicates a return key input.

General flags

Flag	Description
+sec	Outputs the start address and size of each section.
-v	Sorts the sections contents according to the symbol value.

Refer to the following examples for information on the flag effects. Refer to Appendix C for more details on the flag.

Since the rel88 output corresponds to the standard output, a file will be created according to the output redirect.

Example: `C:\USER>c:\EPSON\rel88 -v +sec sample.a > sample.ref`↵

Inputs the absolute object file "sample.a" created in the USER of the sub-director of drive C and then creates the symbol list file "sample.ref" in the same directory as the input file.
If the PATH to rel88 is set, then there is not need to specify the path before rel88.

The following indicate the list of symbols that are created.

Correlation with flag

```
*** rel88 (default) format ***

0x8000c         acia.o
0x80b8d         acia.o
0x8000C  n_getch
0x80bcD  _buffer
0x8059C  n_recept
0x8045C  n_outch
0x80baD  _ptlec
0x80b8D  _ptecr
0x8082C  n_main

*** rel88 -v format ***

SECTION 1
0x008000 c      acia.o
0x008000 C   n_getch
0x008045 C   n_outch
0x008059 C   n_recept
0x008082 C   n_main
```

```
SECTION 2
0x0080b8 d      acia.o
0x0080b8 D  _ptecr
0x0080ba D  _ptlec
0x0080bc D  _buffer

*** rel88 +sec format ***

SECTION 1:  code
       address = 0x008000  size = 0x000b8

SECTION 2:  data
       address = 0x0080b8  size = 0x00000
```

(For reference)

```
*** -a format ***

0x000000 c  sec: 1      acia.o
0x0000b8 d  sec: 2      acia.o
0x0000bc D  sec: 2 _buffer
0x0000b8 D  sec: 2 _ptecr
0x0000ba D  sec: 2 _ptlec
0x000000 C  sec: 1 n_getch
0x000082 C  sec: 1 n_main
0x000045 C  sec: 1 n_outch
0x000059 C  sec: 1 n_recept

*** -d format ***

0x000000 c      acia.o
0x0000b8 d      acia.o
0x000000 C  n_getch
0x0000bc D  _buffer
0x000059 C  n_recept
0x000045 C  n_outch
0x0000ba D  _ptlec
0x0000b8 D  _ptecr
0x000082 C  n_main

*** -g format ***

0x000000 C  n_getch
0x0000bc D  _buffer
0x000059 C  n_recept
0x000045 C  n_outch
0x0000ba D  _ptlec
0x0000b8 D  _ptecr
0x000082 C  n_main

*** +dec format ***

       0 c      acia.o
     184 d      acia.o
       0 C  n_getch
     188 D  _buffer
      89 C  n_recept
      69 C  n_outch
     186 D  _ptlec
     184 D  _ptecr
     130 C  n_main
```

A.2.6.2 Creating symbolic table file (sym88)

The sym88 symbolic table file generator converts symbol information reference (.ref) output from the rel88 symbol information generator into an information file that contains a symbolic table for symbolic debugging in the ICE.

<sym88 operation procedure>

(1) Set the directory in which the symbol information reference file (.ref) is presented as the current drive.

(2) Start-up the sym88 with the next format.

`sym88`_input file name↵

_ indicates a space key input.
↵ indicates a return key input.

Example: `C:\USER>c:\EPSON\sym88 sample.ref`↵

Inputs the symbol information reference file "sample.ref" created in the USER of the sub-director of drive C and then creates the symbolic table file "sample.sy" in the same directory as the input file.
If the PATH to sym88 is set, then there is not need to specify the path before sym88.

APPENDIX B CREATING PROCEDURE OF ASSEMBLY SOURCE FILE (Sub tool chain)

B.1 Outline

When you develop a program using the assembly language, first create an assembly source file using the CPU instructions and the pseudo-instructions included with the cross assembler. The assembly source file should be created according to the contents and rules to be explained hereafter, using an editor you have.

B.1.1 File Name

As explained in Section A.2.3, this assembler is separated into two programs: the structured preprocessor sap88 which expands macro instructions into the format that can be assembled by the asm88, and the cross assembler asm88 which actually executes assembly. Files to be handled in this series of procedures are an assembly source file. However, since there are some difference in each file, extensions of the file names are specified as below.

Structured assembly source file: file_name.s

This is an assembly source file which includes macro instructions, etc., and is input into the structured preprocessor sap88. When you create programs using the assembler language, create assembly source files to make the file name with the extension ".s".

Assembly source file: file_name.ms

This is an assembly source file in which the macro instructions have been expanded, and is generated from the structured preprocessor sap88.

In the structured preprocessor sap88 and the cross assembler asm88, files with other extensions can be input, but generally use the above mentioned extension.

B.1.2 Source File Differences Depending on sap88 and asm88

As explained in the previous section, format of the file to be input to the cross assembler asm88 is different from that of the structured preprocessor sap88 as to contents.

The statement (line) such as macro instruction and sap88 pseudo-instruction, which can be used in the structured preprocessor sap88, cannot be distinguished in the cross assembler asm88, and will cause an error. Consequently, when using the macro instructions, be sure to expand it to the format which can be input into the cross assembler asm88, using the structured preprocessor sap88.

In particularly, attention should be paid when modifying the source file ".ms" being input into the asm88 directly.

The pseudo-instructions which are incorporated in the cross assembler asm88 functions will not cause an error in the structured preprocessor sap88.

In the pseudo-instructions explained later, details for only the structured preprocessor sap88 are indicated by [sap88 only] or the notes are described. Take care when reading.

B.1.3 Macro Instructions

Macro instruction allows the user to define virtual instructions with instruction sequences. The structured preprocessor sap88 expands the defined instructions into the source format that can be assembled by the cross assembler asm88. The following describes the outline of it.

When using the same statement block in multiple parts of a program, previous define the statement block with an optional name, after this the statement block can be called using the defined name. The defined statement block is Macro. Describe the macro name that has been defined and necessary parameters in program, to call the macro. That part is expanded in the contents of the statement block that have been defined as a macro by the structured preprocessor sap88, and at that point the changing of the specified parameters is also to be done.

In addition to the macro-definition and the macro-call, some pseudo-instructions related to the macro have been provided. For details, see Section B.3.8.

B.2 General Format of Source File

Assembly source file is composed of statements (lines) such as the CPU instruction set, pseudo-instructions which are incorporated in the sap88 and asm88, and comments, and is completed by END pseudo-instruction (pseudo-instruction to terminate assembly). (Statements can be described after the END pseudo-instruction, however, that part will not be assembled.)
The following explains the asm88 fundamentally. (Functions permitted on the asm88 will not cause an error on the sap88.)

Example of source file

```
     subtitle  "assembly source file example (sample.s)"
     public    main
     external  src_address, dst_address, counter
;
     code
main:
     ld   ix,[src_address]
     ld   iy,[dst_address]
     ld   hl,[counter]
     ret
;***
     end
```

The following explains the general particulars such as the composition of the statement and characters and notation for numerical values which can be used.

Each source program statement should be written using the following format.

Symbol field	Mnemonic field	Operand field	Comment field

Example:

```
    on              equu            1000h
    start:          jrl             init            ;to initialize
    flag:           db              [1]
    value:          db              080h
```

In the above sort of format line, the line end normally is the termination, however, the operand may be described over several lines.

Symbol field: In this field, describe a symbol. A colon (:) must be used following the symbol except for the statement of the EQU or SET instruction.
Use symbols properly in accordance with the following definition.

Symbol •Label (Colon must follow)
•Name (Constant definition by EQU or SET instruction)

Mnemonic field: In this field, describe an operation code or a pseudo-instruction.

Operand field: In this field, describe an operand or constant of each instruction, a variable, a defined symbol, a symbol that indicates memory address, or an operational expression.

Comment field: Put semicolon (;) at the beginning of this field, and describe a comment following it.

B.2.1 Symbol

Symbol is the name in which the specific value is defined. The following two ways are to define a symbol.

(1) Label

The symbol that is put at the beginning of statement of CPU instructions or data definition is defined as a label. The value that is defined to the symbol is the address of the CPU instruction or data area.

(2) Name

It is defined using the EQU or SET pseudo-instruction. The value that is defined to the symbol is the value of <expression> that is specified using the EQU or SET pseudo-instruction.

The symbol definition is in accordance with the following rules.

- Although the symbol length is not restricted, a maximum of 15 characters from the front will be distinguished as a symbol.
- In the case of a label, it can be described from any column, however, a colon (:) must be used at the end of a label.
- In the case of a name, it must begin from column 1.
- The characters that can be used for symbols are as follows:
  Alphabetic characters (A–Z, a–z), Arabic numerals (0–9), _
- To input symbol it does not matter whether capital letters or small letters are used. In the default setting, capital letters and small letters are not distinguished, therefore symbols ABC and abc are handled identically. However, when the -c flag is used, they are distinguished.
- A symbol cannot begin with a number.
  Symbol names must begin with an alphabetic character or "_".

B.2.2 Mnemonic

A CPU instruction or a pseudo-instruction is placed in the mnemonic field. These are normally composed of character-strings that end with a blank space. These are discussed later.

In the default setting of the asm88 and sap88, capital letters and small letters are not distinguished. In such cases, even if inputting the following, they will all be considered as correct and the same.

Examples: `byte  BYTE  bYtE`

In the default setting, it is also permissible for a CPU instruction set to be written either in capital letters or small letters. When writing programs, it is better to write them with the standard method. However, when handling the symbol name to distinguish between capital letters and small letters using -c flag, be sure to describe the CPU instruction set and register name in small letters.

Example:

```
     jrl  ABC  ;jump to label ABC
     ld   a,b  ;A register <- B register
```

B.2.3 Operand

0 or more operands can be placed in accordance with the content of the mnemonic field. These operands are allocated by the parameter strings. They begin from a blank character indicating the termination of the mnemonic field, are delimited by a comma and end with a blank character or semicolon.

B.2.4 Comment

Comments are disregarded in the process of assembly. The comment begins with a ";" (semicolon) and ends at the termination of the line end (line feed code).

B.2.5 Numerical Expression

Bit control is frequently executed in a microcomputer built into the equipment. For this reason, asm88 and sap88 can handle binary, octal, hexadecimal and decimal expressions as the radix of numerical expression.

The radix is recognized by placement of the following characters after the number.

B:	Binary
O, Q:	Octal
H:	Hexadecimal
None:	Decimal (D can be used.)

(These may also be written as small letters.)

The numbers must begin with Arabic numerals (0–9). For example, the number "10" can appear as follows.

10:	Decimal
1010B:	Binary
12Q:	Octal
0AH:	Hexadecimal

(To distinguish from names all hexadecimal numbers using letters A to F must have a "0" in front. eg. 0AH = HEX number, AH = name)

B.2.6 Characters

The sap88 and asm88 have adopted the notation that has been normally called ASCII (American Standard Code for Information Interchange) for expression of characters and character strings.

B.2.7 ASCII Character Set

The ASCII character set code is composed of two parts: 7 bits data according to the characters and 1 bit parity to check whether there is an error during transfer. The ASCII character set is classified into the following four types.

In the asm88, the notation characters can be handled as a character constant by enclosing them with single quotation marks such as 'A', 'Z' and 'X'. '\'' is particularly used for the single quotation marks themselves.

To express a character which can not be displayed such as a control code, the asm88 permits the following notations for control characters thought to have a particularly high usage frequency.

'\a'	Bell	(07H)
'\n'	New-line	(0AH)
'\r'	Return	(0DH)
'\t'	Tab	(09H)
'\b'	Back space	(08H)
'\e'	Escape	(1BH)
'\i'	Shift-in	(0FH)
'\o'	Shift-out	(0EH)

Table B.2.7.1 ASCII character code table

L \ H	00	01	02	03	04	05	06	07
00	NUL	DEL	SP	0	@	P	`	p
01	SOH	DC1	!	1	A	Q	a	q
02	STX	DC2	"	2	B	R	b	r
03	ETX	DC3	#	3	C	S	c	s
04	EOT	DC4	$	4	D	T	d	t
05	ENQ	NAK	%	5	E	U	e	u
06	ACK	SYN	&	6	F	V	f	v
07	BEL	ETB	'	7	G	W	g	w
08	BS	CAN	(	8	H	X	h	x
09	HT	EM	)	9	I	Y	i	y
0a	LF	SUB	*	:	J	Z	j	z
0b	VT	ESC	+	;	K	[	k	{
0c	FF	FS	'	<	L	\(¥)	l	|
0d	CR	GS	-	=	M	]	m	}
0e	SO	RS	.	>	N	^	n	~
0f	SI	US	/	?	O	_	o	DEL
	00		01		10		11	

Section

The notation, \nnn (nnn is an octal), can also be used. When this notation is used, bell, for example, can be written '\007'.

These descriptions by escape sequences are only permitted in character strings. The character string can be handled by ASCII instruction, and they can also be expressed by sets of characters enclosed by single quotation marks.

B.2.8 Expressions

Constants are set at many points within programs, for example, the operands for CPU instruction set and the parameters for pseudo-instructions. Moreover, constants can be shown using expressions. The cross assembler asm88 evaluates expressions and can make the result value into the constant. A variable of the same size as the numbers used by the CPU or a larger one may be used for the expression evaluation during assembly.

NOTE:

(1) When a relocatable code is made, the address can only be used within the expression of which the result will be a quantity that becomes relocatable or a constant.

Consequently, the following expressions may be used.

```
label1    - label2     ;When two labels are in the same program selection
label1    + <constant>
label1    - <constant>
```

The following expressions may not be used because the result will not be a relocatable quantity or a constant.

```
label1    + label2
label1    & label2
label1    * <constant>
label1    / <constant>
label1    % <constant>
label1    * label2
label1    / label2
label1    % label2
<constant>+ label2
label1    - label2     ;When two labels are in the different program selection
```

(2) Since the results do not become relocatable quantity, logic operations using a relocatable address become errors during assembly.

Expressions are composed of several terms linked by binary operators (for example, +). In the evaluation, these expressions are calculated with 16-bit precision.
The following terms may be used within the expressions.

1 Numbers
2 Variables which have been defined by the user to use the EQU and SET instructions, and declared labels
3 Location counters $

When $ is used as the operand for the CPU instructions, the address immediately preceding the instruction is applied.

The asm88 is a two pass assembler and the values for several variables which are used in program are not defined in the pass 1 stage. When variables for which values are undefined appear within expressions during the pass 1 execution, 0 is assigned for them. And if there are variables for which values are still undefined in pass 2 execution, an error results. Also, if variables which were undefined when used for the expression in pass 1 are used in pass 2, it causes a phase error. Consequently, you should define the values for variables prior to using them in an expression.

B.2.9 Operators

The asm88 accepts the following operators.

Table B.2.9.1a Unary operator

Operator	Function
+a	Positive sign Example: `ld a,#+25h`
-a	Negative sign Example: `add b,#-13h`
~a	Assigns the values reversing each bit. Example: `and a,#~10h`
LOW a	Assigns a lower 8-bit value of an expression. Example: `or b,#low 1234h`
HIGH a	Assigns a lower 8-bit value of an expression after the expression value is shifted 8-bit to the right. This is the same as that to return the upper 8-bit of a 16-bit expression. Example: `ld h,#high 1020h`
BOC	Calculates a bank value from a physical address. This operator is effective for a physical address. (Bank Of Code) Example: `ld a,#boc label` `ld nb,a`

LOC	Calculates a logical address within the logical space from a physical address. This operator is effective for a physical address.
	(Logical address Of Code)
	Example: `ld hl,#loc label`
	`jp hl`
	`:`
	`label:`

POD	Calculates a page value from a physical address. This operator is effective for a physical address. (Page Of Data) Example: `ld a,#pod label` `ld ep,a`

LOD	Calculates a logical address within the page from a physical address. This operator is effective for a physical address.
	(Logical address Of Data)
	Example: `ld ix,#lod label`
	`ld a,[ix]`
	`:`
	`label:`

Table B.2.9.1b Binary operator

Operator	Function
a+b	Addition (32-bit signed integer) Example: `sbc [hl],#25h+10h`
a-b	Subtraction (32-bit signed integer) Example: `sub a,#63h-03h`
a*b	Multiplication (32-bit signed integer) Example: `xor l,#48h*5h`
a/b	Integer division (32-bit signed integer) Example: `cp ba,#1256h/31h`
a%b	Remainder. Divides the left operand by the right operand, and returns the remainder. Example: `add a,#0d7h%4fh`
a&b	Logical AND. Returns true if both operands are true. Returns false if either of the operands is false or both operands are false. Example: `ld sp,#04a1h&2030h`
a|b	Logical OR. Returns true if either operand is true or both operands are true. Example: `ld ix,#3026h|1000h`
a^b	Exclusive OR. Returns true if one operand is true and the other is false. Returns false if both operands are true or false. Example: `ld [iy],#44h^10h`
a<<b	Shift to left. Shifts b (integer) bits to the left. Example: `adc hl,#5000h<<3`
a>>b	Shift to right. Shifts b (integer) bits to the right. Example: `cp ba,#8130h>>10h`

Priority for operators

An expression is evaluated from left to right, however, an operator with higher priority is evaluated earlier than the other operators immediately in front of or behind it. If there are two or more continued operators equal in priority, the operators are evaluated from the left side.
Every left parenthesis "(" must have a corresponding right parenthesis ")".
The following table shows the priority for operators.

Table B.2.9.2 Priority for operators

Operators	Priority
|, ^, &	Low
+ (addition), - (subtraction)	↑
*, /, %, <<, >>	
BOC, LOC, POD, LOD	↓
HIGH, LOW, ~, -, +	High

Operation rules for BOC, LOC, POD and LOD

In the unary operators, four operators BOC, LOC, POD and LOD are peculiar to the S1C88, and possesses original rules for operation as the below.

```
BOC   (physical address & 0x7f8000) >> 15
LOC   If (physical address & 0x7f8000)
              (physical address & 0x7fff) | 0x8000
      else
              (physical address & 0x7fff) | 0x0000
POD   (physical address & 0xff0000) >> 16
LOD   (physical address & 0xffff)
```

In the above, the value indicates the physical value possessed by the operand. During assembly, the asm88 only generates special relocation information corresponding to each operator and the actual address calculation is done by the link88 during linking.

B.2.10 Instruction Set

The asm88 accepts each of the following instructions as CPU instruction set.

S1C88 Family instruction list

```
adc  cp   inc  neg  rete sep  swap
add  cpl  int  nop  rets sla  upck
and  dec  jp   or   rl   sll  xor
bit  div  jrl  pack rlc  slp
call djr  jrs  pop  rr   sra
carl ex   ld   push rrc  srl
cars halt mlt  ret  sbc  sub
```

B.2.11 Register Name

The CPU register names indicated in the following have been reserved as keywords in the asm88. Refer to the "S1C88 Core CPU Manual" for information on the respective register functions.

a	Data register A
b	Data register B
ba	A and B register pair
h	Data register H
l	Data register L
hl	Index register HL
ix	Index register IX
iy	Index register IY
sp	Stack pointer SP
br	Base register BR
sc	System condition flag SC
pc	Program counter PC
nb	New code bank register NB
cb	Code bank register CB
ep	Expand page register EP
xp	XP expand page register for IX
yp	YP expand page register for IY
ip	XP and YP register

B.2.12 Addressing Mode

The S1C88 determines the execution address according to the following 12 types of addressing modes.

Table B.2.12.1 List of S1C88 addressing modes

No.	Addressing mode
1	Immediate data addressing
2	Register direct addressing
3	Register indirect addressing
4	Register indirect addressing with displacement
5	Register indirect addressing with index register
6	8-bit absolute addressing
7	16-bit absolute addressing
8	8-bit indirect addressing
9	16-bit indirect addressing
10	Signed 8-bit PC relative addressing
11	Signed 16-bit PC relative addressing
12	Implied register addressing

Refer to the "S1C88 Core CPU Manual" for details on each addressing mode. The notation rules for the operands corresponding to these addressing modes are as follows.

Table B.2.12.2 Notation rules for operands

No.	Notation rule
1	A "#" is to be placed in front of numeric expressions and symbols
2	Register name is to be written directly
3	Index register is to be enclosed by brackets ([ ])
4	Index register and displacement are to be enclosed by brackets ([ ])
5	Index register + L is to be enclosed by brackets ([ ])
6	A "BR:" is to be placed in front of numeric expressions and enclosed by brackets ([ ])
7	Numeric expressions and symbols are to be enclosed by brackets ([ ])
8	Numeric expressions and symbols are to be enclosed by brackets ([ ])
9	Numeric expressions and symbols are to be enclosed by brackets ([ ])
10	Numeric expressions and symbols are to be written directly
11	Numeric expressions and symbols are to be written directly
12	None

B.2.13 Example for Mnemonic Notation

The examples for mnemonic notation in each addressing mode are shown in the below.

Addressing	Constant	Name name equ 50h	Label (default) label: address 00ffh	Default definition
#nn 0 to 255	eg.) ld a,#0ffh	eg.) ld a,#name	eg.) ld a,#label	-----
#mmnn 0 to 65535	eg.) ld ba,#1000h	eg.) ld ba,#name	eg.) ld ba,#label	-----
[br:ll] 0 to 255	eg.) ld b,[br:0ffh]	eg.) ld b,[br:name]	eg.) ld b,[br:label]	[br:low lod label]
[hhll] 0 to 65535	eg.) ld 1,[1000h]	eg.) ld l,[name]	eg.) ld l,[label]	[lod label]
[ix+dd] [iy+dd] [sp+dd] -128 to 127	eg.) ld [ix+10h],a	eg.) ld [ix+name],a	-----	-----
#hh 0 to 255	eg.) ld br,#0ffh	eg.) ld br,#name	eg.) ld br,#label	high lod label
#pp 0 to 255	eg.) ld ep,#05h	eg.) ld ep,#name	eg.) ld ep,#label	pod label
#bb 0 to 255	eg.) ld nb,#05h	eg.) ld nb,#name	eg.) ld nb,#label	boc label
rr -128 to 127	eg.) jrs 10h	eg.) jrs name	eg.) jrs label	loc label
[kk] 0 to 255	eg.) jp [10h]	eg.) jp [name]	eg.) jp [label]	[low lod label]
qqrr -32768 to 32767	eg.) jrl 1000h	eg.) jrl name	eg.) jrl label	loc label

- Meaning of the above mentioned default definitions are as follows:
  For example, when "jrl label" has been described, the cross assembler asm88 judges as "jrl loc label".

  ```
  jrl  label  →  jrl  loc label
  ```

  The program sequence is long jumped to the logical address converted from the physical address.

- An error occurs when the operand exceeding the above mentioned addressing range has been specified, or when it is judged to exceed it.

- In programming, pay attention to the following points when using the short branch or long branch instruction.

  `jrs(l)  10H` ..... Jumps to the address at a distance of (10+1)H from current address
  `jrs(l)  $+10H` ... Jumps to the address at a distance of 10H from current address

Except for the above, notations described in the "S1C88 Core CPU Manual" can be used as is.

B.3 Pseudo-Instructions

In this chapter the usage of each type of pseudo-instruction supported by the asm88 and sap88 is explained in the form classified by function. The format as explained below has been adopted for each explanation to permit reference to it at any time.

View of the explanation

The explanation contents of each pseudo-instruction have been configured as the following format.

1) Name

Name of the pseudo-instruction . . . Function of the instruction

2) Format

Here the instruction format is described. The format is explained using notations according to the following rules.
The explanations of the respective terms used in the operand notations are as follows.

<Expression>
General expression composed of symbols and constants including operators

<Numerical expression>
Constant expression using a numerical value expression (including name which has been defined as constant by EQU instruction)

<Label>
Symbols having a definition within the self-module that has a relocatable property

<Name>
Symbols defined by EQU and SET instructions

<Symbol>
Name to be defined for the specific value

<Character string>
Character strings enclosed by double quotation marks
The following symbols have been given special meanings.

{ } ... The enclosed part indicated an optional selection.
{ }* .. This option may be placed repeatedly any number of times.
| | |.. When different parameters of a number of different types can be adopted, one among them that is delimited by this symbol must necessarily be used as a parameter.

Other symbols
Commas ","s, brackets "[" and "]", and parentheses "(" and ")" may be input as assembler sources.

3) Functions

Here the operations of the instruction are explained in detail.

4) Examples

Here usage examples are indicated. Several types may be written depending on the instruction.

5) Related items

Here instructions that function in a similar manner and instructions that assist in understanding are indicated.

6) Restriction

Here restrictions for use are provided. Also, causes of errors that occur in the use of an instruction (forgetting the separator, for example) are explained.

B.3.1 Section Setting Pseudo-Instructions

The section setting pseudo-instructions set each section (code section and data section) and decides program area. The section setting pseudo-instructions are as follows:

CODE DATA

The section setting pseudo-instruction of the cross assembler asm88 has been defined on assumption that the code section should be allocated into ROM and data section into RAM. It aims that the non-volatile data such as program codes and constant data should not be assigned into RAM, since the microcomputer to built into an equipment has RAM area that the initial values become undefined. Therefore, when the non-volatile data such as program codes and constant data are described, it must be described within code section to set the code section by CODE pseudo-instruction. When the volatile data such as work area and stack area are described, it must be described within data section to set the data section by DATA pseudo-instruction.
Correspondence of each pseudo-instruction, setting section, area used, and contents to be described are shown in table below.

Section name	Area used	Contents to be described
Code section (CODE)	ROM	Data allocation that is necessary to decide from the power on, such as program code, constant data, and table.
Data section (DATA)	RAM	Reservation for data area that does not matter if the initial value is undefined at power on, such as work area, stack area, flags, and buffers.

Name:

CODE.....Definition of program section

Format:

CODE

Functions:

This instruction is used to allocate the program and constants in the CODE section (ROM area). An optional number of CODE sections may be defined within one module and resumed during assembly. Since this instruction specifies the section with the same function as the DATA pseudo-instruction, be sure to specify which when in the assembly. When it has not been specified, an error message is output.

Example:

Defines the program and constants in the code section.

```
        code
trans:  ld  [iy],[ix]
        inc ix
        inc iy
        djr nz,trans
        ret
        db  01h, 02h, 03h, 04h, 05h
```

Related items:

DATA, ORG

Name:

DATA.....Definition of data section

Format:

DATA

Functions:

This instruction is used to reserve and allocate the data area in the DATA section (RAM area). An optional number of DATA sections may be defined within one module and resumed during assembly. Normally, the data section definition performs only area reservation, and it is not output to the object as a result of the assembly. However, this section is a RAM area. When using equipment with built in microcomputer, pay attention that the RAM area is undefined at the power on and the initial values are invalidated.

Since this instruction specifies the section with the same function as the CODE pseudo-instruction, be sure to specify which when in the assembly for the data section. When it has not been specified, an error message is output.

Example:

Reserves an area for flag and buffer table in the data section.

```
            data
flag:       db [1]
buffer:     db [256*8]
```

Related items:

CODE, ORG

B.3.2 Data Definition Pseudo-Instructions

Data definition pseudo-instruction is the pseudo-instruction to define data to be stored into the memory. The data definition pseudo-instructions are as follows:

DB DW DL ASCII PARITY

Name:

DB.... Reserve/constant setting of the byte unit data area

Format 1:

DB <expression> {,<expression>}*

Format 2:

DB <expression> (<numeric expression>) {,<expression> (<numeric expression>)}*

Format 3:

DB [<numeric expression>] {,[<numeric expression>]}*

Functions:

This instruction is used to reserve the 1 byte unit data area and to set the constant. The setting of constants are done according to a string of numeric values delimited by a comma or the specification for the repeat number. The parameters for this instruction can be described over several lines, but you should take care that the relocation information for linking are not included. Further when this instruction is used, it should be described within the DATA (RAM) area when reserving data area, and within the CODE (ROM) area when setting constant. The code generation rules for each format are as follows.

- Format 1

  This format defines the optional constant as the optional number of object codes in 1 byte unit and multiple expressions can be specified for an operand field. The expression is handled as constant value of 1 byte and when multiple specifications are made, the object codes are generated in the order of specification.

- Format 2

  This format repeat defines the optional constant in 1 byte units and sets the repeat number in a <numeric expression> enclosed by parentheses.

- Format 3

  This format reserves the area for the number of bytes that have been assigned by the <numeric expression> enclosed by brackets. The code generated within the object at this time is 0.

Integer numeric constants, character constants and symbols can be used as the expressions for formats 1 and 2, but they must necessarily have an absolute numeric attribute. The value of the expression must also be within the range of -128 to 255. When an operation result is outside the above range, it will be made an error and the value of the lower 1 byte will be made the evaluation value. Each format can be premixed for one instruction.

Examples:

```
buffer: db [50]           ;Reserves 50 bytes area

tratbl: db '0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F'
                          ;Reserves 16 bytes data as the constant

xhrbuf: db ' '(64)        ;Reserves 64 bytes and initializes at the character code for the space
        db '*'(64)        ;Reserves 64 bytes '*' as the constant
```

Related items:

DW, DL

Name:

DW..... Reserve / constant setting of the word unit data area

Format 1:

DW <expression> {,<expression>}*

Format 2:

DW <expression> (<numeric expression>) {,<expression> (<numeric expression>)}*

Format 3:

DW [<numeric expression>] {,[<numeric expression>]}*

Functions:

This instruction is used to reserve the word (2 bytes) unit data area and to set the constant. The setting of constants are done according to a string of numeric values delimited by a comma or the specification for the repeat number. The parameters for this instruction can be described over several lines. Further when this instruction is used, it should be described within the DATA (RAM) area when reserving data area, and within the CODE (ROM) area when setting constant. The code generation rules for each format are as follows.

- Format 1

  This format defines the optional constant as the optional number of object codes in word (2 bytes) units and multiple expressions can be specified for an operand field. The expression is handled as a long word constant value or symbol value and when multiple specifications are made, the object codes are generated in the order of specification.

- Format 2

  This format repeat defines the optional constant in word units and sets the repeat number in a <numeric expression> enclosed by parentheses.

- Format 3

  This format reserves the area for the number of words that have been assigned by the <numeric expression> enclosed by brackets. The code generated within the object at this time is 0.

Integer numeric constants, character constants and symbols can be used as the expressions for formats 1 and 2. When the expression has a relocatable quality, the logical address of the location where the concerned symbol has been allocated is rearranged during linking. The value of the expression must also be within the range of -32766 to 65535. When an operation result is outside the above range, it will be made an error and the value of the lower 2 bytes will be made the evaluation value. Each format can be premixed for one instruction.

Examples:

```
array:    dw [10]  ;Reserves 10 word size area

          external func1,func2,func3,func4,func5
jmptbl:   dw func1,func2,func3,func4,func5
                   ;Jump table of the functions
```

Related items:

DB, DL

Name:

DL ..... Reserve / constant setting of the long word unit data area

Format 1:

DL <expression> {,<expression>}*

Format 2:

DL <expression> (<numeric expression>) {,<expression> (<numeric expression>)}*

Format 3:

DL [<numeric expression>] {,[<numeric expression>]}*

Functions:

This instruction is used to reserve the long word (4 bytes) unit data area and to set the constant. The setting of constants are done according to a string of numeric values delimited by a comma or the specification for the repeat number. The parameters for this instruction can be described over several lines. Further when this instruction is used, it should be described within the DATA (RAM) area when reserving data area, and within the CODE (ROM) area when setting constant. The code generation rules for each format are as follows.

- Format 1
  This format defines the optional constant as the optional number of object codes in long word (4 bytes) units and multiple expressions can be specified for an operand field. The expression is handled as a long word constant value or symbol value and when multiple specifications are made, the object codes are generated in the order of specification.
- Format 2
  This format repeat defines the optional constant in long word units and sets the repeat number in a <numeric expression> enclosed by parentheses.
- Format 3
  This format reserves the area for the number of long words that have been assigned by the <numeric expression> enclosed by brackets. The code generated within the object at this time is 0.

Integer numeric constants, character constants and symbols can be used as the expressions for formats 1 and 2. When the expression has a relocatable quality, the lower 16 bits value is rearranged as a valid value during linking. Each format can be premixed for one instruction.

Examples:

```
lubarr:  dl [10]    ;Reserves 10 4 byte size areas
lonum:   dl 13768   ;Sets the constant lonum with a long word size integer
```

Related items:

DB, DW

Name:

ASCII.....ASCII text storing in memory

Format:

ASCII character expression {, character expression}*

character expression = character string | character constant | byte constant

Functions:

This instruction is used to store the ASCII character code in memory.

For the area reserved by this instruction, the ASCII text assigned by the parameter must be stored in the memory. The character string for the parameter is decoded and stored in the memory sequentially from low-order addresses.

The area size becomes the number of bytes for the decoded parameter. The operand is a character string of one or more characters enclosed by double quotation marks.

The ASCII instruction stores the character code of each character of the character string in the memory, however, since the information showing the length and the termination of the character string is not output, the character strings may be set without a limitation.

Examples:

```
ascii "S1C88 Family"
ascii "bell",'\a'             ; bell and BELL code
ascii "bell\07"               ; Other format example
ascii "bell",'\07'            ; Other format example
ascii 62h,65h,6ch,6ch,07h     ; Other format example
```

Related item:

Table of ASCII character set

Name:

PARITY.....Setting / resetting of parity bit

Format:

PARITY <operand>

Functions:

The alphabet that has been adopted in the cross assembler asm88 is an ASCII character set. The ASCII character data are indicated with 7 bits and the most significant bit shows the parity. This bit can be optionally set or reset either always 0 or always 1 using the PARITY instruction. In addition, the total number for 1 bit can be made odd or even. The following parities can be specified for an <operand>.

PARITY 7	Sets the parity bit at 0 (default)
PARITY 8	Sets the parity bit at 1
PARITY ODD	It is set such that "1" within the 8 bits becomes odd
PARITY EVEN	It is set such that "1" within the 8 bits becomes even

Related item:

Table of ASCII character set

B.3.3 Symbol Definition Pseudo-Instructions

Symbol definition pseudo-instruction is the pseudo-instruction to define an expression with a name. The symbol definition pseudo-instructions are as follows:

EQU SET

Name:

EQU.....Name value setting

Format:

```
<name>   EQU   <expression>
```

Functions:

This instruction is used to define the <expression> with a <name>. The value of a name that has been defined by this instruction may not be changed later. Nor may an EXTERNAL declared symbol be placed on the right side of the equals sign.
Length of the expression is not restricted, but up to a 6 character hexadecimal number can be output to the assembly list. When a 7 or more character hexadecimal number has been defined, a warning is output.

In the sap88, the name defined by the EQU can be used in the conditional expression of the IFC statement that hereafter occurs, or it can be used as the parameter for the IFDEF / IFNDEF statements. [sap88 only]

Examples:

```
false   equ  0              ;Initialization
true    equ  -1
tablen  equu TABFIN-TABSTA  ;Calculation of table length
nul     equ  00h            ;Defines a character string indicating ASCII characters
soh     equ  01h
stx     equ  02h
etx     equ  03h
eot     equ  04h
enq     equ  05h
```

Related items:

SET, IFC, IFDEF, IFNDEF, REPT

Limitation:

The <name> description must begin from the 1st column.

Name:

SET.....Name value setting

Format:

<name> SET <expression>

Functions:

This instruction is the same as the EQU instruction, it is intended, among others, to improve maintenance of the assembler source code and it serves to link <numeric expressions> with the <names>. Unlike in the case of the EQU instruction, a name defined by the SET instruction can be redefined any number of times for other values and can be treated as an assembler variable. Among the attributes of the cross-reference list, which is one of the output lists of the assembler, those are defined as variables take this symbol. The right side of the equals sign must be defined before this instruction. The main object of this instruction is to use the name as a conditional assemble or macro variable and it serves as a valuable function in the structured preprocessor sap88. However, it does not have too much application in the cross assembler asm88 itself, other than functioning to permit the redefining of names.

Length of the expression is not restricted, but up to a 6 character hexadecimal number can be output to the assembly list. When a 7 or more character hexadecimal number has been defined, a warning is output.

In the sap88, the name defined by the SET can be used in the conditional expression of the IFC statement that hereafter occurs, or it can be used as the parameter for the IFDEF / IFNDEF statements. [sap88 only]

Examples:

```
abc     set  1
        ld   a,#abc
abc     set  2
        ld   a,#abc
```

Related items:

EQU, IFC, IFDEF, IFNDEF, REPT

Limitation:

The <name> description must begin from the 1st column.

B.3.4 Location Counter Control Pseudo-Instruction

The location counter control pseudo-instruction is as follows:

ORG

Name:

ORG.....Changing of location counter value

Format:

ORG <expression>

Functions:

This instruction is used to specify addresses where program has been placed. <expression> must be a relative value from a label within the current program section. At this time, an attempt to insert an absolute address into the program counter results as an error.

Length of the expression can be defined up to a 6 digit hexadecimal number, and an error occurs if 7 digits or more has been defined.

Examples:

```
sizstk  equ  200h           ;The stack size is 512 bytes
topstk:                     ;Reserves space for the stack
        org  topstk+sizstk
```

Related items:

CODE, DATA

B.3.5 External Definition and External Reference Pseudo-Instructions

External definition and external reference pseudo-instructions are the pseudo-instructions to define and refer symbols which are commonly used between modules.

- External reference pseudo-instruction .....**EXTERNAL**
- External definition pseudo-instruction .....**PUBLIC**

Name:

EXTERNAL..Symbol external definition declaration

Format:

EXTERNAL <symbol> {,<symbol>}*

Functions:

EXTERNAL and PUBLIC instructions are used so that the same symbol will be used between multiple modules. Declaration must be done with an EXTERNAL instruction to reference symbols not defined within the self-module, but rather defined within other modules. If a declaration is made in EXTERNAL, it will simultaneously be made in PUBLIC as well.

Example:

```
external     sqrt
carl         sqrt
```

Related item:

PUBLIC

Name:

PUBLIC.....Global declaration of symbol

Format:

PUBLIC <symbol> {,<symbol>}*

Functions:

When optional symbols are used in multiple modules, they are declared with the PUBLIC and EXTERNAL instructions. PUBLIC is used for declaration of symbols, such that there is a definition within the self-module that permits reference from other modules.

Example:

```
        public  sqrt    ;SQRT permits reference from other modules
sqrt:                   ;Routine that computes the square root of an integer
        .....
        etc.
```

Related item:

EXTERNAL

B.3.6 Source File Insertion Pseudo-Instruction [sap88 only]

Source file insertion pseudo-instruction is a pseudo-instruction to read and insert other files into the optional location of source file.

INCLUDE

* This instruction can only be used in the structured preprocessor sap88. The sap88 expands this instruction and creates the source file in which the specified file is inserted. In the cross assembler asm88, this instruction cannot be used and will cause an error if used.

Name:

INCLUDE.....Another file insertion

Format:

INCLUDE <file name>

Functions:

This instruction reads the specified file in the following an INCLUDE statement.
Including can be nested to optional depths. Another file can be further included into a file that is already included.

The sap88 analyses this pseudo-instruction and creates the output file in which the specified file is inserted. This pseudo-instruction is not transferred to the asm88 as is.

Examples:

```
include chargen.s  ;Character generator
include utilsub    ;General purpose subroutine group
```

Limitation:

This instruction can only be used in the structured preprocessor sap88. In the cross assembler asm88, it cannot be used and will cause an error if used.

B.3.7 Assembly Termination Pseudo-Instruction

Assembly termination pseudo-instruction terminates each source program.

END

Name:

END.....Assembly stop

Format:

END {<Label>}

Functions:

This instruction is used to stop the assembly. A list for the portion following this instruction is output, but not assembled.

B.3.8 Macro-Related Pseudo-Instructions [sap88 only]

The following pseudo-instructions are related to the macro functions, and they perform a macro definition, a macro deletion, a repeat definition, and the like.

MACRO ~ ENDM
DEFINE
LOCAL
PURGE
UNDEF
IRP ~ ENDR
IRPC ~ ENDR
REPT ~ ENDR

* These pseudo-instructions can only be used in the structured preprocessor sap88. The sap88 outputs the source file in which the setting contents of these pseudo-instructions are expanded into a form that can be assembled by the cross assembler asm88. Further these macro-related pseudo-instructions cannot be accepted in the asm88 and will cause an error if used.

Name:

MACRO.....Macro definition

Format:

```
<macro name>    MACRO   [<parameter> [, <parameter>] * ]
                <statement string>
                [EXITM]
                <statement string>
[<macro name>]  ENDM
```

Functions:

This instruction performs a macro definition. If the specified macro name is already used, the previous definition will be overridden and this current definition will redefine the macro. Names including any characters except blank characters, brackets "(" , ")", "{" , "}", "[" , "]" and a colon ":" can be used as macro names. It is not necessary to define the macro name for the ENDM line except the case that the macro definition was nested. Moreover, there is no limitation as to the number of parameters.
Arguments delimited by a comma "," can be specified by the number of your choice at the time of a macro call. The number of arguments should not necessarily be equal to the number of parameters at the time of a macro definition. If a character string identical to one parameter exists in the macro body, it will be replaced with the corresponding argument character string at the time of a macro call. If any corresponding argument does not exit it will be replaced with a blank character string. It is also possible to specify a blank character string on arguments. In this case, specification should be done using the characters which are not included in the blank character string. For example, if it is specified as shown below at the time of a certain macro "xmac" call :

```
xmac   1,,2
```

The second argument will become a blank character string. At the same time, the number of argument at the time of the call will be replaced with the sap88 system parameters NARG and narg. The blank character string arguments at this time will also be counted.
All the parameters are not necessarily independent as tokens. Some will be replaced with arguments even when they occur inside character strings. In order to reduce substitution, it is advisable to use special symbols so that too much substitution can be evaded. All symbols except a comma "," and brackets "(" , ")", "{" , "}", "[" , "]" can be used for parameters and arguments.

For example :

```
sum     macro   c,d
        ld      a,[c]
        add     a,d
        sld     [c],a
        endm

        sum     total,#20
```

The above will be interpreted as follows :

```
        l#20    a,[total]
        a#20#20 a,#20
        l#20    [total],a
```

If you redefine your macro definition as shown below, your input will be correctly replaced :

```
sum     macro   c,&d
        ld      a,[c]
        add     a,&d
        ld      [c],a
        endm
```

The blank characters before and after parameters and arguments will be discarded. The blank characters inside parameters and arguments, however, are valid. Please take caution in this respect. A macro call from inside the body of the macro for a macro definition can also be done. In this case, a macro call should be initiated at the time the macro call generates.
For example :

```
 maca   macro   x,y
        add     x,y
        endm

 macb   macro   x,y
        maca    x,y
        endm

        macb    a,#2    →    add  a,#2

 maca   macro   x,y
        sub     x,y
        endm

        macb    a,#2    →    sub  a,#2
```

A macro call from the body of the macro can be executed according to the depth of your choice. However, if the call enters a loop, the macro call will be suspended. Take a simple example for instance :

```
add     macro   x,y
        ld      a,x
        add     a,y
        ld      x,a
        endm
```

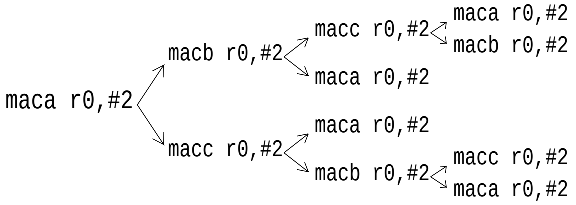
When the macro defined as above is called, it is expanded as follows :

```
                             ld   a,b
        add     b,#2    →    add  a,#2
                             ld   b,a
```

"add a,y" in the third line will call itself. The macro call, therefore, will not occur. It will turn out to be a simple "add" instruction. If we take a look at a little more complicated example :

```
maca    macro   x,y
        macb    x,y
        macc    x,y
        endm
```

```
macb    macro   x,y
        macc    x,y
        maca    x,y
        endm
macc    macro   x,y
        maca    x,y
        macb    x,y
        endm
```



When performing a conditional assembly using the IFC statement inside the body of the macro, the judgment will be made at the time of the macro call. If an EXITM line occurs at this time, the macro expansion will be suspended and the macro call will end at that moment.
For example :

```
xmac    macro   x,y
        ...
        ifc     MODE == 2
                exitm
        endif
        ...
        endm
MODE    set     2
        xmac    #3,#4
```

When called as shown above, the macro expansion will end at the EXITM line.

```
MODE    set     1
        xmac    #3,#4
```

When called as shown above, the macro expansion will be executed to the last.

It is possible to include a macro definition in the body of the macro. In this case, however, the macro name of the MACRO line corresponding to the ENDM line will be required :

```
x       macro
        ...
y       macro
        ...
z       macro
        ...
z       endm
        ...
y       endm
        ...
        endm
```

With the case shown above, the macro "y" definition will be executed at the time the macro "x" is called. In this case, however, it is not necessary to specify a macro name for the outermost macro definition ("x" in the above example) of the ENDM line. Nesting can be done to the depth of your choice.

Related items:

EQU, IFC, IFDEF, IFNDEF, IRP, IRPC, PURGE, SET

Limitation:

This pseudo-instruction can only be used in the structured preprocessor sap88. It cannot be accepted in the asm88 and will cause an error if used.

Name:

DEFINE.....Character-string macro definition

Format:

DEFINE <character-string macro name> [<substitute character-string>]

Functions:

This instruction performs a character-string macro definition. The token identical to the character-string macro name in the source after the DEFINE statement will be replaced with a macro instruction in the specified substitute character-string prior to the evaluation of all the statements except the IFDEF and IFNDEF statements. In the case that a substitute character-string is not specified, it will be replaced with a blank character-string. In addition, a character-string macro name will be subject to be evaluated in the IFDEF or IFNDEF statements.

Example:

```
define  XMAX    #128

        cp      a,XMAX
        ↓
        cp      a,#128
```

Related items:

IDEF, IFNDEF, UNDEF

Limitation:

This pseudo-instruction can only be used in the structured preprocessor sap88. It cannot be accepted in the asm88 and will cause an error if used.

Name:

LOCAL.....Definition of local label

Format:

LOCAL [<local label name> [,<local label name>] *]

Functions:

This instruction declares a local label. When a token with the name identical to that of a local label occurs inside a macro definition, it will be replaced in macros by a different label name, which will be automatically generated at each macro expansion. According to the rule of local label generation, the numerals in four digits starting with 0001 should follow the front character string "L". The front character string can be changed if specified at the start-up of the sap88.

Example:

```
macl    macro
        local   x
        cp      a,#3
        jr      c,x
        ld      d,r0
x:
        endm

        macl
        macl
        ↓
        cp      a,#3
        jr      c,L001
        ld      d,a
L001:
        cp      a,#3
        jr      c,L002
        ld      d,a
L002:
```

Related item:

MACRO

Limitation:

This pseudo-instruction can only be used in the structured preprocessor sap88. It cannot be accepted in the asm88 and will cause an error if used.

Name:

PURGE.....Macro deletion

Format:

PURGE [<macro name>]

Functions:

Once this instruction is executed, the macro definition of specified name that occur thereafter will be deleted. When name is not specified, all the macro definitions will be deleted. It is also possible to specify undefined macro name.

Example:

```
purge  add     ;delete the macro add
add    ba,#10  ;use the add instruction
```

Related item:

MACRO

Limitation:

This pseudo-instruction can only be used in the structured preprocessor sap88. It cannot be accepted in the asm88 and will cause an error if used.

Name:

UNDEF.....Deletion of a character string macro

Format:

UNDEF <character string macro name>

Functions:

The character-string macro definition will be deleted of the specified name that occur after this instruction is executed. It is also possible to specify undefined character-string macro name.

Example:

```
undef  XMAX ;delete the character string macro XMAX
```

Related items:

DEFINE, IFDEF, IFNDEF

Limitation:

This pseudo-instruction can only be used in the structured preprocessor sap88. It cannot be accepted in the asm88 and will cause an error if used.

Name:

IRP.....Repetition using character strings

Format:

```
IRP     <parameter>, <argument> [, <argument>] *
        <statement string>
ENDR
```

Functions:

With this instruction, arguments will be assigned to parameters in sequence from the left and expansion will be repeatedly performed up to the ENDR line by the times equal to the number of the arguments. If, at this time, a character string identical to the parameter exists between the IRP line and the ENDR line, such a character string will be replaced with the character string keyed by the argument.

All the parameters are not necessarily independent as tokens. Even when they occur inside character strings, they will be replaced with arguments. In order to reduce substitution, it is advisable to use special symbols for parameters so that too much substitution can be evaded. All except a comma "," and brackets " (" , ") ", " {" , "} ", " [" , "] " can be used as special symbols.
For example :

```
irp     w,10,20,30
        dw      w
endr
```

The above will be interpreted as :

```
        d10     10
        d20     20
        d30     30
```

If you modify the symbols as follows, your input will be correctly replaced:

```
irp     &w,10,20,30
        dw      &w
endr
```

The blank characters before or after parameters or arguments can be discarded. However, the blank characters located inside parameters and arguments are valid. Please take caution in this regard.

Each statement of IRP, IRPC and REPT can be nested to the depth of your choice. The ENDR line at this time will correspond to the inside IRP / IRPC / REPT lines.

Example:

```
irp     char,30,31,32,33,34,35,36,37,38,39
c_char: dw      charh
endr
        ↓
c_30:   dw      30h
c_31:   dw      31h
c_32:   dw      32h
c_33:   dw      33h
c_34:   dw      34h
c_35:   dw      35h
c_36:   dw      36h
c_37:   dw      37h
c_38:   dw      38h
c_39:   dw      39h
```

Related items:

IRPC, MACRO, REPT

Limitation:

This pseudo-instruction can only be used in the structured preprocessor sap88. It cannot be accepted in the asm88 and will cause an error if used.

Name:

IRPC.....Repetition by characters

Format:

```
IRPC  <parameter>, <argument character string>
      <statement string>
ENDR
```

Functions:

With this instruction, the characters of argument character strings will be assigned to parameters one by one in sequence from the left. The expansion will be repeatedly performed till the ENDR line by the times equal to the number of characters of arguments. If, at this time, the character strings identical to the parameters exist between the IRPC line and the ENDR line, such strings will be replaced with the characters keyed by the arguments.

All the parameters are not necessarily independent as tokens. Even when they occur inside character strings, they will be replaced with arguments. In order to reduce substitution, it is advisable to use special symbols so that excessive substitution can be prevented. All symbols except a comma "," and brackets "(" , ")", "{" , "}", "[" , "]" can be used as special symbols for parameters and arguments. For example :

```
irpc    w,abc
        dw      'w'
endr
```

The above will be interpreted as :

```
        da      'a'
        db      'b'
        dc      'c'
```

If you modify the symbols as follows, your input will be correctly replaced :

```
irpc    &w,abc
        dw      '&w'
endr
```

The blank characters before or after the parameters or arguments will be discarded. However, the blank characters inside the parameters and arguments are valid. Please take caution in this respect. Each statement of IRP, IRPC and REPT can be nested to the depth of your choice. The ENDR line at this time will correspond to the inside IRP / IRPC / REPT lines.

Example:

```
irp     char,Hello, world!
        dw      'char'
endr
        ↓
        dw      'H'
        dw      'e'
        dw      'l'
        dw      'l'
        dw      'o'
        dw      ','
        dw      ' '
        dw      'w'
        dw      'o'
        dw      'r'
        dw      'l'
        dw      'd'
        dw      '!'
```

Related items:

IRPC, MACRO, REPT

Limitation:

This pseudo-instruction can only be used in the structured preprocessor sap88. It cannot be accepted in the asm88 and will cause an error if used.

Name:

REPT .... Repetition by the specified number of times

Format:

```
REPT    <operation expression>
        <statement string>
ENDR
```

Functions:

The portion between the REPT line and the ENDR line will be repeatedly expanded by the number of times equal to the value of the operation expression. If there is any undefined name in the operation expression, the value of such a name will be evaluated as "0".

Each statement of IRP, IRPC and REPT can be nested to the depth of your choice. The ENDR line at this time will correspond to the inside IRP / IRPC / REPT lines.

Example:

```
rept    4           ;4-bit shift
        sll     a
endr
```

Related items:

EQU, IRP, IRPC, SET

Limitation:

This pseudo-instruction can only be used in the structured preprocessor sap88. It cannot be accepted in the asm88 and will cause an error if used.

B.3.9 Conditional Assembly Pseudo-Instructions [sap88 only]

The conditional assembly pseudo-instructions decide whether or not to perform the assembly within the specified range by the evaluation result of the conditional expression or whether the name has been defined or not. The conditional assembly pseudo-instructions are as follows:

IFC ~ ENDIF
IFDEF ~ ENDIF
IFNDEF ~ ENDIF

* These pseudo-instructions can only be used in the structured preprocessor sap88. The sap88 outputs the source file in which the statements subject for assembly are included. Further these conditional assembly pseudo-instructions cannot be accepted in the asm88 and will cause an error if used.

Name:

IFC..... Conditional assembly by conditional expression

Format:

```
IFC     <conditional expression>
        <statement string> [
ELSEC
        <statement string> ]
ENDIF
```

Functions:

This instruction evaluates a conditional expression. If an expression is evaluated as "true", the statements following the IFC line will become a subject to be assembled until either an ELSEC line or an ENDIF line appears. If it is evaluated as "false", the statements following the IFC line will not be considered a subject to be assembled. In the case that there is an ELSEC line, the portion between the ELSEC and ENDIF lines will become a subject to be assembled if the conditional expression of the IFC line is "false". If it is "true", the ELSEC line through the ENDIF line will not become a subject for assembly.

Each statement of IFC, IFDEF and IFNDEF can be nested to the depth of your choice. The ELSEC line and the ENDIF line at this time will correspond to the inside IFC / IFDEF / IFNDEF lines.

As explained in the following, the conditional expression comes in three cases :

1) <operation expression>
When only an operation expression is used, a decision will be made as to whether the value of the expression is "0" or not "0". If it is "0", the value will be considered as "false". If it is not "0", the value will be considered as "true". In the case that there is any undefined name in the operation expression, the value of such a name will be evaluated as "0". For instance :

```
IFC  ee
```

will be decided as equivalent to

```
IFC  ee != 0
```

2) <operation expression> <relational operator> <operation expression>
The values of each operation expression are compared. If, at this time, there is any undefined name in the operation expressions, the value of the undefined name will be evaluated as "0".
The following relational operators are available :

==	"true" if the value of the left side is equal to that of the right side
!=	"true" if the value of the left side is not equal to that of the right side
<	"true" if the left side is smaller than the right side
>	"true" if the left side is larger than the right side
<=	"true" if the left side is smaller than, or equal to the right side
>=	"true" if the left side is larger than, or equal to around the right side

3) [<conditional expression>] <logical operator> <conditional expression>

A complex conditional expression can be expressed using a logical operator. The logical operation expressions include the following :

Unary operator:

!	"true" if the conditional expression is "false"

Binary operator:

&&	"true" if the left side is "true" and the right side is also "true"
||	"true" if the left side is "true" or the right side is "true"

The operators will be classified as follows from high to low precedence : either an operation expression or a conditional expression enclosed by a round bracket > a unary operator > an operator of an ordinary operation expression > a relational operator > &&> ||

The same operator precedence will take effect inside a round bracket. A unary operator is defined as a unary operator of an ordinary operation expression and "!" of a logical operator.

In addition, "character string" can be used as an operation expression.

When such character strings occurs on both sides of a relational operator, a character string will be compared to another character string. Otherwise, the value of the length of character strings will be compared.

Example:

```
table   macro&1,&2
        ifc   narg == 1
                   ifc   ! USE_DEFAULT || DEFAULT_SIZE<64
                         &1:   db   0(64)
                   elsec
                         &1:   db   0(DEFAULT_SIZE)
                   endif
              elsec
                   &1: db     0(&2)
              endif
endm
```

Related items:

EQU, IFDEF, IFNDEF, SET

Limitation:

This pseudo-instruction can only be used in the structured preprocessor sap88. It cannot be accepted in the asm88 and will cause an error if used.

Name:

IFDEF .... Conditional assembly by the name either defined or undefined

Format:

```
IFDEF      <name>
           <statement string> [
ELSEC
           <statement string> ]
ENDIF
```

Functions:

If the name is defined by either the EQU statement or the SET statement, or is a character-string macro name which is defined by the DEFINE statement, the statements following the IFDEF line will become a subject to be assembled until either the ELSEC line or ENDIF line occurs. If the name is undefined, the statements following the IFDEF line will not become a subject to be assembled. In the case that there is an ELSEC line, the portion between the ELSEC line and the ENDIF line corresponding to the IFDEF line will become a subject to be assembled if the name of the IFDEF line is not defined. If the name is defined, the ELSEC line through the ENDIF line will not become a subject to be assembled.

Each statement of IFC, IFDEF and IFNDEF can be nested to the depth of your choice. The ELSEC line and the ENDIF line at this time corresponds to the inside IFC / IFDEF / IFNDEF lines.

Example:

```
ifdef   EXTRA_MEMORY
stack_start     equu    4000h
stack_size      equ     1000h
elsec
stack_start     equ     3800h
stack_size      equ     800h
endif
```

Related items:

DEFINE, EQU, IF, IFNDEF, SET

Limitation:

This pseudo-instruction can only be used in the structured preprocessor sap88. It cannot be accepted in the asm88 and will cause an error if used.

Name:

IFNDEF ..... Conditional assembly by the name either undefined or defined

Format:

```
IFNDEF    <name>
          <statement string> [
ELSEC
          <statement string> ]
ENDIF
```

Functions:

If the name is not defined neither by the EQU statement or SET statement, nor defined by the DEFINE statement as a character-string macro name, the statements following the IFNDEF line will become a subject to be assembled until either the ELSEC line or the ENDIF line occurs. If the name is defined, the statements following the IFNDEF line will not be processed as a subject to be assembled. In addition, in the case that there is an ELSEC line, the portion between the ELSEC line and the ENDIF line corresponding to the IFNDEF line will become a subject to be assembled if the name of the IFNDEF line is defined. If not defined, the portion will not become a subject to be assembled.

Each statement of IFC, IFDEF and IFNDEF can be nested to the depth of your choice. The ELSEC line and the ENDIF line at that time will correspond to the inside IFC / IFDEF / IFNDEF lines.

Example:

```
ifndef   SMALL_MEMORY
stack_start     equu    3800h
stack_size      equ     800h
elsec
stack_start     equ     4000h
stack_size      equ     1000h
endif
```

Related items:

DEFINE, EQU, IF, IFNDEF, SET

Limitation:

This pseudo-instruction can only be used in the structured preprocessor sap88. It cannot be accepted in the asm88 and will cause an error if used.

B.3.10 Output List Control Pseudo-Instructions

The output list control pseudo-instructions are used for that can be easily referred, and are as following 7 types:

LINENO
SUBTITLE
SKIP
NOSKIP
LIST
NOLIST
EJECT

Name:

LINENO ... Change of line number for assembly list file

Format:

LINENO <numeric expression>

Functions:

This instruction forcibly changes the line number for the assembly list file to the following line number set by the <numeric expression>. The line number can be changed up to 65535, and starts from 0 if it exceeds the upper limit.

Example:

```
lineno  99    ;line number begins from 100
```

Name:

SUBTITLE .... Subtitle setting to assembly list file

Format:

SUBTITLE <character string>

Functions:

The SUBTITLE instruction is used for outputting optional character string as subtitles onto the 4th line of the list output. After the first page, SUBTITLE appearing within the current page is used as the subtitle of the following page and continue to be used until a new SUBTITLE appears.

The character string should be enclosed by double quotation marks.

Example:

```
subtitle     "asm88 Special function library"
```

Name:

SKIP .... Suppresses all initialization codes output that exceed 4 bytes to assembly list file

Format:

SKIP

Functions:

When this instruction appears, even when there is an initialization that exceeds a one line assembly list file, that is, a size greater than 5 bytes in each of the following instructions ASCII, DB, DL and DW, it will output a 1 line code only to the assembly list file and will suppress code outputs that do not fit on the assembly list file. The NOSKIP instruction serves to counter this function, however, SKIP is set in the default.

Example:

```
noskip
     db   1,2,3,4,5,6,7,8,9,0
          ;All the hexadecimal codes output to the assembly list file
skip
     ascii"1234567890"
          ;ASCII codes output to list file as one line only
```

Related item:

NOSKIP

Name:

NOSKIP .... Outputs all initialization codes to assembly list file

Format:

NOSKIP

Functions:

This instruction is used to reverse the function of the SKIP instruction (default) that suppresses output of codes exceeding 4 bytes to the assembly list file. When this instruction appears, thereafter, if initialization codes are set for each of the ASCII, DB, DL and DW instructions, all of these codes will be output onto the list.

Example:

```
noskip
     db   1,2,3,4,5,6,7,8,9,0
          ;All the hexadecimal codes output to the assembly list file
skip
     ascii"1234567890"
          ;ASCII codes output to list file as one line only
```

Related item:

SKIP

Name:

LIST.....Assembly list file output

Format:

LIST

Functions:

When this instruction appears, thereafter, the assembly list file will be output. In the default, LIST is set.

Related item:

NOLIST

Name:

NOLIST ... Prohibition of assembly list file output

Format:

NOLIST

Functions:

When this instruction appears, thereafter, the assembly list file output will be prohibited. In order to resume the assembly list file output, use the LIST instruction. Further the line number is updated if the assembly list file output has been prohibited by NOLIST.

Related item:

LIST

Name:

EJECT.....Form feed of assembly list file

Format:

EJECT

Functions:

When this instruction appears, the form feed with the page header is inserted to the assembly list file same as an auto form feed. This instruction itself is shown in the first line of the page after form feeding.

APPENDIX C ASSEMBLY TOOL REFERENCE (Sub tool chain)

The explanation for each software tool has been arranged by the items shown below.

PROGRAM NAME

Shows the program name.

SUMMARY

Functions of the software tool are explained.

INPUT/OUTPUT FILES

Shows the execution flow and input/output files.

START-UP FORMAT

Shows the start-up command format of the software tool. This format includes the main component elements of the command line; the name of tool itself and all the flags that can be received in the tool. The command cannot be started up if you input invalid flags and/or arguments and forget the necessary arguments.

Flags are listed in [ ] by a delimiter "-" and the names. In principle, the flags are listed in alphabetical order. Flags that are composed of values alone, are listed behind all other flags. In the case of flags that accompany some values, the type of concerned value as well is shown by one of the below codes (assigned immediately following the flag name).

Code	Types of value
*	Character string
#	Integer (word size)
##	Integer (long word size)
?	Single character

The hash mark # shows word size (2-byte) integers. Double hash marks ## show long word size (4-byte) integers. When integers begin with 0x or 0X they may be interpreted as hexadecimal numbers. When they begin with O as octal numbers and in other cases as decimal numbers, they can optionally be preceded by either plus + or minus - signs.
A caret "^" immediately follows the value code, of formats of the type where there are two or more assignments per flag such that the values are stacked.
For example, the asm88 utility format is as follows:

```
asm88 -[all c l o* q RAM# ROM#
         sig# suf* x] [drive:]<files>↵
```

We know that the asm88 receives the following 10 different sorts of flags. That means, a word size integer value is assigned to the flags -RAM, -ROM and -sig. The flags -all, -c, -l, -q and -x do not have values. Character strings are assigned to -o and -suf.
Be careful of flags which normally have a hyphen placed immediately in front, appearing without one. (Provided there is no particular specification and a hyphen is assumed.)
When specifying the flag individually, RAM# in the list shown above should be assigned as -RAM#.
Furthermore, flags without values can continuously be specified by placing a "-" (hyphen) only for the head of the flags to be specified, for example, -clq.
The location and meaning of a non-flag argument is indicated by a word within < and > (<files> in the above example). Each meta-concept shows 0 or 1 or more arguments on the command line. When inputting command lines, type all the command line where meta-concepts appear in their position on the concerned line. In the case of the asm88, input one or more file names in the position shown by <files>. Meta-concepts in brackets are optional specifications. It is all right if they appear, and they may appear more than once.

FLAGS

Functions of all flags are listed. In some cases, supplementary explanations follow them depending on the situation.

ERROR MESSAGES

A list of error messages displayed during execution.

RETURN VALUE

When execution has been completed, each tool returns either of two values, "success" or "failure". This item describes the conditions under which either of the two are returned by the tool. Generally, the return value of "success" indicates that the tool executed all the necessary file processing.
This return value is used to evaluate an execution result of the tool when executing batch processing.

EXAMPLE

Here is an example using the software tool.

NOTE

Here notes for use are described.

C.1 Structured Preprocessor <sap88>

PROGRAM NAME

sap88.exe

SUMMARY

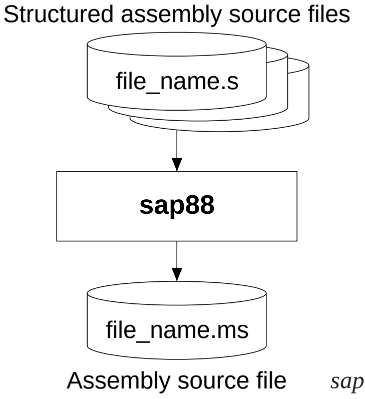
The structured preprocessor sap88 adds the macro functions to the cross assembler asm88.

The sap88 expands the macro and structured control statements included in the specified S1C88 assembly source file into a format that can be assembled by the asm88, and outputs it. At this time, the sap88 also executes the processing for including of the modularized S1C88 assembly source files and conditional assembly.

When file name has not been specified, the sap88 reads from the standard input (console).

INPUT/OUTPUT FILE

• Execution flow



sap88 execution flow

• Input file

Structured assembly source file: file_name.s

This is a structured assembly source file which is created by an editor such as EDLIN.

• Output file

Assembly source file: file_name.ms

This is the output file in which the macros in the structured assembly source file are expanded into the S1C88 instructions that can be assembled by the asm88. This file becomes an input file of the asm88. The output file extension should be made as ".ms".

START-UP FORMAT

```
sap88 -[d*^ l* o* q] [drive:] <file>↵
```

flags:

Character strings enclosed with [ ] mean flags. Explanations for each flag are discussed later.

drive:

In case the input file is not in current drive, input the drive name in front of the input file name. It can be omitted if the input file is in current drive.

file:

Specify the file name to be input to the sap88. This file name can be input using either capital letters or small letters. When <file> has not been specified, the sap88 reads from the standard input.

Note: The extension for the structured assembly source file should be made as ".s".

FLAGS

The sap88 can accept the following flags. The flags should be input with small letters.

Function	Flag	Explanation
Character-string macro definition	`-d*^`	A character-string macro is defined prior to reading in an input file. "*" has the following format: <character-string macro name> = <substitution character string> If the substitution character string is not defined and only the <character-string macro name> is defined, only the character-string macro will be defined and the substitution character string will become a blank character string. The character-string macros using the -d flag can be defined up to a maximum of 20.
Front character string specification	`-l*`	The front character string of a label name that is created at the time of the expansion of the structured control statement is designated. It is "L" in default.
Creating output file	`-o*`	An output file name is turned to *. The default status is standard output.
Suppression of start-up message	`-q`	Does not output any message related to processing of the structured preprocessor.

ERROR MESSAGES

Error message	Description
unexpected EOF in ~	The file is terminated in the middle of ~.
can't include ~	~ cannot be included.
illegal ~	~ is incorrect.
illegal define	"define" statement is incorrect.
illegal expression at ~	~ in the expression is incorrect.
illegal undef	"undef" statement is incorrect.

RETURN VALUE

The sap88 returns "success" if there is no syntax error in the input file. If there is a syntax error, "failure" is returned even if the contents of the input file are correct.

EXAMPLE

Expands the structured assembly source file "sample.s" to the assembly source file "sample.ms".

```
   C>sap88 -o sample.ms sample.s↵
```

NOTE

If there is no syntax error in a macro statement, the sap88 expands it normally even though it contains illegal operands such as wrong register names. This error will be detected by the assembler asm88.

C.2 Cross Assembler <asm88>

PROGRAM NAME

asm88.exe

SUMMARY

The cross assembler asm88 converts an assembly source file to machine language by assembling the assembly source file in which the macros are expanded by the structured preprocessor sap88. The asm88 is a high speed assembler whose functions have been simplified to increase speed, and all the added functions, such as macro and conditional assembly, are supplemented with another utility (sap88).
The asm88 deals with the relocatable assembly for modular development.
In the relocatable assembly, the relocatable object file to link up with the other modules using the linker link88 is created.
In addition, the asm88 can directly input an assembly source file and in such case, the source program can be described in free format as the following format.

Label: Mnemonic Operand ;Comment

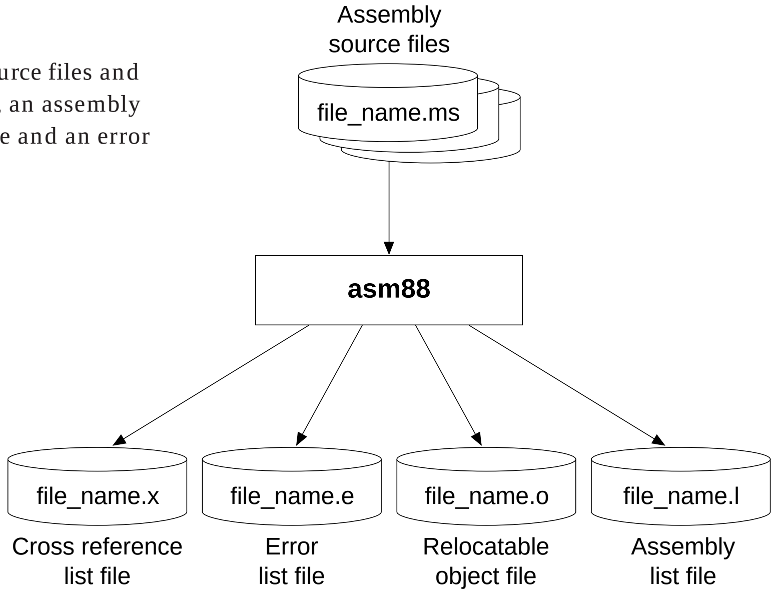
In the above format, ":" indicates the end of the label and ";" indicates the beginning of the comment. It is possible to format freely by using these separators.

The asm88 also outputs three types of lists for the programmer, an assembly list, an error list and a cross-reference list. The assembly list is composed of a line number, address and a machine code corresponding to each source statement. The line number is output as a decimal number and the address and machine code as a hexadecimal number. When errors occur during assembly, an error list file is created that is composed of a file name, the line number that generated the error, the error level and an English error message.
Also in the assembly list file, a mark "*" is placed at the line number in which an error has been generated.
It has also been designed such that the relationship between the definitions and the references of the symbols within the files can be easily understood by a cross-reference list. Since these are created as individual files, file management has also been simplified. Processing can continue even when an error occurs, provided it is not a fatal error.

INPUT/OUTPUT FILES

• Execution flow

The asm88 inputs assembly source files and outputs relocatable object files, an assembly list file, a cross reference list file and an error list file after assembly.



asm88 execution flow

• Input file

Assembly source file: file_name.ms

This is an assembly source file created by the sap88. In the default of the asm88, ".ms" is set as the input file extension. Although the extension can be changed by specifying an option, do not change the default setting if unnecessary.

• Output files

1. ***Relocatable object file: file_name.o***

   This is the file output from the asm88 after converting the assembly source file to the relocatable S1C88 machine language by the relocatable assembly. This file becomes an input file for the linker link88.

2. ***Assembly list file: file_name.l***

   This is the file in which the machine language converted by assembly and the address are output as a list corresponding to each source statement. The addresses are output as relative addresses that the head of the CODE section or the DATA section in the file assume as "000000H". The creating of this file can be prohibited by a start-up flag.

3. ***Cross reference list file: file_name.x***

   This is a list of addresses in which a symbol has been defined and referred. Creating this file can be prohibited by a start-up flag.

4. ***Error list file: file_name.e***

   This is a list of errors that have been generated during assembly.

START-UP FORMAT

```
asm88 -[all c l o* q RAM# ROM# sig# suf* x] [drive:] <files>↵
```

flags:

Character strings enclosed with [ ] mean flags. Explanations for each flag are discussed later.

drive:

In case the input file is not in current drive, input the drive name in front of the input file name. It can be omitted if the input file is in current drive.

files:

Specify the file name to be input to the asm88. This file name can be input using either capital letters or small letters, and specifying two or more source files is possible. An error will occur when <files> are not specified.

Note: Up to eight characters are available for the source file name. Furthermore, the extension ".ms" must be input.

FLAGS

The asm88 can accept the following flags.
-ROM# and -RAM# should be input using capital letters and the others should be input using small letters.

Function	Flag	Explanation
All symbols output	`-all`	Outputs all symbols including local symbols to a symbol table. In default, only global symbols and undefined symbols are output.
Differentiation between capital and small letters within source program	`-c`	Differentiates capital and small letters within the input source. Since capital and small letters are not differentiated in default, ABC and abc are handled as the same symbol. When this flag is specified, the CPU instructions and the register names must be described using small letters.
Prohibition of assembly list generation	`-l`	Prohibits the creation of an assembly list file. In default, an assembly list file with the extension ".l" is created.
Creating output file	`-o*`	Creates output files with the name "*". In default, the output file name is the same as the input file and the extension becomes ".o" when the input file extension is ".ms". When the input file extension is other than ".ms", the default output file name becomes "xeq". Example: When creating "out.o" from "sample.ms", specify as below. `asm88 -o out.o sample.ms`↵
Suppression of start-up message	`-q`	Does not output any messages related to the assembly processing.
RAM capacity setting	`-RAM#`	Sets the RAM capacity in byte units with #. When the total size of the DATA section exceeds the value set by this flag, an error is output. Example: When the internal RAM capacity is set in 2K (2048 bytes), specify as below. `asm88 -RAM 2048 sample.ms`↵
ROM capacity setting	`-ROM#`	Sets the ROM capacity in byte units with #. When the total size of the CODE section exceeds the value set by this flag, an error is output. Example: When the internal ROM capacity is set in 16K (16384 bytes), specify as below. `asm88 -ROM 16384 sample.ms`↵
Setting character numbers of symbols	`-sig#`	Character numbers of symbols that are significant can be set with a # value. In default the # is set to 15 characters.
Change of input file extension	`-suf*`	Changes the extension of the input file to * (a separator "." is not included). The default is ".ms". Example: When the extension of an input source file (sample.ms) is changed to ".bs", specify as below. `asm88 -suf bs sample.bs`↵
Prohibition of cross reference list file creation	`-x`	Prohibits the creation of a cross reference list file. In default, a cross reference list file with the extension ".x" is created.

When one or more <files> without the -o flag are specified and the file name extension of the input file name is the suffix of the default file name, the asm88 outputs the object files with the same name as the input files and the extension ".o".

```
asm88 file1.ms file2.ms files3.ms
```

By inputting the above, the three object files file1.o, file2.o and file3.o are automatically created. Be aware that the -o flag will not function, when multiple files have been specified for <files>.

ERROR MESSAGE

• Fatal errors

Error message	Description
can't create <file>	<file> cannot be created.
can't open <file>	<file> cannot be opened.
can't read tmp file	Temporary file cannot be read.
can't write tmp file	Temporary file cannot be written.
namelist full	Name list table is full.
no i/p file	There is no input file specification.
insufficient memory	There is not enough memory.
can't seek on vmem file	Seeking of virtual memory file has failed.
can't seek to end of vmem file	Cannot reach the end of virtual memory file.
no swappable page	There is no swap space.
read error on vmem file	Reading of virtual memory file has failed.
write error on vmem file	Writing to virtual memory file has failed.

• Severe errors

Error message	Description
<numeric label> already defined	The numeric label has been defined previously.
<identifier> wrong type	An illegal identifier has appeared.
<token> expected	A token is needed.
' missing	A quotation mark is missing.
attempted division by zero	Attempt has been made to divide by zero.
attempt to redefine <identifier>	Attempt has been made to redefine an identifier.
constant expected	A constant expression is required.
end expected	There is no end instruction.
encountered too early end of line	The line has terminated in the middle.
field overflow	The field to be secured has overflowed.
invalid branch address	An external defined symbol is used for the operand of the short branch instruction.
invalid byte relocation	The byte relocation is invalid.
invalid character	Three is an illegal character.
invalid flag	The flag is invalid.
invalid operand	The operand is invalid.
invalid relocation item	The relocation item is invalid.
invalid register	The register is invalid.
invalid register pair	The register combination is invalid.
invalid symbol define	The symbol definition is invalid.
invalid word relocation	The word relocation is invalid.
new origin incompatible with current psect	There is an absolute origin within the relocatable section (relocatable mode).
non terminated string	The termination of a string cannot be located.
<identifier> not defined	Undefined identifier has appeared.
missing numeric expression	A numeric expression is missing.
cars or jrs out of range	Branch destination by cars or jrs is out of range.
carl or jrl out of range	Branch destination by carl or jrl is out of range.
operand expected	There is no operand.
psect name required	A section name must be specified.
phase error <identifier>	The label address is different between pass 1 and pass 2.
CODE or DATA missing	There is no section setting pseudo-instruction.
ROM capacity overflow	ROM capacity has overflowed.
RAM capacity overflow	RAM capacity has overflowed.
relocation error in expression	A relocation error has appeared within the expression.
<identifier> reserved word	<identifier> is a reserved word.
syntax error <token> expected	Syntax error due to insufficient token(s)
syntax error <token> unexpected	Syntax error due to excess token(s)
syntax error - invalid identifier <identifier>	Syntax error due to an illegal identifier
syntax error <token> invalid in expression	Syntax error due to an illegal token
system error < > <token>	System error due to an illegal token
unsupported instruction	Unsupported instruction has appeared.
unsupported operand	Unsupported operand has appeared.

• Warning errors

Error message	Description
directive is ignored in relocatable mode	The pseudo-instruction is skipped because it is in the relocatable mode.
possibly missing relocatability	Relocatability may lose.
constant overflow	Seven or more digits has been defined for the name.
expected operator	There is no operator (BOC, LOC, POD, LOD).

RETURN VALUE

When there is no syntax error within the input file nor pass 2 error, and all the processing is successfully completed, the asm88 returns "success".

EXAMPLE

Performs relocatable assembly of the file "sample.ms" to simultaneously obtain the list file "sample.l".

```
C>asm88 sample.ms↵
```

C.3 Linker <link88>

PROGRAM NAME

link88.exe

SUMMARY

The link88 links multi-section relocatable object files for the S1C88 and creates an absolute object file. The absolute object file is used to create a program data HEX file that is used for debugging with the ICE by inputting to the binary / HEX converter hex88. It will also be used to create absolute symbol information (rel88) after linking the relocatable assembled file.

The basic functions of the link88 process are as follows.

1) The global flag controls the overall link88 process.

2) It defines the new CODE section and DATA section by the addition of a flag and a file.

3) It relocates sections, rearranging them in optional locations of the physical memory and permits them to be mutually "stacked" (chaining) in appropriate storage boundaries.

4) Each object file input affects the current CODE section and DATA section.

5) The final output starts with the header, thereafter (in the named order) all CODE sections, all DATA sections, symbolic table and the relocation stream for all CODE sections and all DATA sections. The respective component elements for these sorts of outputs are controlled through use of the appropriate global flag which will be described later.

6) Since all the sections are continuous in the linker output, the binary / HEX converter hex88 must be used for writing the section into the appropriate physical location, in order to execute it in a special location within the memory.

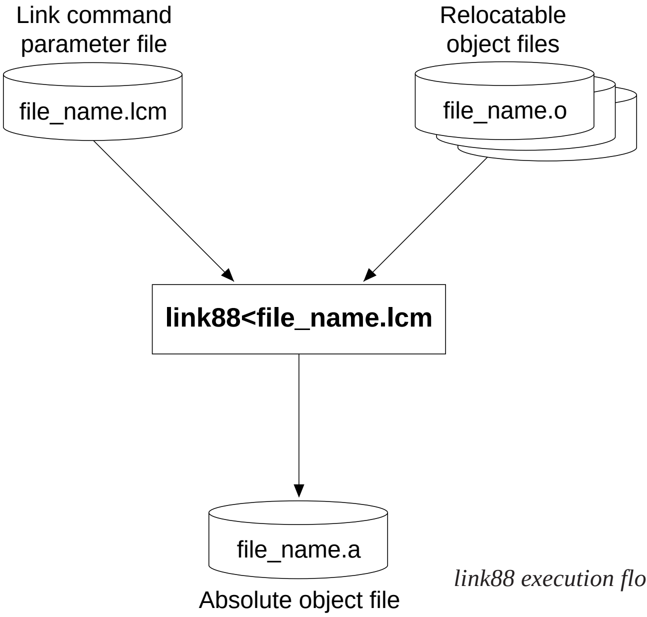
The S1C88 has a 24-bit wide address space (maximum 16M bytes). It splits that address space into a 32K-byte bank (code section) or a 64K-byte page (data section) by controlling the most significant 8-bit by registers such as the code bank register (CB) and the expanded page registers (EP, XP and YP) in an effort to expand the access performance within that range. It is possible to access an optional bank or page from an optional bank or page by rewriting the content of the register, thus permitting easy management of such things as large programs and data bases. However, since the register will not be automatically renewed, even if the bank and the register are crossed, a load module image permitting the 16M-byte address space to be described linearly cannot be created.

The S1C88 adopts a multi-segment system for linking relocatable objects, in order to create load images to be laid out in the optional physical addresses of the address spaces managed by it.

This is a technique in which "All the spaces are split into optional sections of 64K-byte (page) or 32K-byte (bank) units and the address information necessary for the memory layout determines all the address information in accordance with the assignment to each segment unit."
In this technique, since the creation of continuous data objects whose size exceeds 64K bytes (page) and 32K bytes (bank) for one section is not permitted, a limitation is imposed whereby the total size for the CODE sections included in the modules of assembly units cannot exceed 32K bytes and the total size for the DATA section cannot exceed 64K bytes. This restriction reflects the address restriction of the CPU itself and even if a diagnosis of a data overflow generated during assembly were overlooked, it is set up such that it would be rediagnosed during linking.
However, it outputs an error when the size exceeds 64K bytes in default, but does not output when the size exceeds 32K bytes. Consequently, a flag must be specified for judgment when the size exceeds 32K bytes.

INPUT/OUTPUT FILES

• Execution flow



link88 execution flow

• Input files

1. ***Relocatable object file: file_name.o***
   This is a relocatable file in machine language that is output through relocatable assembly with the cross assembler asm88.
2. ***Link command parameter file: file_name.lcm***
   This is a link command parameter file that is directly described by the user.

• Output file

Absolute object file: file_name.a

This is a multi-section object file created by the link88.

Note: Multi-section object file is an absolute object image whose format is composed of a global header, a section descriptor, objects within all CODE sections, objects within all DATA sections, objects within all DEBUG sections, objects within all ZPAG section, a symbolic table, a debug symbolic table, and all relocation information.

START-UP FORMAT

```
link88 -[c cd +dead max## o* q] <sections>
```

<sections> includes one or more following contents.

```
-[+code +data m## p##] [drive:]↵
```

flags:

Character string enclosed with [ ] mean flags. Flags within the first [ ] are global flags and flags within the [ ] included in <sections> are local flags.

drive:

In case of the relocatable object files or the libraries are not in current drive, input the drive name in front of these file names. It can be omitted if these files are in current drive.

Note: The extension for the relocatable object files should be made as ".o".

FLAGS

The link88 can accept the following flags. The flags should be input with small letters.

• Global flags

Function	Flag	Explanation
Distinction between capital and small letters within symbols	`-c`	Distinguishes capital and small letters used for symbols within the relocatable object file. In default, they are not distinguished, therefore ABC and abc are handled as the same symbol.
Deletion of DATA code part	`-cd`	Does not output the code part for the DATA section. -cd is used to create modules that define only symbol values for such purposes as specification of the addresses for the common library.
Listing of undefined symbols	`+dead`	Outputs a list of dead wood symbols on the CRT, that is, symbols that have been defined, but are not referred as absolute.
Setting of maximum section size	`-max##`	Sets the maximum section size at ## bytes. The default value is FFFFFFH (16M bytes). This value is used when sections are linked. When it exceeds this value, an error will occur.
Setting of output file name	`-o*`	Writes the output module on the file *. The default output file name is xeq.
Skip start-up message	`-q`	Does not output any message related to link processing.

When the arguments on the command line are not transferred to the link88, the list of flags and files that become arguments of the link88 are transferred from standard input. When a "-" (hyphen) first appears in the argument list of the command line, a standard input is incorporated into the argument list in place of the "-". The occurrences of "-" following thereafter are disregarded.
The specified <files> are linked in that order.

• Local flags

Flags for sections

Function	Flag	Explanation
Beginning CODE section	`+code`	Begins a new CODE section, then processes the local flag for that section.
Beginning DATA section	`+data`	Begins a new DATA section, then processes the local flag for that section.

A new section of a specified format is not actually created, when the final section of that format has a zero size. However, a new local flag is processed and overwrites the preceding value. These two flags must immediately precede the local flag set to appropriately process the flags and to decide to what flag is to be applied.

Flags used only together with +code or +data

Function	Flag	Explanation
Setting of individual section size	`-m##`	Sets the maximum size of the individual segment as ## bytes. The default size is 8000H (CODE section) or 10000H (DATA section). An error will occur if the section size exceeds this setting value.
Physical address setting	`-p##`	Sets the physical address of the beginning of the section as ##.

ERROR MESSAGES

Error message	Description
bad file format: 'FILE NAME'	Format of the input file 'FILE NAME' is incorrect.
bad relocation item	There is long integer type relocation information.
bad symbol number: 'NUMBER'	'NUMBER' is detected as illegal symbol code.
can't create 'FILE NAME'	The file 'FILE NAME' cannot be created.
can't create tmp file	Temporary file cannot be created.
can't open: 'FILE NAME'	The input file 'FILE NAME' cannot be opened.
can't read binary header: 'FILE NAME'	Header of the file 'FILE NAME' cannot be read.
can't read file header: 'FILE NAME'	First two bytes of the file 'FILE NAME' cannot be read.
can't read symbol table: 'FILE NAME'	Symbol table cannot be read from the file 'FILE NAME'.
can't read tmp file	Temporary file cannot be read.
can't write output file	Cannot write into output file.
can't write tmp file	Cannot write into temporary file.
field overflow	Branch destination by cars or jrs is out of range.
inquiry phase error: 'SYMBOL NAME'	Symbol value of the 'SYMBOL NAME' is different between pass 1 and pass 2.
link: early EOF in pass2	Unexpected EOF is detected during pass 2 processing.
multiply defined 'SYMBOL NAME'	'SYMBOL NAME' is multiply defined.
no object files	No input object files exist.
no relocation bits: 'FILE NAME'	The relocation information corresponding to the file 'FILE NAME' is suppressed.
'SECTION NAME' overflow	The section size in the 'SECTION NAME' exceeds the upper limit value.
phase error: 'SYMBOL NAME'	Symbol value of the 'SYMBOL NAME' is different between pass 1 and pass 2.
previous reference blocked: 'SYMBOL NAME' range error	The information related relocation bit width is unmatched.
read error in pass2	Read error is generated during pass 2 processing.
undefined 'SYMBOL NAME'	'SYMBOL NAME' has not been defined.

RETURN VALUE

When an error message is not output to the standard output, in other words, no undefined symbol remains and all reads and writes have succeeded, the link88 returns "success". If not, it returns "failure".

EXAMPLE

Links the sample.o by the link88 via standard input.

```
A>link88↵
-o c88xxx.a +code -p0x100 +data -p0x8000↵
sample.o↵
^Z↵
A>
A>link88 < sample.lcm↵
```

C.4 Symbol Information Generator <rel88>

PROGRAM NAME

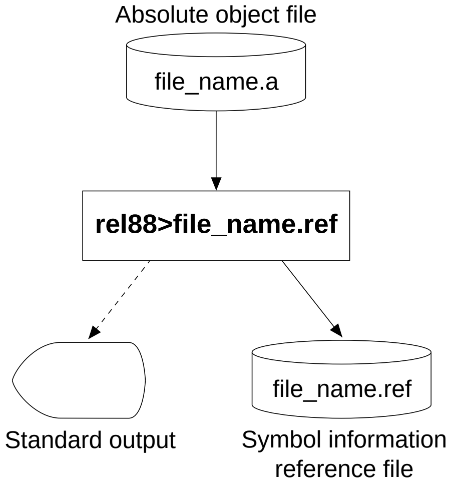
rel88.exe

SUMMARY

The rel88 checks the multi-section relocatable objects. The files that become the object of such checks are relocatable object files output by the cross assembler asm88 and absolute object files output by the link88. The rel88 can be used to check the size and configuration of relocatable object files and to output symbol information in absolute object files output from the link88.

INPUT/OUTPUT FILES

- **Execution flow**



- **Input file**

Absolute object file: file_name.a

Inputs an absolute object file created by the link88.

- **Output file**

Standard output or
Symbol information reference file: file_name.ref

The rel88 outputs a symbol information reference file that is allocated in the physical address from the absolute object file.

rel88 execution flow

START-UP FORMAT

rel88 -[a +dec d g +in +sec v] [drive:] <files>↵

flags:

Character strings enclosed with [ ] mean flags. Explanations for each flag are discussed later.

drive:

In case an input file is not in current drive, input the drive name in front of the input file name. It can be omitted if an input file is in current drive.

files:

Specify the file name to be input into the rel88. This file name can be input using either capital or small letters and specifying two or more files is possible. An error will occur when <files> is not specified.

FLAGS

The rel88 can accept the following flags. The flags should be input with small letters.

Function	Flag	Explanation
Sorting of symbol names	`-a`	Sorts outputs in alphabetical order of the symbol names.
Decimal output	`+dec`	Outputs symbol values and segment sizes in decimal numbers. The default is a hexadecimal number.
Output of defined symbols	`-d`	Outputs all defined symbols within each file, one per line. The symbol value, the "relocation code" showing to what the value is related and the symbol name are entered on each line. Values are output in the number of digits needed to indicate the integers in the S1C88. The meanings of the relocation codes in the outputs are as follows. • C indicates CODE relativity • D indicates DATA relativity • A indicates absolute (not relocatable) • ? indicates rel88 cannot recognize it. Small letters are used to indicate local symbols. Capital letters are used for global symbols.
Output only global symbols	`-g`	Outputs global symbols only.
Standard input	`+in`	Takes <files> from standard input and adds them to command line. Redirecting is also possible and is valid when many files are specified.
Physical address and size of multi-section	`+sec`	Outputs the physical address and size of each section of multi-segment output files.
Sorting by symbol values	`-v`	Sorts the inside of section by symbol values. The aforementioned -d flag is tacitly specified. Symbols that have the same value are sorted in alphabetic order. Absolute (non-relocatable) symbols are displayed first and are followed by CODE relative symbols and DATA relative symbols.

<files> are zero, or one or more files and they must have a multi-section format. When two or more files are specified, the name of each file or module precedes the information that is output pertaining to it. Each name is followed by a colon and a new-line. When there is no <files> specification, or when a "-" appears on the command line, xeq is used as an input file.

ERROR MESSAGE

Error message	Description
can't read binary header	Reading of the object header excluding magic number and configuration byte has failed.
can't read header	Reading of the first two bytes of the object header (magic number and configuration byte) has failed.
can't read symbol table	Reading of the symbolic table in the object has failed.

RETURN VALUE

When a diagnostic message has not been created (in other words, when all the reads have succeeded and all the file formats are valid), rel88 returns "success".

EXAMPLE

Obtains a list of all the symbols within the module in alphabetic order in hexadecimal numbers.

```
C>rel88 -a alloc.o↵
0x0074C _alloc
0x0000D _exit
0x01feC _free
0x00beC _nalloc
0x0000D _sbreak
0x0000D _write
```

NOTE

When no symbol is in the object or local symbols only exist, rel88 outputs a "no memory" message. However, the local symbols are registered in the symbolic table by setting the -all flag of the asm88 (all symbols output). If you wish to refer to all symbols, set the -all flag of the asm88.

C.5 Symbolic Table File Generator <sym88>

PROGRAM NAME

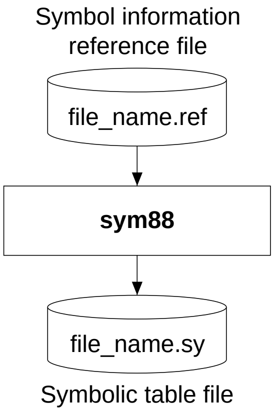
sym88.exe

SUMMARY

The symbolic table file generator sym88 converts a symbolic information file (file_name.ref) generated in file redirect with the symbol information generating utility rel88 to a symbolic table file (file_name.sy) that can be referenced in the ICE. Loading the symbolic table file and the corresponding relocatable assembly program file in the ICE makes symbolic debugging possible.

INPUT/OUTPUT FILE

• Execution flow



sym88 execution flow

• Input file

Symbol information reference file: file_name.ref

Inputs a symbol information reference file created by the rel88.

• Output file

Symbolic table file: file_name.sy

The sym88 converts a symbol information file into a format that can be loaded to the ICE and outputs a symbolic table file.

START-UP FORMAT

sym88 <file>↵

file:

Specify the symbol information file (.ref) to be input to the sym88.
This file name can be input using either capital letters or small letters.
An error will occur when <file> is not specified.

ERROR MESSAGE

Error message	Description
No Input File	Input file ".ref" has not been specified.

RETURN VALUE

The sym88 returns "success" if there is no error in the input file and an output file is created. If there is an error in the input file or internal created file, "failure" is returned.

EXAMPLE

Converts the symbol information reference file sample.ref into the symbolic table file sample.sy.

```
A:\>sym88 sample.ref↵
```

NOTES

1. Drives and directories for input files can not be specified in the startup command of the sym88. Therefore, be sure to start up the sym88 after setting the directory of the input file as the current directory.

2. The sym88 does not check the format of the input file. Therefore, the symbol information file to be input to the sym88 must only be generated using the symbol information generating utility rel88 with the flags shown below.

```
A:\>rel88 -v +sec sample.a>sample.ref
```

C.6 Binary/HEX Converter <hex88>

PROGRAM NAME

hex88.exe

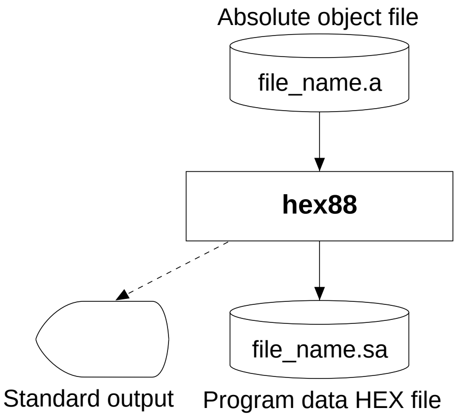
SUMMARY

The hex88 converts an absolute object file created by the link88 into a hexadecimal data conversion format (program data HEX file). This system adopted Motorola S record format. An absolute object file is read from the <ifile>. When an <ifile> is not assigned, or when an assigned file name is a "-" (hyphen), file xeq is read.
Further, S2 format in Motorola S record (can convert up to 3-byte address) is used since the S1C88 has a maximum 16M-byte address space (000000–FFFFFFH).

INPUT/OUTPUT FILES

• Execution flow

The hex88 is a tool to convert an absolute object file output from the linker (link88) into a program data HEX file in hexadecimal format. The execution flow is shown below.



hex88 execution flow

• Input file

Absolute object file: file_name.a

File to be input into the hex88 is an absolute object file output from linker.

• Output file

Standard output
or Program data HEX file: file_name.sa

The hex88 converts an absolute object file to an ASCII file that can be input to the unused area filling utility fil88XXX.

START-UP FORMAT

hex88 -[o*] [drive:] <ifile>↵

flag:

Character string enclosed with [ ] means flag. Explanations for the flag is discussed later.

drive:

In case an absolute object file is not in current drive, input the drive name in front of the file name. It can be omitted if an input file is in current drive.

ifile:

Specify the file name input to the hex88. This file name can be input using either capital or small letters. When an <ifile> is not assigned, or when an assigned file name is a "-" (hyphen), file xeq is read.

Note: The extension for the absolute object file should be made as ".a".

FLAG

The hex88 can accept the following flag. The flag should be input with small letters.

Function	Flag	Explanation
Output file specification	`-o*`	Writes the output module for the file *. The default is standard output. (hex88 fixed setting flag)

ERROR MESSAGE

Error message	Description
bad file format	Input file format is incorrect.
can't read <input file>	Reading of the <input file> has failed.
can't write <output file>	Writing to the <output file> has failed.

RETURN VALUE

If an error message is not printed, in other words if all the records have meanings, and all the reading and writing is successful, the hex88 returns "success". Otherwise, the hex88 returns "failure".

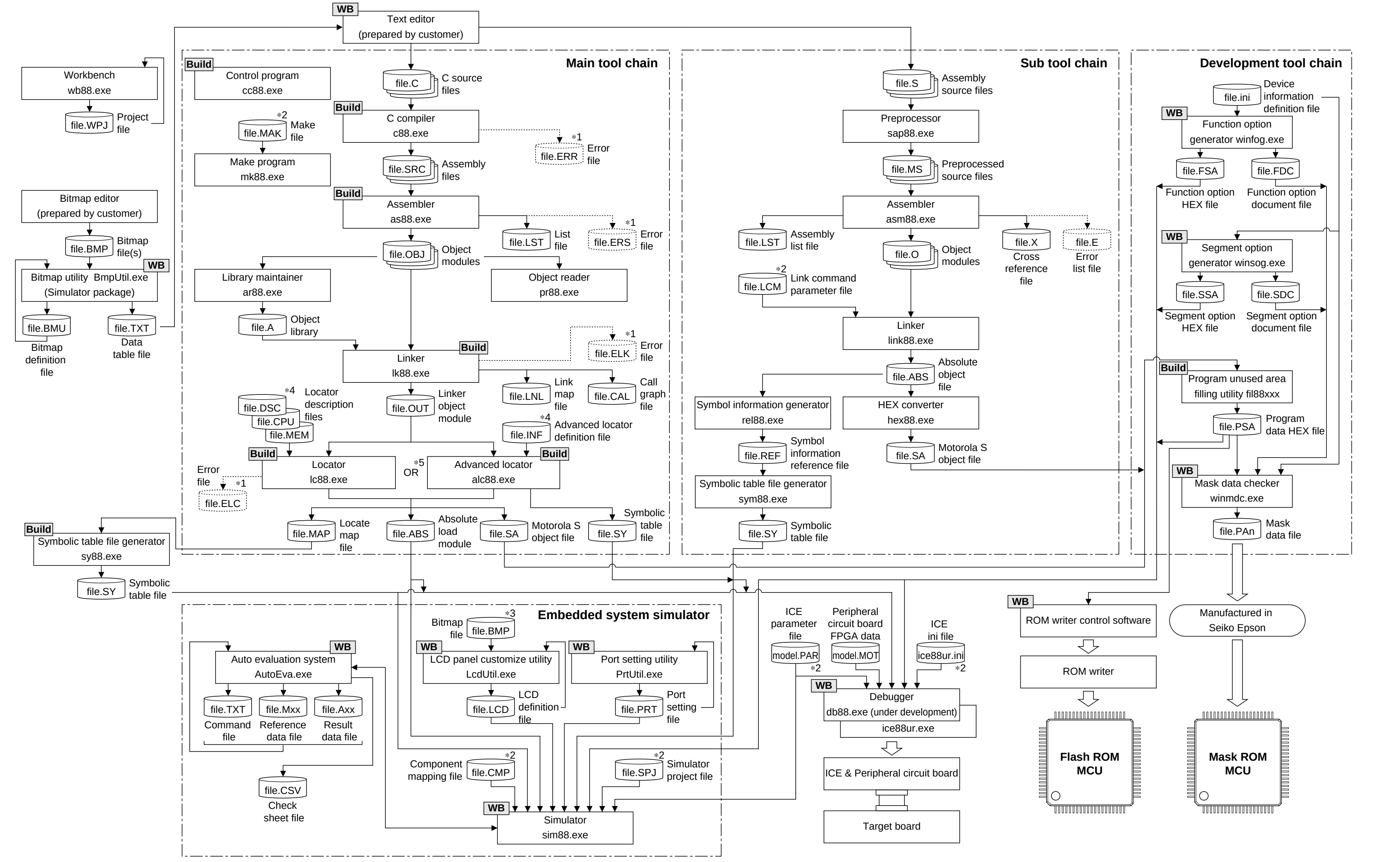
EXAMPLE

Converts the absolute object file sample.a into the program data HEX file in the Motorola S2 format.

```
A>hex88 -o sample.sa sample.a↵
```

S1C88 Family Development Tools

Quick Reference



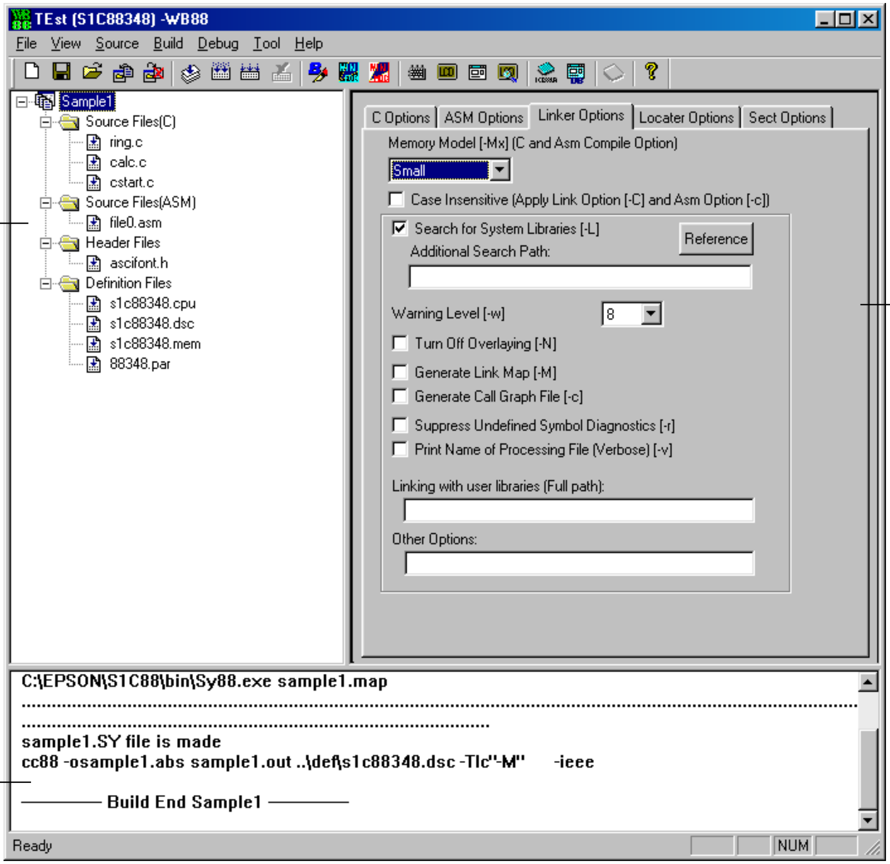
WB Can be invoked from the workbench wb88. Build Tools executed automatically during build process by wb88.

*1: If the error file is generated, wb88 displays the contents of the file in the message view and allows a tag jump function. *2: Created using a text editor. *3: Created using a bitmap editor. *4: Created using the wb88 section editor (or a text editor). *5: Selected by wb88.

Outline

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

Windows



Sample1
- Source Files(C)
  - ring.c
  - calc.c
  - cstart.c
- Source Files(ASM)
  - file0.asm
- Header Files
  - ascifont.h
- Definition Files
  - s1c88348.cpu
  - s1c88348.dsc
  - s1c88348.mem
  - 88348.par

Memory Model [-Mx] (C and Asm Compile Option): Small

□ Case Insensitive (Apply Link Option [-C] and Asm Option [-c])

☑ Search for System Libraries [-L]

Additional Search Path:

Warning Level [-w]: 8

□ Turn Off Overlaying [-N]

□ Generate Link Map [-M]

□ Generate Call Graph File [-c]

□ Suppress Undefined Symbol Diagnostics [-r]

□ Print Name of Processing File (Verbose) [-v]

Linking with user libraries (Full path):

Other Options:

```
C:\EPSON\S1C88\bin\Sy88.exe sample1.map
.........................................................................................................................................................................................
..................................................................................................
sample1.SY file is made
cc88 -osample1.abs sample1.out ..\def\s1c88348.dsc -Tlc''-M''     -ieee

———— Build End Sample1 ————
```

Project view

This area shows the currently opened work space folder and lists all the files that can be edited by the user in the project, with a structure similar to Windows Explorer.

Double-clicking a source file icon invokes the specified editor to open the source file.

Option view

This area displays the selected options of the C compiler, assembler, linker, locator and segment editor, and also allows option selection.

The option view changes its display contents according to the selection in the project view (whether node or file) as well as clicking a tool name tab.

Message view

This area displays the messages delivered from the executed tools in a build or compile process.

Double-clicking a syntax error message with a source line number displayed in this window invokes the specified editor. The editor opens the corresponding source and displays the source line in which the error has occurred (available when an editor with the tag jump function that can be specified by wb88 is used).

Buttons

Tool bar

[New Project] button
Creates a new project.

[Save Project] button
Saves the project being edited. The file will be overwritten.
This button becomes inactive if a project is not opened.

[Insert a file] button
Inserts the specified source/header file into the current opened project.
This button becomes inactive if a project is not opened.

[Remove a file] button
Removes the selected file from the project.

[Open] button
Opens a document. A dialog box will appear allowing selection of the file to be opened.
When a source or header file is selected, the specified editor activates and opens the file.

[Compile/Assemble] button
Compiles or assembles the source file selected in the option view according to the source format.

[Build] button
Builds the currently opened project using a general make process.

[Rebuild] button
Builds the currently opened project. All the source files will be compiled/assembled regardless of whether they are updated or not.

[Stop Build] button
Stops the build process being executed.

[BMPUtil] button
Invokes the bitmap utility BmpUtil.

[WinFOG] button
Invokes the function option generator winfog.

[WinMDC] button
Invokes the mask data checker winmdc.

Tool bar

[PrtUtil] button
Invokes the port setting utility PrtUtil.

[LCDUtil] button
Invokes the LCD panel customize utility LCDUtil.

[Sim88] button
Invokes the simulator Sim88.

[AutoEva] button
Invokes the auto evaluation system AutoEva.

[ICE88UR] button
Invokes the ice88ur debugger.

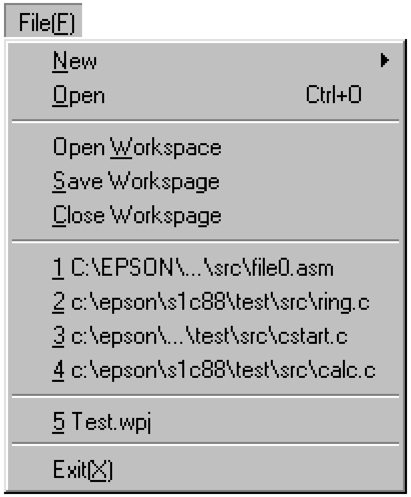
[DB88] button
Invokes the db88 debugger.

[ROM Writer] button
Invokes the on-board ROM writer control software.

[About] button
Displays the version of wb88.

Menus

[File] menu


File(F)
New
Open Ctrl+O
Open Workspace
Save Workspage
Close Workspage
1 C:\EPSON\...\src\file0.asm
2 c:\epson\s1c88\test\src\ring.c
3 c:\epson\...\test\src\cstart.c
4 c:\epson\s1c88\test\src\calc.c
5 Test.wpj
Exit(X)

The file names listed in this menu are recently used source and project files. Selecting one opens the file.

New - C Source File
Creates a new C source file.
(Invokes editor)

New - Asm Source File
Creates a new assembly source file.
(Invokes editor)

New - Header File
Creates a new header file.
(Invokes editor)

New - Project
Creates a new project.

Open ([Ctrl]+[O])
Opens a source file, header file or project file.

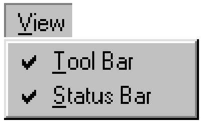
Open Workspace
Opens a project.

Save Workspace
Saves the currently opened project.

Close Workspace
Closes the currently opened project.

Exit
Terminates wb88.

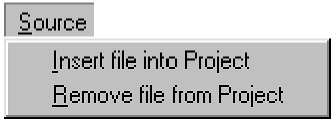
[View] menu



Tool Bar
Shows or hides the tool bar.

Status Bar
Shows or hides the status bar.

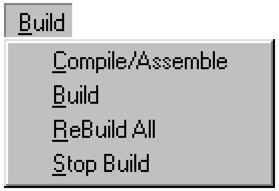
[Source] menu



Insert file into Project
Adds the specified source file in the currently opened project.

Remove file from Project
Removes the source file selected in the Project view from the currently opened project.

[Build] menu



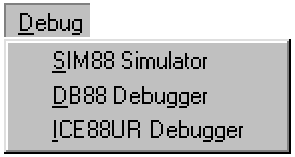
Compile/Assemble
Compiles or assembles the source file selected in the option view according to the source format.

Build
Builds the currently opened project using a general make process.

ReBuild All
Builds the currently opened project.

Stop Build
Stops the build process being executed.

[Debug] menu

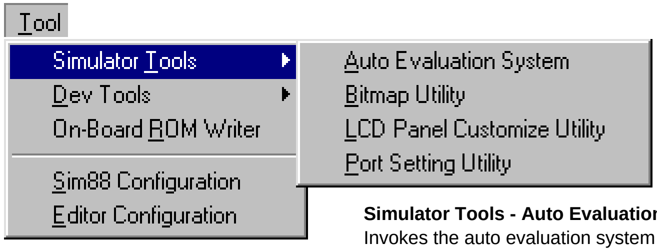


Sim88 Simulator
Invokes the Sim88 simulator.

DB88 Debugger
Invokes the db88 debugger.

ICE88UR Debugger
Invokes the ice88ur debugger.

[Tools] menu



Simulator Tools - Auto Evaluation System
Invokes the auto evaluation system AutoEva.

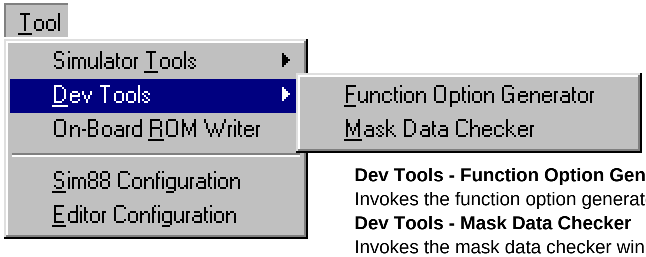
Simulator Tools - Bitmap Utility
Invokes the bitmap utility BmpUtil.

Simulator Tools - LCD Panel Customize Utility
Invokes the LCD panel customize utility LCDUtil.

Simulator Tools - Port Setting Utility
Invokes the port setting utility PrtUtil.

Menus

[Tools] menu



Dev Tools - Function Option Generator
Invokes the function option generator winfog.

Dev Tools - Mask Data Checker
Invokes the mask data checker winmdc.

On-Board ROM Writer
Invokes the on-board ROM writer control software.

Sim88 Configuration
Displays a dialog box for setting the path to the simulator Sim88.exe.

Editor Configuration
Displays a dialog box for setting the editor path and the command line options.

[Help] menu

About WB88
Displays a dialog box showing the version of the work bench.

Error Messages

System error

not enough memory	There is insufficient memory to run wb88.

Error output when generating a project

The file is not a WB88 project file. (<filename>)	The file <filename> is not a wb88 project file.
The version of the project file is not supported. (<filename>)	This version of the project file <filename> is not supported.
Unable to create a project : cannot access. <filename>	Unable to generate a project because the file <filename> could not be accessed correctly.
Unable to create a project : Unable to copy DEF file.(<filename>)	Unable to generate a project because wb88 failed to copy the definition file <filename>.
The project is already existed.(<filename>)	Unable to create a project because the file <filename> already exists. Two or more projects with the same name cannot be created in the same folder.

Error Messages

Error output when generating a project

Unable to create a project : Dev Directory of S1C88 family package does not exist.	Unable to create a project because no DEV directories exist. The DEV directory of the package contains various definition files required for build task. No projects can be built without this directory.

Error output when adding files to the project

The file cannot be added to the project. It is not a C file.(<filename>)	The file <filename> cannot be added to the project because it is not a C source file.
The file cannot be added to the project. It is not an ASM file.(<filename>)	The file <filename> cannot be added to the project because it is not an assembly source file.
The file cannot be added to the project. It is not a header file.(<filename>)	The file <filename> cannot be added to the project because it is not a header file.
The file is already existed in the project. It cannot be added in the project.(<filename>)	The file <filename> cannot be added to the project because it already exists.
WB88 does not support such source file type. (<filename>)	This source type file is not supported by wb88.

File error

Failed to access the file.(<filename>)	Failed to operate on the file <filename>.
Unable to open the file.(<filename>)	Failed to open the file <filename>.

Error output when starting a tool

Unable to execute ICE88UR.exe : Unable to access <filename>.	Cannot start S5U1C88000H5 because wb88 could not access the file <filename>.
Unable to execute Sim88 : Unable to access the DEF file.(<filename>)	Cannot start Sim88 because wb88 could not access the definition file.
Unable to execute <toolname>.	Unable to start <toolname>.

Error output when building

Select a C or an ASM file.	Select a C source or assembly source file. Before source files can be compiled, you must select the target file from tree view.
Build Command needs an active project.	The build target must be project.
No target file is found in the project.	No target files to build are found in the project. Source files must be registered to a project before they can be built.

Other error

The command needs an active project.	The command requires a project. This error message is displayed if, in the absence of a project, a function is executed for which a project must be present.

Startup Command

```
c88 [[option]...[file]...]...
```

Options

Include options

-f *file*	Read options from *file*
-H *file*	Include *file* before starting compilation
-I*directory*	Look in *directory* for include files

Preprocess options

-D*macro*[=*def*]	Define preprocessor *macro*

Code generation options

-M{**s**|**c**|**d**|**l**}	Select memory model: small, compact code, compact data or large
-O{**0**|**1**}	Control optimization

Output file options

-e	Remove output file if compiler errors occur
-o *file*	Specify name of output *file*
-s	Merge C-source code with assembly output

Diagnostic options

-V	Display version header only
-err	Send diagnostics to error list file (`.err`)
-g	Enable symbolic debug information
-w[*num*]	Suppress one or all warning messages

Error/Warning Messages

I: information E: error F: fatal error S: internal compiler error W: warning

Frontend

F 1:	evaluation expired	Your product evaluation period has expired.
W 2:	unrecognized option: '*option*'	The option you specified does not exist.
E 4:	expected *number* more '#endif'	The preprocessor part of the compiler found the '**#if**', '**#ifdef**' or '**#ifndef**' directive but did not find a corresponding '**#endif**' in the same source file.
E 5:	no source modules	You must specify at least one source file to compile.
F 6:	cannot create "*file*"	The output file or temporary file could not be created.
F 7:	cannot open "*file*"	Check if the file you specified really exists.
F 8:	attempt to overwrite input file "*file*"	The output file must have a different name than the input file.
E 9:	unterminated constant character or string	This error can occur when you specify a string without a closing double-quote (") or when you specify a character constant without a closing single-quote (').
F 11:	file stack overflow	This error occurs if the maximum nesting depth (50) of file inclusion is reached.
F 12:	memory allocation error	All free space has been used.
W 13:	prototype after forward call or old style declaration - ignored	Check that a prototype for each function is present before the actual call.
E 14:	';' inserted	An expression statement needs a semicolon.
E 15:	missing filename after -o option	The **-o** option must be followed by an output filename.
E 16:	bad numerical constant	A constant must conform to its syntax. Also, a constant may not be too large to be represented in the type to which it was assigned.
E 17:	string too long	This error occurs if the maximum string size (1500) is reached.
E 18:	illegal character (0x*hexnumber*)	The character with the hexadecimal ASCII value 0x*hexnumber* is not allowed here.
E 19:	newline character in constant	The newline character can appear in a character constant or string constant only when it is preceded by a backslash (\).
E 20:	empty character constant	A character constant must contain exactly one character. Empty character constants ('') are not allowed.
E 21:	character constant overflow	A character constant must contain exactly one character. Note that an escape sequence is converted to a single character.
E 22:	'#define' without valid identifier	You have to supply an identifier after a '**#define**'.

Error/Warning Messages

Frontend

E 23:	'#else' without '#if'	'**#else**' can only be used within a corresponding '**#if**', '**#ifdef**' or '**#ifndef**' construct.
E 24:	'#endif' without matching '#if'	'**#endif**' appeared without a matching '**#if**', '**#ifdef**' or '**#ifndef**' preprocessor directive.
E 25:	missing or zero line number	'**#line**' requires a non-zero line number specification.
E 26:	undefined control	A control line (line with a '#*identifier*') must contain one of the known preprocessor directives.
W 27:	unexpected text after control	'**#ifdef**' and '**#ifndef**' require only one identifier. Also, '**#else**' and '**#endif**' only have a newline. '**#undef**' requires exactly one identifier.
W 28:	empty program	The source file must contain at least one external definition. A source file with nothing but comments is considered an empty program.
E 29:	bad '#include' syntax	A '**#include**' must be followed by a valid header name syntax.
E 30:	include file "*file*" not found	Be sure you have specified an existing include file after a '**#include**' directive. Make sure you have specified the correct path for the file.
E 31:	end-of-file encountered inside comment	The compiler found the end of a file while scanning a comment. Probably a comment was not terminated.
E 32:	argument mismatch for macro "*name*"	The number of arguments in invocation of a function-like macro must agree with the number of parameters in the definition. Also, invocation of a function-like macro requires a terminating ")" token.
E 33:	"*name*" redefined	The given identifier was defined more than once, or a subsequent declaration differed from a previous one.
W 34:	illegal redefinition of macro "*name*"	A macro can be redefined only if the body of the redefined macro is exactly the same as the body of the originally defined macro.
E 35:	bad filename in '#line'	The string literal of a **#line** (if present) may not be a "**wide-char**" string.
W 36:	'debug' facility not installed	'**#pragma debug**' is only allowed in the debug version of the compiler.
W 37:	attempt to divide by zero	A divide or modulo by zero was found.
E 38:	non integral switch expression	A `switch` condition expression must evaluate to an integral value.
F 39:	unknown error number: *number*	This error may not occur.
W 40:	non-standard escape sequence	Your escape sequence contains an illegal escape character.

Frontend

E 41:	'#elif' without '#if'	The '**#elif**' directive did not appear within an '**#if**', '**#ifdef**' or '**#ifndef**' construct.
E 42:	syntax error, expecting parameter type/declaration/ statement	A syntax error occurred in a parameter list a declaration or a statement.
E 43:	unrecoverable syntax error, skipping to end of file	The compiler found an error from which it could not recover.
I 44:	in initializer "*name*"	Informational message when checking for a proper constant initializer.
E 46:	cannot hold that many operands	The value stack may not exceed 20 operands.
E 47:	missing operator	An operator was expected in the expression.
E 48:	missing right parenthesis	')' was expected.
W 49:	attempt to divide by zero - potential run-time error	An expression with a divide or modulo by zero was found.
E 50:	missing left parenthesis	'(' was expected.
E 51:	cannot hold that many operators	The state stack may not exceed 20 operators.
E 52:	missing operand	An operand was expected.
E 53:	missing identifier after 'defined' operator	An identifier is required in a #if defined(*identifier*).
E 54:	non scalar controlling expression	Iteration conditions and '**if**' conditions must have a scalar type (not a struct, union or a pointer).
E 55:	operand has not integer type	The operand of a '**#if**' directive must evaluate to an integral constant.
W 56:	'<*debugoption*><*level*>' no associated action	There is no associated debug action with the specified debug option and level.
W 58:	invalid warning number: *number*	The warning number you supplied to the **-w** option does not exist.
F 59:	sorry, more than number errors	Compilation stops if there are more than 40 errors.
E 60:	label "*label*" multiple defined	A label can be defined only once in the same function.
E 61:	type clash	The compiler found conflicting types.
E 62:	bad storage class for "*name*"	The storage class specifiers `auto` and `register` may not appear in declaration specifiers of external definitions. Also, the only storage class specifier allowed in a parameter declaration is `register`.
E 63:	"*name*" redeclared	The specified identifier was already declared. The compiler uses the second declaration.

Error/Warning Messages

Frontend

E 64:	incompatible redeclaration of "*name*"	The specified identifier was already declared.
W 66:	function "*name*": variable "*name*" not used	A variable is declared which is never used.
W 67:	illegal suboption: *option*	The suboption is not valid for this option.
W 68:	function "*name*": parameter "*name*" not used	A function parameter is declared which is never used.
E 69:	declaration contains more than one basic type specifier	Type specifiers may not be repeated.
E 70:	'break' outside loop or switch	A `break` statement may only appear in a `switch` or a loop (`do`, `for` or `while`).
E 71:	illegal type specified	The type you specified is not allowed in this context.
W 72:	duplicate type modifier	Type qualifiers may not be repeated in a specifier list or qualifier list.
E 73:	object cannot be bound to multiple memories	Use only one memory attribute per object.
E 74:	declaration contains more than one class specifier	A declaration may contain at most one storage class specifier.
E 75:	'continue' outside a loop	`continue` may only appear in a loop body (`do`, `for` or `while`).
E 76:	duplicate macro parameter "*name*"	The given identifier was used more than one in the format1 parameter list of a macro definition.
E 77:	parameter list should be empty	An identifier list, not part of a function definition, must be empty.
E 78:	'void' should be the only parameter	Within a function prototype of a function that does not except any arguments, void may be the only parameter.
E 79:	constant expression expected	A constant expression may not contain a comma. Also, the bit field width, an expression that defines an `enum`, array-bound constants and `switch case` expressions must all be integral constant expressions.
E 80:	'#' operator shall be followed by macro parameter	The '#' operator must be followed by a macro argument.
E 81:	'##' operator shall not occur at beginning or end of a macro	The '##' (token concatenation) operator is used to paste together adjacent preprocessor tokens, so it cannot be used at the beginning or end of a macro body.
W 86:	escape character truncated to 8 bit value	The value of a hexadecimal escape sequence (a backslash, \, followed by a 'x' and a number) must fit in 8 bits storage.
E 87:	concatenated string too long	The resulting string was longer than the limit of 1500 characters.
W 88:	"*name*" redeclared with different linkage	The specified identifier was already declared.

Frontend

E 89:	illegal bitfield declarator	A bit field may only be declared as an integer, not as a pointer or a function for example.
E 90:	#error *message*	The *message* is the descriptive text supplied in a '**#error**' preprocessor directive.
W 91:	no prototype for function "*name*"	Each function should have a valid function prototype.
W 92:	no prototype for indirect function call	Each function should have a valid function prototype.
I 94:	hiding earlier one	Additional message which is preceded by error E 63. The second declaration will be used.
F 95:	protection error: *message*	Something went wrong with the protection key initialization.
E 96:	syntax error in #define	`#define id(` requires a right-parenthesis ')'.
E 97:	"..." incompatible with old-style prototype	If one function has a parameter type list and another function, with the same name, is an old-style declaration, the parameter list may not have ellipsis.
E 98:	function type cannot be inherited from a typedef	A `typedef` cannot be used for a function definition.
F 99:	conditional directives nested too deep	'**#if**', '**#ifdef**' or '**#ifndef**' directives may not be nested deeper than 50 levels.
E 100:	case or default label not inside switch	The `case:` or `default:` label may only appear inside a `switch`.
E 101:	vacuous declaration	Something is missing in the declaration.
E 102:	duplicate case or default label	Switch case values must be distinct after evaluation and there may be at most one `default:` label inside a `switch`.
E 103:	may not subtract pointer from scalar	The only operands allowed on subtraction of pointers is pointer - pointer, or pointer - scalar.
E 104:	left operand of operator has not struct/union type	The first operand of a '**.**' or '**->**' must have a `struct` or `union` type.
E 105:	zero or negative array size - ignored	Array bound constants must be greater than zero.
E 106:	different constructors	Compatible function types with parameter type lists must agree in number of parameters and in use of ellipsis. Also, the corresponding parameters must have compatible types.
E 107:	different array sizes	Corresponding array parameters of compatible function types must have the same size.
E 108:	different types	Corresponding parameters must have compatible types and the type of each prototype parameter must be compatible with the widened definition parameter.

Error/Warning Messages

Frontend

E 109:	floating point constant out of valid range	A floating point constant must have a value that fits in the type to which it was assigned.
E 110:	function cannot return arrays or functions	A function may not have a return type that is of type array or function. A pointer to a function is allowed.
I 111:	parameter list does not match earlier prototype	Check the parameter list or adjust the prototype. The number and type of parameters must match.
E 112:	parameter declaration must include identifier	If the declarator is a prototype, the declaration of each parameter must include an identifier. Also, an identifier declared as a `typedef` name cannot be a parameter name.
E 114:	incomplete struct/union type	The `struct` or `union` type must be known before you can use it.
E 115:	label "*name*" undefined	A `goto` statement was found, but the specified label did not exist in the same function or module.
W 116:	label "*name*" not referenced	The given label was defined but never referenced. The reference of the label must be within the same function or module.
E 117:	"*name*" undefined	The specified identifier was not defined. A variable's type must be specified in a declaration before it can be used.
W 118:	constant expression out of valid range	A constant expression used in a case label may not be too large. Also when converting a floating point value to an integer, the floating point constant may not be too large.
E 119:	cannot take 'sizeof' bitfield or void type	The size of a bit field or `void` type is not known. So, the size of it cannot be taken.
E 120:	cannot take 'sizeof' function	The size of a function is not known. So, the size of it cannot be taken.
E 121:	not a function declarator	This is not a valid function.
E 122:	unnamed formal parameter	The parameter must have a valid name.
W 123:	function should return something	A return in a non-`void` function must have an expression.
E 124:	array cannot hold functions	An array of functions is not allowed.
E 125:	function cannot return anything	A `return` with an expression may not appear in a `void` function.
W 126:	missing return (function "*name*")	A non-`void` function with a non-empty function body must have a `return` statement.
E 129:	cannot initialize "*name*"	Declarators in the declarator list may not contain initializations. Also, an `extern` declaration may have no initializer.
W 130:	operands of operator are pointers to different types	Pointer operands of an operator or assignment ('**=**'), must have the same type.

Frontend

E 131:	bad operand type(s) of *operator*	The operator needs an operand of another type.
W 132:	value of variable "*name*" is undefined	This warning occurs if a variable is used before it is defined.
E 133:	illegal struct/union member type	A function cannot be a member of a `struct` or `union`. Also, bit fields may only have type `int` or `unsigned`.
E 134:	bitfield size out of range - set to 1	The bit field width may not be greater than the number of bits in the type and may not be negative.
W 135:	statement not reached	The specified statement will never be executed.
E 138:	illegal function call	You cannot perform a function call on an object that is not a function.
E 139:	*operator* cannot have aggregate type	The type name in a (cast) must be a scalar (not a `struct`, `union` or a pointer) and also the operand of a (cast) must be a scalar.
E 140:	*type* cannot be applied to a register/bit/bitfield object or builtin/inline function	For example, the '**&**' operator (address) cannot be used on registers and bit fields.
E 141:	*operator* requires modifiable lvalue	The operand of the '**++**', or '**--**' operator and the left operand of an assignment or compound assignment (**lvalue**) must be modifiable.
E 143:	too many initializers	There may be no more initializers than there are objects.
W 144:	enumerator "*name*" value out of range	An `enum` constant exceeded the limit for an `int`.
E 145:	requires enclosing curly braces	A complex initializer needs enclosing curly braces.
E 146:	argument #*number*: memory spaces do not match	With prototypes, the memory spaces of arguments must match.
W 147:	argument #*number*: different levels of indirection	With prototypes, the types of arguments must be assignment compatible.
W 148:	argument #*number*: struct/union type does not match	With prototypes, both the prototyped function argument and the actual argument was a `struct` or `union`, but they have different tags. The tag types should match.
E 149:	object "*name*" has zero size	A struct or union may not have a member with an incomplete type.
W 150:	argument #*number*: pointers to different types	With prototypes, the pointer types of arguments must be compatible.
W 151:	ignoring memory specifier	Memory specifiers for a struct, union or enum are ignored.
E 152:	operands of *operator* are not pointing to the same memory space	Be sure the operands point to the same memory space.

Error/Warning Messages

Frontend

E 153:	'sizeof' zero sized object	An implicit or explicit `sizeof` operation references an object with an unknown size.
E 154:	argument #*number*: struct/union mismatch	With prototypes, only one of the prototyped function argument or the actual argument was a `struct` or `union`. The types should match.
E 155:	casting lvalue '*type*' to '*type*' is not allowed	The operand of the '**++**', or '**--**' operator or the left operand of an assignment or compound assignment (**lvalue**) may not be cast to another type.
E 157:	"*name*" is not a formal parameter	If a declarator has an identifier list, only its identifiers may appear in the declarator list.
E 158:	right side of *operator* is not a member of the designated struct/union	The second operand of '**.**' or '**->**' must be a member of the designated `struct` or `union`.
E 160:	pointer mismatch at *operator*	Both operands of *operator* must be a valid pointer.
E 161:	aggregates around *operator* do not match	The contents of the structs, unions or arrays on both sides of the *operator* must be the same.
E 162:	*operator* requires an lvalue or function designator	The '**&**' (address) operator requires an **lvalue** or function designator.
W 163:	operands of *operator* have different level of indirection	The types of pointers or addresses of the operator must be assignment compatible.
E 164:	operands of *operator* may not have type 'pointer to void'	The operands of *operator* may not have operand (`void *`).
W 165:	operands of *operator* are incompatible: pointer vs. pointer to array	The types of pointers or addresses of the operator must be assignment compatible. A pointer cannot be assigned to a pointer to array.
E 166:	*operator* cannot make something out of nothing	Casting type `void` to something else is not allowed.
E 170:	recursive expansion of inline function "*name*"	An _inline function may not be recursive.
E 171:	too much tail-recursion in inline function "*name*"	If the function level is greater than or equal to 40 this error is given.
W 172:	adjacent strings have different types	When concatenating two strings, they must have the same type.
E 173:	'void' function argument	A function may not have an argument with type `void`.
E 174:	not an address constant	A constant address was expected. Unlike a static variable, an automatic variable does not have a fixed memory location and therefore, the address of an automatic is not a constant.
E 175:	not an arithmetic constant	In a constant expression no assignment operators, no '**++**' operator, no '**--**' operator and no functions are allowed.

Frontend

E 176:	address of automatic is not a constant	Unlike a static variable, an automatic variable does not have a fixed memory location and therefore, the address of an automatic is not a constant.
W 177:	static variable "*name*" not used	A static variable is declared which is never used.
W 178:	static function "name" not used	A static function is declared which is never called.
E 179:	inline function "*name*" is not defined	Possibly only the prototype of the inline function was present, but the actual inline function was not.
E 180:	illegal target memory (*memory*) for pointer	The pointer may not point to *memory*.
W 182:	argument #*number*: different types	With prototypes, the types of arguments must be compatible.
I 185:	(prototype synthesized at line *number* in "*name*")	This is an informational message containing the source file position where an old-style prototype was synthesized.
E 186:	array of type bit is not allowed	An array cannot contain bit type variables.
E 187:	illegal structure definition	A structure can only be defined (initialized) if its members are known.
E 188:	structure containing bit-type fields is forced into bitaddressable area	This error occurs when you use a bitaddressable storage type for a structure containing bit-type members.
E 189:	pointer is forced to bitaddressable, pointer to bitaddressable is illegal	A pointer to bitaddressable memory is not allowed.
W 190:	"long float" changed to "float"	In ANSI C floating point constants are treated having type `double`, unless the constant has the suffix '**f**'.
E 191:	recursive struct/union definition	A `struct` or `union` cannot contain itself.
E 192:	missing filename after -f option	The **-f** option requires a filename argument.
E 194:	cannot initialize typedef	You cannot assign a value to a `typedef` variable.
F 199:	demonstration package limits exceeded	The demonstration package has certain limits which are not present in the full version.
W 200:	unknown pragma - ignored	The compiler ignores pragmas that are not known.
W 201:	"*name*" cannot have storage type - ignored	A `register` variable or an automatic/parameter cannot have a storage type.
E 202:	"*name*" is declared with 'void' parameter list	You cannot call a function with an argument when the function does not accept any (`void` parameter list).
E 203:	too many/few actual parameters	With prototyping, the number of arguments of a function must agree with the prototype of the function.

Error/Warning Messages

Frontend

W 204:	U suffix not allowed on floating constant - ignored	A floating point constant cannot have a '**U**' or '**u**' suffix.
W 205:	F suffix not allowed on integer constant - ignored	An integer constant cannot have a '**F**' or '**f**' suffix.
E 206:	'*name*' named bit-field cannot have 0 width	A bit field must be an integral constant expression with a value greater than zero.
E 212:	"*name*": missing static function definition	A function with a `static` prototype misses its definition.
W 303:	variable '*name*' possibly uninitialized	Possibly an initialization statement is not reached, while a function should return something.
E 327:	too many arguments to pass in registers for _asmfunc '*name*'	An `_asmfunc` function uses a fixed register-based interface between C and assembly, but the number of arguments that can be passed is limited by the number of available registers. With function *name* this limit was reached.

Backend

W 501:	function qualifier used on non-function	A function qualifier can only be used on functions.
E 502:	Intrinsic function '_int()' needs an immediate value as parameter	The argument of the `_int()` intrinsic function must be an integral constant expression rather than any type of integral expression.
E 503:	Intrinsic function '_jrsf()' needs an immediate value 0..3	The given number must be a constant value between 0 and 3.
W 508:	function qualifier duplicated	Only one function qualifier is allowed.
E 511:	interrupt function must have void result and void parameter list	A function declared with `_interrupt(`*`n`*`)` may not accept any arguments and may not return anything.
W 512:	'*number*' illegal interrupt number (0, or 3 to 251) - ignored	The interrupt vector number must be 0, or in the range 3 to 251. Any other number is illegal.
E 513:	calling an interrupt routine, use '_swi()'	An interrupt function cannot be called directly, you must use the intrinsic function `_swi()`.
E 514:	conflict in '_interrupt'/ '_asmfunc' attribute	The attributes of the current function qualifier declaration and the previous function qualifier declaration are not the same.
E 515:	different '_interrupt' number	The interrupt number of the current function qualifier declaration and the previous function qualifier declaration are not the same.
E 516:	'*memory_type*' is illegal memory for function	The storage type is not valid for this function.

Backend

W 517:	conversion of long address to short address	This warning is issued when pointer conversion is needed.
F 524:	illegal memory model	See the compiler usage for valid arguments of the **-M** option.
E 526:	function qualifier '_asmfunc' not allowed in function definition	`_asmfunc` is only allowed in the function prototype.
E 528:	_at() requires a numerical address	You can only use an expression that evaluates to a numerical address.
E 529:	_at() address out of range for this type of object	The absolute address is not present in the specified memory space.
E 530:	_at() only valid for global variables	Only global variables can be placed on absolute addresses.
E 531:	_at() only allowed for uninitialized variables	Absolute variables cannot be initialized.
E 532:	_at() has no effect on external declaration	When declared `extern` the variable is not allocated by the compiler.
W 533:	c88 language extension keyword used as identifier	A language extension keyword is a reserved word, and reserved words cannot be used as an identifier.
E 536:	illegal syntax used for default section name '*name*' in -R option	See the description of the **-R** option for the correct syntax.
E 537:	default section name '*name*' not allowed	See the description of the **-R** option for the correct syntax.
W 538:	default section name '*name*' already renamed to '*new_name*'	Only use one of the **-R** option or the renamesect pragma or use another name.
W 542:	optimization stack underflow, no optimization options are saved with #pragma optimize	This warning occurs if you use a `#pragma endoptimize` while there were no options saved by a previous `#pragma endoptimize`.
W 555:	current optimization level could reduce debugging comfort (-g)	You could have **HLL** debug conflicts with these optimization settings.
E 560:	Float/Double: not yet implemented	Floating point will be supported in a following version.

C Compiler c88 (7)

Library

<ctype.h>	isalnum, isalpha, isascii, iscntrl, isdigit, isgraph, islower, isprint, ispunct, isspace, isupper, isxdigit, toascii, _tolower, tolower, _toupper, toupper
<errno.h>	Error numbers No C functions.
<float.h>	Constants for floating-point operation
<limits.h>	Limits and sizes of integral types No C functions.
<locale.h>	localeconv, setlocale Delivered as skeletons.
<math.h>	acos, asin, atan, atan2, ceil, cos, cosh, exp, fabs, floor, fmod, frexp, ldexp, log, log10, modf, pow, sin, sinh, sqrt, tan, tanh
<setjmp.h>	longjmp, setjmp
<signal.h>	raise, signal Functions are delivered as skeletons.
<simio.h>	_simi, _simo
<stdarg.h>	va_arg, va_end, va_start
<stddef.h>	offsetof, definition of special types
<stdio.h>	clearerr, fclose, _fclose, feof, ferror, fflush, fgetc, fgetpos, fgets, fopen, _fopen, fprintf, fputc, fputs, fread, freopen, fscanf, fseek, fsetpos, ftell, fwrite, getc, getchar, gets, _ioread, _iowrite, _lseek, perror, printf, putc, putchar, puts, _read, remove, rename, rewind, scanf, setbuf, setvbuf, sprintf, sscanf, tmpfile, tmpnam, ungetc, vfprintf, vprintf, vsprintf, _write
<stdlib.h>	abort, abs, atexit, atof, atoi, atol, bsearch, calloc, div, exit, free, getenv, labs, ldiv, malloc, mblen, mbstowcs, mbtowc, qsort, rand, realloc, srand, strtod, strtol, strtoul, system, wcstombs, wctomb
<string.h>	memchr, memcmp, memcpy, memmove, memset, strcat, strchr, strcmp, strcol, strcpy, strcspn, strerror, strlen, strncat, strncmp, strncpy, strpbrk, strrchr, strspn, strstr, strtok, strxfrm
<time.h>	asctime, clock, ctime, difftime, gmtime, localtime, mktime, strftime, time All functions are delivered as skeletons.

Startup Command

```
as88 [option]...source-file [map-file]
```

Options

-C *file*	Include *file* before source
-D*macro*[=*def*]	Define preprocessor *macro*
-L[*flag*...]	Remove specified source lines from list file
-M[**s**|**c**|**d**|**l**]	Specify memory model
-V	Display version header only
-c	Switch to case insensitive mode (default case sensitive)
-e	Remove object file on assembly errors
-err	Redirect error messages to error file
-f *file*	Read options from *file*
-i[**l**|**g**]	Default label style local or global
-l	Generate listing file
-o *filename*	Specify name of output file
-t	Display section summary
-v	Verbose mode. Print the filenames and numbers of the passes while they progress
-w[*num*]	Suppress one or all warning messages

Functions

```
@function_name(argument[,argument]...)
```

Mathematical Functions

ABS	Absolute value
MAX	Maximum value
MIN	Minimum value
SGN	Return sign

String Functions

CAT	Catenate strings
LEN	Length of string
POS	Position of substring in string
SCP	Compare strings
SUB	Substring from a string

Macro Functions

ARG	Macro argument function
CNT	Macro argument count
MAC	Macro definition function
MXP	Macro expansion function

Assembler Mode Functions

AS88	Assembler executable name
DEF	Symbol definition function
LST	LIST control flag value
MODEL	Selected model of the assembler

Address Handling Functions

CADDR	Code address
COFF	Code page offset
CPAG	Code page number
DADDR	Data address
DOFF	Data page offset
DPAG	Data page number
HIGH	256 byte page number
LOW	256 byte page offset

Assembler Directives

Debugging

CALLS	Pass call information to object file. Used to build a call tree at link time for overlaying overlay sections.
SYMB	Pass symbolic debug information

Assembly Control

ALIGN	Specify alignment
COMMENT	Start comment lines. This directive is not permitted in IF/ELIF/ELSE/ENDIF constructs and MACRO/DUP definitions.
DEFINE	Define substitution string
DEFSECT	Define section name and attributes
END	End of source program
FAIL	Programmer generated error message
INCLUDE	Include secondary file
MSG	Programmer generated message
RADIX	Change input radix for constants
SECT	Activate section
UNDEF	Undefine DEFINE symbol
WARN	Programmer generated warning

Symbol Definition

EQU	Equate symbol to a value; accepts forward references
EXTERN	External symbol declaration; also permitted in module body
GLOBAL	Global symbol declaration; also permitted in module body
LOCAL	Local symbol declaration
NAME	Identify object file
SET	Set symbol to a value; accepts forward references

Data Definition/Storage Allocation

ASCII	Define ASCII string
ASCIZ	Define NULL padded ASCII string
DB	Define constant byte
DS	Define storage
DW	Define constant word

Macros and Conditional Assembly

DUP	Duplicate sequence of source lines
DUPA	Duplicate sequence with arguments
DUPC	Duplicate sequence with characters
DUPF	Duplicate sequence in loop
ENDIF	End of conditional assembly
ENDM	End of macro definition
EXITM	Exit macro
IF	Conditional assembly directive
MACRO	Macro definition
PMACRO	Purge macro definition

Error Messages

Warnings (W)

W 101:	use *option* at the start of the source; ignored	Primary options must be used at the start of the source.
W 102:	duplicate attribute "*attribute*" found	An attribute of an EXTERN directive is used twice or more. Remove one of the duplicate attributes.
W 104:	expected an attribute but got *attribute*; ignored	
W 105:	section activation expected, use *name* directive	Use the SECT directive to activate a section.
W 106:	conflicting attributes specified "*attributes*"	You used two conflicting attributes in an EXTERN statement directive.
W 107:	memory conflict on object "*name*"	A label or other object is explicit or implicit defined using incompatible memory types.
W 108:	object attributes redefinition "*attributes*"	A label or other object is explicit or implicit defined using incompatible attributes.
W 109:	label "*label*" not used	The label *label* is defined with the GLOBAL directive and neither defined nor referred, or the label is defined with the LOCAL directive and not referenced.
W 110:	extern label "*label*" defined in module, made global	The label *label* is defined with an EXTERN directive and defined as a label in the source. The label will be handled as a global label.
W 111:	unknown $LIST flag "*flag*"	You supplied an unknown *flag* to the $LIST control.
W 112:	text found after END; ignored	An END directive designates the end of the source file. All text after the END directive will be ignored.
W 113:	unknown $MODEL specifier; ignored	You supplied an unknown model.
W 114:	$MODEL may only be specified once, it remains "*model*"; ignored	You supplied more than one model.
W 115:	use ON or OFF after control name	The control you specified must have either ON or OFF after the control name.
W 116:	unknown parameter "*parameter*" for *control-name* control	See the description of the control for the allowed parameters.
W 118:	inserted "*extern name*"	The symbol *name* is used inside an expression, but not defined with an EXTERN directive.
W 119:	"*name*" section has not the MAX attribute; ignoring RESET	

Warnings (W)

W 120:	assembler debug information: cannot emit non-tiof expression for *label*	The SYMB record contains an expression with operations that are not supported by the IEEE-695 object format.
W 121:	changed alignment size to *size*	
W 123:	expression: *type-error*	The expression performs an illegal operation on an address or combines incompatible memory spaces.
W 124:	cannot purge macro during its own definition	
W 125:	"*symbol*" is not a DEFINE symbol	You tried to UNDEF a symbol that was not previously DEFINEd or was already undefined.
W 126:	redefinition of "*define-symbol*"	The symbol is already DEFINEd in the current scope. The symbol is redefined according to this DEFINE.
W 127:	redefinition of macro "*macro*"	The macro is already defined. The macro is redefined according to this macro definition.
W 128:	number of macro arguments is less than definition	You supplied less arguments to the macro than when defining it.
W 129:	number of macro arguments is greater than definition	You supplied more arguments to the macro than when defining it.
W 130:	DUPA needs at least one value argument	The DUPA directive needs at least two arguments, the dummy parameter and a value parameter.
W 131:	DUPF increment value gives empty macro	The step value supplied with the DUPF macro will skip the DUPF macro body.
W 132:	IF started in previous file "*file*", line *line*	The ENDIF or ELSE pre-processor directive matches with an IF directive in another file.
W 133:	currently no macro expansion active	The @CNT() and @ARG() functions can only be used inside a macro expansion.
W 134:	"*directive*" is not supported, skipped	The supplied directive is not supported by this assembler.
W 135:	define symbol of "*define-symbol*" is not an identifier; skipped definition	You supplied an illegal identifier with the **-D** option on the command line.
W 137:	label "*label*" defined *attribute* and *attribute*	The label is defined with an EXTERN and a GLOBAL directive.
W 138:	warning: *WARN-directive-arguments*	Output from the WARN directive.
W 139:	expression must be between *hex-value* and *hex-value*	
W 140:	expression must be between *value* and *value*	

Error Messages

Warnings (W)

W 141:	*global/local* label "*name*" not defined in this module; made extern	The label is declared and used but not defined in the source file.
W 170:	code address maps to zero page	The code offset you specified to the @CPAG function is in the zero page.
W 171:	address offset must be between 0 and FFFF	The offset you specified in the @CADDR or @DADDR function was too large.
W 172:	page number must be between 0 and FF	The page number you specified in the @CADDR or @DADDR function was too large.

Errors (E)

E 200:	*message*; halting assembly	The assembler stops the further processing of your source file.
E 201:	unexpected newline or line delimiter	The syntax checker found a newline or line delimiter that does not confirm to the assembler grammar.
E 202:	unexpected character: '*character*'	The syntax checker found a character that does not confirm to the assembler grammar.
E 203:	illegal escape character in string constant	The syntax checker found an illegal escape character in the string constant that does not confirm to the assembler grammar.
E 204:	I/O error: open intermediate file failed (*file*)	The assembler opens an intermediate file to optimize the lexical scanning phase. The assembler cannot open this file.
E 205:	syntax error: expected *token* at *token*	The syntax checker expected to find a token but found another token.
E 206:	syntax error: *token* unexpected	The syntax checker found an unexpected token.
E 207:	syntax error: missing ':'	The syntax checker found a label definition or memory space modifier but missed the appended semi-colon.
E 208:	syntax error: missing ')'	The syntax checker expected to find a closing parentheses.
E 209:	invalid radix value, should be 2, 8, 10 or 16	The RADIX directive accepts only 2, 8, 10 or 16.
E 210:	syntax error	The syntax checker found an error.
E 211:	unknown model	Substitute the correct model, one of s, c, d or l.
E 212:	syntax error: expected *token*	The syntax checker expected to find a token but found nothing.
E 213:	label "*label*" defined *attribute* and *attribute*	The label is defined with a LOCAL and a GLOBAL or EXTERN directive.
E 214:	illegal addressing mode	The mnemonic used an illegal addressing mode.
E 215:	not enough operands	The mnemonic needs more operands.
E 216:	too many operands	The mnemonic needs less operands.

Errors (E)

E 217:	*description*	There was an error found during assembly of the mnemonic.
E 218:	unknown mnemonic: "*name*"	The assembler found an unknown mnemonic.
E 219:	this is not a hardware instruction (use $OPTIMIZE OFF "H")	The assembler found a generic instruction, but the **-Oh** (hardware only) option or the $OPTIMIZE ON "H" control was specified.
E 223:	unknown section "*name*"	The section name specified with a SECT directive has not (yet) been defined with a DEFSECT directive.
E 224:	unknown label "*name*"	A label was used which was not defined.
E 225:	invalid memory type	You supplied an invalid memory modifier.
E 226:	unknown symbol attribute: *attribute*	
E 227:	invalid memory attribute	The assembler found an unknown location counter or memory mapping attribute.
E 228:	*attr* attribute needs a number	The attribute *attr* needs an extra parameter.
E 229:	only one of the *name* attributes may be specified	
E 230:	invalid section attribute: *name*	The assembler found an unknown section attribute.
E 231:	absolute section, expected "AT" expression	An absolute section must be specified using an 'AT *address*' expression.
E 232:	MAX/OVERLAY sections need to be named sections	Sections with the MAX or OVERLAY attribute must have a name, otherwise the locator cannot overlay the sections.
E 233:	*type* section cannot have *attribute* attribute	Code sections may not have the CLEAR or OVERLAY attribute.
E 234:	section attributes do not match earlier declaration	In an previous definition of the same section other attributes were used.
E 235:	redefinition of section	An absolute section of the same name can only be located once.
E 236:	cannot evaluate expression of *descriptor*	Some functions and directives must evaluate their arguments during assembly.
E 237:	*descriptor* directive must have positive value	Some directives need to have a positive argument.
E 238:	Floating point numbers not allowed with DB directive	The DB directive does not accept floating point numbers.
E 239:	byte constant out of range	The DB directive stores expressions in bytes.
E 240:	word constant out of range	The DW directive stores expressions in words.
E 241:	Cannot emit non tiof functions, replaced with integral value '0'	Floating point expressions and some functions can not be represented in the IEEE-695 object format.
E 242:	the *name* attribute must be specified	A section must have the CODE or DATA attribute.

Error Messages

Errors (E)

E 243:	use $OBJECT OFF or $OBJECT "*object-file*"	
E 244:	unknown control "*name*"	The specified control does not exist.
E 246:	ENDM within IF/ENDIF	The assembler found an ENDM directive within an IF/ENDIF pair.
E 247:	illegal condition code	The assembler encountered an illegal condition code within an instruction.
E 248:	cannot evaluate origin expression of org "*name*: *address*"	All origins of absolute sections must be evaluated before creation of the object file.
E 249:	incorrect argument types for function "*function*"	The supplied argument(s) evaluated to a different type than expected.
E 250:	tiof function not yet implemented: "*function*"	The supplied object format function is not yet implemented.
E 251:	@POS(,,*start*) start argument past end of string	The *start* argument is larger than the length of the string in the first parameter.
E 252:	second definition of label "*label*"	The label is defined twice in the same scope.
E 253:	recursive definition of symbol "*symbol*"	The evaluation of the symbol depends on its own value.
E 254:	missing closing '>' in include directive	The syntax checker missed the closing '>' bracket in the INCLUDE directive.
E 255:	could not open include file *include-file*	The assembler could not open the given include-file.
E 256:	integral divide by zero	The expression contains an divide by zero.
E 257:	unterminated string	All strings must end on the same line as they are started.
E 258:	unexpected characters after macro parameters, possible illegal white space	Spaces are not permitted between macro parameters.
E 259:	COMMENT directive not permitted within a macro definition and conditional assembly	This assembler does not permit the usage of the COMMENT directive within MACRO/DUP definitions or IF/ELSE/ENDIF constructs.
E 260:	definition of "*macro*" unterminated, missing "endm"	The macro definition is not terminated with an ENDM directive.
E 261:	macro argument name may not start with an '_'	MACRO and DUP arguments may not start with an underscore.
E 262:	cannot find "*symbol*"	Could not find a definition of the argument of a '%' or '?' operator within a macro expansion.
E 263:	cannot evaluate: "*symbol*", value is unknown at this point	The symbol used with a '%' or '?' operator within a macro expansion has not been defined.

Errors (E)

E 264:	cannot evaluate: "*symbol*", value depends on an unknown symbol	Could not evaluate the argument of a '%' or '?' operator within a macro expansion.
E 265:	cannot evaluate argument of *dup* (unknown or location dependant symbols)	The arguments of the DUP directive could not be evaluated.
E 266:	*dup* argument must be integral	The argument of the DUP directive must be integral.
E 267:	*dup* needs a parameter	Check the syntax of the DUP directive.
E 268:	ENDM without a corresponding MACRO or DUP definition	The assembler found an ENDM directive without an corresponding MACRO or DUP definition.
E 269:	ELSE without a corresponding IF	The assembler found an ELSE directive without an corresponding IF directive.
E 270:	ENDIF without a corresponding IF	The assembler found an ENDIF directive without an corresponding IF directive.
E 271:	missing corresponding ENDIF	The assembler found an IF or ELSE directive without an corresponding ENDIF directive.
E 272:	label not permitted with this directive	Some directives do not accept labels.
E 273:	wrong number of arguments for *function*	The function needs more or less arguments.
E 274:	illegal argument for *function*	An argument has the wrong type.
E 275:	expression not properly aligned	
E 276:	immediate value must be between *value* and *value*	The immediate operand of the instruction does only accept values in the given range.
E 277:	address must be between $*address* and $*address*	The address operand is not in the range mentioned.
E 278:	operand must be an address	The operand must be an address but has no address attributes.
E 279:	address must be short	
E 280:	address must be short	The operand must be an address in the short range.
E 281:	illegal option "*option*"	The assembler found an unknown or misspelled command line option.
E 282:	"Symbols:" part not found in map file "*name*"	The map file may be incomplete.
E 283:	"Sections:" part not found in map file "*name*"	The map file may be incomplete.
E 284:	module "*name*" not found in map file "*name*"	The map file may be incomplete.

Error Messages

Errors (E)

E 285:	*file-kind* file will overwrite *file-kind* file	The assembler warns when one of its output files will overwrite the source file you gave on the command line or another output file.
E 286:	$CASE options must be given before any symbol definition	The $CASE options may only be given before any symbol is defined.
E 287:	symbolic debug error: *message*	The assembler found an error in a symbolic debug (SYMB) instruction.
E 288:	error in PAGE directive: *message*	The arguments supplied to the PAGE directive do not conform to the restrictions.
E 290:	fail: *message*	Output of the FAIL directive. This is an user generated error.
E 291:	generated check: *message*	Integrity check for the coupling between the C compiler and assembler.
E 293:	expression out of range	An instruction operand must be in a specified address range.
E 294:	expression must be between *hexvalue* and *hexvalue*	
E 295:	expression must be between *value* and *value*	
E 296:	optimizer error: *message*	The optimizer found an error.
E 297:	jump address must be a code address	Jumps and jump-subroutines must have a target address in code memory.
E 298:	size depends on location, cannot evaluate	The size of some constructions (notably the align directives) depend on the memory address.

Fatal Error (F)

F 401:	memory allocation error	A request for free memory is denied by the system. All memory has been used.
F 402:	duplicate input filename "*file*" and "*file*"	The assembler requires one input filename on the command line.
F 403:	error opening *file-kind* file: "*file-name*"	The assembler could not open the given file.
F 404:	protection error: *message*	No protection key or not a IBM compatible PC.
F 405:	I/O error	The assembler cannot write its output to a file.
F 406:	parser stack overflow	
F 407:	symbolic debug output error	The symbolic debug information is incorrectly written in the object file.
F 408:	illegal operator precedence	The operator priority table is corrupt.
F 409:	Assembler internal error	The assembler encountered internal inconsistencies.

Fatal Error (F)

F 410:	Assembler internal error: duplicate mufom "*symbol*" during rename	The assembler renames all symbols local to a scope to unique symbols. In this case the assembler did not succeed into making an unique name.
F 411:	symbolic debug error: "*message*"	An error occurred during the parsing of the SYMB directive.
F 412:	macro calls nested too deep (possible endless recursive call)	There is a limit to the number of nested macro expansions. Currently this limit is set to 1000.
F 413:	cannot evaluate "*function*"	A function call is encountered although it should have been processed.
F 414:	cannot recover from previous errors, stopped	Due to earlier errors the assembler internal state got corrupted and stops assembling your program.
F 415:	error opening temporary file	The assembler uses temporary files for the debug information and list file generation. It could not open or create one of those temporary files.
F 416:	internal error in optimizer	The optimizer found a deadlock situation. Try to assemble without any optimization options. Please fill out the error report form and send it to Seiko Epson.

Startup Command

```
lk88 [option]...file...
```

Options

-C	Link case insensitive (default case sensitive)
-L *directory*	Additional search path for system libraries
-L	Skip system library search
-M	Produce a link map (`.lnl`)
-N	Turn off overlaying
-O *name*	Specify basename of the resulting map files
-V	Display version header only
-c	Produce a separate call graph file (`.cal`)
-e	Clean up if erroneous result
-err	Redirect error messages to error file (`.elk`)
-f *file*	Read command line information from *file*, '-' means `stdin`
-l *x*	Search also in system library `lib`*x*`.a`
-o *filename*	Specify name of output file
-r	Suppress undefined symbol diagnostics
-u *symbol*	Enter *symbol* as undefined in the symbol table
-v or **-t**	Verbose option. Print name of each file as it is processed
-w *n*	Suppress messages above warning level *n*

Error Messages

Warnings (W)

W 100:	Cannot create map file *filename*, turned off -M option	The given file could not be created.
W 101:	Illegal filename (*filename*) detected	A filename with an illegal extension was detected.
W 102:	Incomplete type specification, type index = T*hexnumber*	An unknown type reference.
W 103:	Object name (*name*) differs from filename	Internal name of object file not the same as the filename.
W 104:	'-o *filename*' option overwrites previous '-o *filename*'	Second **-o** option encountered, previous name is lost.
W 105:	No object files found	No files where specified at the invocation.
W 106:	No search path for system libraries. Use -L or env "*variable*"	System library files (those given with the **-l** option) must have a search path, either supplied by means of the environment, or by means of the option **-L**.
W 108:	Illegal option: *option* (-H or -\? for help)	An illegal option was detected.
W 109:	Type not completely specified for symbol <*symbol*> in *file*	Not a complete type specification in either the current file or the mentioned file.
W 110:	Compatible types, different definitions for symbol <*symbol*> in *file*	Name conflict between compatible types.
W 111:	Signed/unsigned conflict for symbol <*symbol*> in *file*	Size of both types is correct, but one of the types contains an unsigned where the other uses a signed type.
W 112:	Type conflict for symbol <*symbol*> in *file*	A real type conflict.
W 113:	Table of contents of *file* out of date, not searched. (Use ar ts <*name*>)	The **ar** library has a symbol table which is not up to date.
W 114:	No table of contents in *file*, not searched. (Use ar ts <*name*>)	The **ar** library has no symbol table.
W 115:	Library *library* contains ucode which is not supported	Ucode is not supported by the linker.
W 116:	Not all modules are translated with the same threshold (-G value)	The library file has an unknown format, or is corrupted.
W 117:	No type found for <*symbol*>. No type check performed	No type has been generated for the symbol.

Error Messages

Warnings (W)

W 118:	Variable <*name*>, has incompatible external addressing modes with file <*filename*>	A variable is not yet allocated but two external references are made by non overlapping addressing modes.
W 119:	error from the Embedded Environment: *message*, switched off relaxed addressing mode check	If the embedded environment is readable for the linker, the addressing mode check is relaxed. For instance, a variable defined as data may be accessed as huge.

Errors (E)

E 200:	Illegal object, assignment of non existing var *var*	The MUFOM variable did not exist. Corrupted object file.
E 201:	Bad magic number	The magic number of a supplied library file was not ok.
E 202:	Section *name* does not have the same attributes as already linked files	Named section with different attributes encountered.
E 203:	Cannot open *filename*	A given file was not found.
E 204:	Illegal reference in address of *name*	Illegal MUFOM variable used in value expression of a variable. Corrupted object file.
E 205:	Symbol '*name*' already defined in <*name*>	A symbol was defined twice.
E 206:	Illegal object, multi assignment on *var*	The MUFOM variable was assigned more than once probably due to a previous error 'already defined', E 205.
E 207:	Object for different processor characteristics	Bits per MAU, MAU per address or endian for this object differs with the first linked object.
E 208:	Found unresolved external(s):	There were some symbols not found.
E 209:	Object format in *file* not supported	The object file has an unknown format, or is corrupted.
E 210:	Library format in *file* not supported	The library file has an unknown format, or is corrupted.
E 211:	Function <*function*> cannot be added to the already built overlay pool <*name*>	The overlay pool has already been built in a previous linker action.
E 212:	Duplicate absolute section name <*name*>	Absolute sections begin on a fixed address. They cannot be linked.
E 213:	Section <*name*> does not have the same size as the already linked one	A section with the EQUAL attribute does not have the same size as other, already linked, sections.
E 214:	Missing section address for absolute section <*name*>	Each absolute section must have a section address command in the object. Corrupted object file.

Errors (E)

E 215:	Section <*name*> has a different address from the already linked one	Two absolute sections may be linked (overlaid) on some conditions. They must have the same address.
E 216:	Variable <*name*>, name <*name*> has incompatible external addressing modes	A variable is allocated outside a referencing addressing space.
E 217:	Variable <*name*>, has incompatible external addressing modes with file <*filename*>	A variable is not yet allocated but two external references are made by non overlapping addressing modes.
E 218:	Variable <*name*>, also referenced in <*name*> has an incompatible address format	An attempt was made to link different address formats between the current file and the mentioned file.
E 219:	Not supported/illegal *feature* in object format *format*	An option/feature is not supported or illegal in given object format.
E 220:	page size (0x*hexvalue*) overflow for section <*name*> with size 0x*hexvalue*	Section is too big to fit into the page.
E 221:	*message*	Error generated by the object.
E 222:	Address of <*name*> not defined	No address was assigned to the variable. Corrupted object file.

Fatal Errors (F)

F 400:	Cannot create file *filename*	The given file could not be created.
F 401:	Illegal object: Unknown command at offset *offset*	An unknown command was detected in the object file. Corrupted object file.
F 402:	Illegal object: Corrupted hex number at offset *offset*	Wrong byte count in hex number. Corrupted object file.
F 403:	Illegal section index	A section index out of range was detected. Corrupted object file.
F 404:	Illegal object: Unknown hex value at offset *offset*	An unknown variable was detected in the object file. Corrupted object file.
F 405:	Internal error *number*	Internal fatal error.
F 406:	*message*	No key no IBM compatible PC.
F 407:	Missing section size for section <*name*>	Each section must have a section size command in the object. Corrupted object file.
F 408:	Out of memory	An attempt to allocate more memory failed.
F 409:	Illegal object, offset *offset*	Inconsistency found in the object module.

Error Messages

Fatal Errors (F)

F 410: Illegal object	Inconsistency found in the object module at unknown offset.
F 413: Only *name* object can be linked	It is not possible to link object for other processors.
F 414: Input file *file* same as output file	Input file and output file cannot be the same.
F 415: Demonstration package limits exceeded	One of the limits in this demo version was exceeded.

Verbose (V)

V 000: Abort !	The program was aborted by the user.
V 001: Extracting files	Verbose message extracting file from library.
V 002: File currently in progress:	Verbose message file currently processed.
V 003: Starting pass *number*	Verbose message, start of given pass.
V 004: Rescanning....	Verbose message rescanning library.
V 005: Removing file *file*	Verbose message cleaning up.
V 006: Object file *file* format *format*	Named object file does not have the standard tool chain object format TIOF-695.
V 007: Library *file* format *format*	Named library file does not have the standard tool chain **ar88** format.
V 008: Embedded environment *name* read, relaxed addressing mode check enabled	Embedded environment successfully read.

Startup Command

```
alC88 project_path  file.out  file.inf
```

Error Messages

Illegal Inf File	Advanced locator definition file (.inf) is invalid.
Duplicate Memory -- 0xnnnn ~ 0xnnnn & 0xnnnn ~ 0xnnnn	Memory allocations in 0xnnnn–0xnnnn and 0xnnnn–0xnnnn are duplicated.
No physical memory available for xxxx	No specified addresses exist to which symbol xxxx can be assigned.
Duplicate Symbol Name -- xxxx	There are duplicates of symbol name xxxx.
Cannot find 0xnnnn bytes for xxxx section	No 0xnnnn bytes of memory are available as needed to map section xxxx.
Found unresolved external -- xxxx	No information is available for external symbol (Extern) xxxx.
There is no stack area	No memory can be allocated for the stack because internal RAM lacks sufficient space.
Absolute address 0xnnnn occupied	The absolute address section area beginning with 0xnnnn is already occupied by another area.

Startup Command

```
lc88 [option]...[file]...
```

Options

-M	Produce a locate map file (`.map`)
-S *space*	Generate specific *space*
-V	Display version header only
-d *file*	Read description file information from *file*, '-' means `stdin`
-e	Clean up if erroneous result
-err	Redirect error messages (`.elc`)
-f *file*	Read command line information from *file*, '-' means `stdin`
-f *format*	Specify output format
-o *filename*	Specify name of output file
-p	Make a proposal for a software part on `stdout`
-v	Verbose option. Print name of each file as it is processed
-w *n*	Suppress messages above warning level *n*

Error Messages

Warnings (W)

W 100:	Maximum buffer size for *name* is *size* (Adjusted)	For the given format, a maximum buffer size is defined.
W 101:	Cannot create map file *filename*, turned off -M option	The given file could not be created.
W 102:	Only one -g switch allowed, ignored -g before *name*	Only one .out file can be debugged.
W 104:	Found a negative length for section *name*, made it positive	Only stack sections can have a negative length.
W 107:	Inserted '*name*' keyword at line *line*	A missing keyword in the description file was inserted.
W 108:	Object name (*name*) differs from filename	Internal name of object file not the same as the filename.
W 110:	Redefinition of system start point	Usually only one load module will access the system table (__lc_pm).
W 111:	Two -o options, output name will be *name*	Second **-o** option, the message gives the effective name.
W 112:	Copy table not referenced, initial data is not copied	If you use a copy statement in the layout part, the initial data is located in rom.
W 113:	No .out files found to locate	No files where specified at the invocation.
W 114:	Cannot find start label *label*	No start point found.
W 116:	Redefinition of name at line *line*	Identifier was defined twice.
W 119:	File *filename* not found in the argument list	All files to be located must be given as an argument.
W 120:	unrecognized name option <*name*> at line *line* (inserted '*name*')	Wrong option assignment. Check the manual for possibilities.
W 121:	Ignored illegal sub-option '*name*' for *name*	An illegal format sub option was detected.
W 122:	Illegal option: *option* (-H or -\? for help)	An illegal option was detected.
W 123:	Inserted *character* at line *line*	The given character was missing in the description file.
W 124:	Attribute *attribute* at line *line* unknown	An unknown attribute was specified in the description file.
W 125:	Copy table not referenced, blank sections are not cleared	Sections with attribute blank are detected, but the copy table is not referenced. The locator generates info for the startup module in the copy table for clearing blank sections at startup.

Error Messages

Warnings (W)

W 127:	Layout *name* not found	The used layout in the named file must be defined in the layout part.
W 130:	Physical block *name* assigned for the second time to a layout	It is not possible to assign a block more than once to a layout block.
W 136:	Removed character at line *line*	The character is not needed here.
W 137:	Cluster *name* declared twice (layout part)	The named cluster is declared twice.
W 138:	Absolute section *name* at non-existing memory address 0x*hexnumber*	Absolute section with an address outside physical memory.
W 139:	*message*	Warning message from the embedded environment.
W 140:	File *filename* not found as a parameter	All processes defined in the locator description file (software part) must be specified on the invocation line.
W 141:	Unknown space <*name*> in -S option	An unknown space name was specified with a -S option.
W 142:	No room for section *name* in read-only memory, trying writable memory ...	A section with attribute read-only could not be placed in read-only memory, the section will be placed in writable memory.

Errors (E)

E 200:	Absolute address 0x*hexnumber* occupied	An absolute address was requested, but the address was already occupied by another section.
E 201:	No physical memory available for section *name*	An absolute address was requested, but there is no physical memory at this address.
E 202:	Section *name* with mau size *size* cannot be located in an addressing mode with mau size *size*	A bit section cannot be located in a byte oriented addressing mode.
E 203:	Illegal object, assignment of non existing var *var*	The MUFOM variable did not exist.
E 204:	Cannot duplicate section '*name*' due to hardware limitations	The process must be located more than once, but the section is mapped to a virtual space without memory management possibilities.
E 205:	Cannot find section for *name*	Found a variable without a section, should not be possible.
E 206:	Size limit for the section group containing section *name* exceeded by 0x*hexnumber* bytes	Small sections do not fit in a page any more.
E 207:	Cannot open *filename*	A given file was not found.

Errors (E)

E 208:	Cannot find a cluster for section *name*	No writable memory available, or unknown addressing mode.
E 210:	Unrecognized keyword <*name*> at line *line*	An unknown keyword was used in the description file.
E 211:	Cannot find 0x*hexnumber* bytes for section *name* (fixed mapping)	One of virtual or physical memory was occupied, or there was no physical memory at all!
E 213:	The physical memory of *name* cannot be addressing in space *name*	A mapping failed. There was no virtual address space left.
E 214:	Cannot map section *name*, virtual memory address occupied	An absolute mapping failed.
E 215:	Available space within *name* exceeded by *number* bytes for section *name*	The available addressing space for an addressing mode has been exceeded.
E 217:	No room for section *name* in cluster *name*	The size of the cluster as defined in the .dsc file is too small.
E 218:	Missing *identifier* at line *line*	This identifier must be specified.
E 219:	Missing ')' at line *line*	Matching bracket missing.
E 220:	Symbol '*symbol*' already defined in <*name*>	A symbol was defined twice.
E 221:	Illegal object, multi assignment on *var*	The MUFOM variable was assigned more than once, probably due to an error of the object producer.
E 223:	No software description found	Each input file must be described in the software description in the .dsc file.
E 224:	Missing <length> keyword in block '*name*' at line *line*	No length definition found in hardware description.
E 225:	Missing <*keyword*> keyword in space '*name*' at line *line*	For the given mapping, the keyword must be specified.
E 227:	Missing <start> keyword in block '*name*' at line *line*	No start definition found in hardware description.
E 230:	Cannot locate section *name*, requested address occupied	An absolute address was requested, but the address was already occupied by another process or section.
E 232:	Found file *filename* not defined in the description file	All files to be located need a definition record in the description file.
E 233:	Environment variable too long in line *line*	Found environment variable in the dsc file contains too many characters.
E 235:	Unknown section size for section *name*	No section size found in this .out file. In fact a corrupted .out file.

Error Messages

Errors (E)

E 236:	Unrecoverable specification at line *line*	An unrecoverable error was made in the description file.
E 238:	Found unresolved external(s):	At locate time all externals should be satisfied.
E 239:	Absolute address *addr.addr* not found	In the given space the absolute address was not found.
E 240:	Virtual memory space *name* not found	In the description files software part for the given file, a non existing memory space was mentioned.
E 241:	Object for different processor characteristics	Bits per MAU, MAU per address or endian for this object differs with the first linked object.
E 242:	*message*	Error generated by the object.
E 244:	Missing *name* part	The given part was not found in the description file, possibly due to a previous error.
E 245:	Illegal *name* value at line *line*	A non valid value was found in the description file.
E 246:	Identifier cannot be a number at line *line*	A non valid identifier was found in the description file.
E 247:	Incomplete type specification, type index = T*hexnumber*	An unknown type was referenced by the given file. Corrupted object file.
E 250:	Address conflict between block *block1* and *block2* (memory part)	Overlapping addresses in the memory part of the description file.
E 251:	Cannot find 0x*hexnumber* bytes for section *section* in block *block*	No room in the physical block in which the section must be located.
E 255:	Section '*name*' defined more than once at line *line*	Sections cannot be declared more than once in one layout/loadmod part.
E 258:	Cannot allocate reserved space for process *number*	The memory for a reserved piece of space was occupied.
E 261:	User assert: *message*	User-programmed assertion failed.
E 262:	Label '*name*' defined more than once in the software part	Labels defined in the description file must be unique.
E 264:	*message*	Error from the embedded environment.
E 265:	Unknown section address for absolute section *name*	No section address found in this .out file. In fact a corrupted .out file.
E 266:	%s %s not (yet) supported	The requested functionality is not (yet) supported in this release.

Fatal Errors (F)

F 400:	Cannot create file *filename*	The given file could not be created.
F 401:	Cannot open *filename*	A given file was not found.
F 402:	Illegal object: Unknown command at offset *offset*	An unknown command was detected in the object file. Corrupted object file.
F 403:	Illegal filename (*name*) detected	A filename with an illegal extension was detected on the command line.
F 404:	Illegal object: Corrupted hex number at offset *offset*	Wrong byte count in hex number. Corrupted object file.
F 405:	Illegal section index	A section index out of range was detected.
F 406:	Illegal object: Unknown hex value at offset *offset*	An unknown variable was detected in the object file. Corrupted object file.
F 407:	No description file found	The locator must have a description file with the description of the hardware and the software of your system.
F 408:	*message*	No protection key or not an IBM compatible PC.
F 410:	Only one description file allowed	The locator accepts only one description file.
F 411:	Out of memory	An attempt to allocate more memory failed.
F 412:	Illegal object, offset *offset*	Inconsistency found in the object module.
F 413:	Illegal object	Inconsistency found in the object module at unknown offset.
F 415:	Only *name* .out files can be located	It is not possible to locate object for other processors.
F 416:	Unrecoverable error at line *line*, *name*	An unrecoverable error was made in the description file in the given part.
F 417:	Overlaying not yet done	Overlaying is not yet done for this .out file, link it first without -r flag!
F 418:	No layout found, or layout not consistent	If there are syntax errors in the layout, it may occur that the layout is not usable for the locator.
F 419:	*message*	Fatal from the embedded environment.
F 420:	Demonstration package limits exceeded	One of the limits in this demo version was exceeded.

Error Messages

Verbose (V)

V 000:	File currently in progress:	Verbose message. On the next lines single filenames are printed as they are processed.
V 001:	Output format: *name*	Verbose message for the generated output format.
V 002:	Starting pass *number*	Verbose message, start of given pass.
V 003:	Abort !	The program was aborted by the user.
V 004:	Warning level *number*	Verbose message, report the used warning level.
V 005:	Removing file *file*	Verbose message cleaning up.
V 006:	Found file <*filename*> via path *pathname*	The description (include) file was not found in the standard directory.
V 007:	*message*	Verbose message from the embedded environment.

Keyword

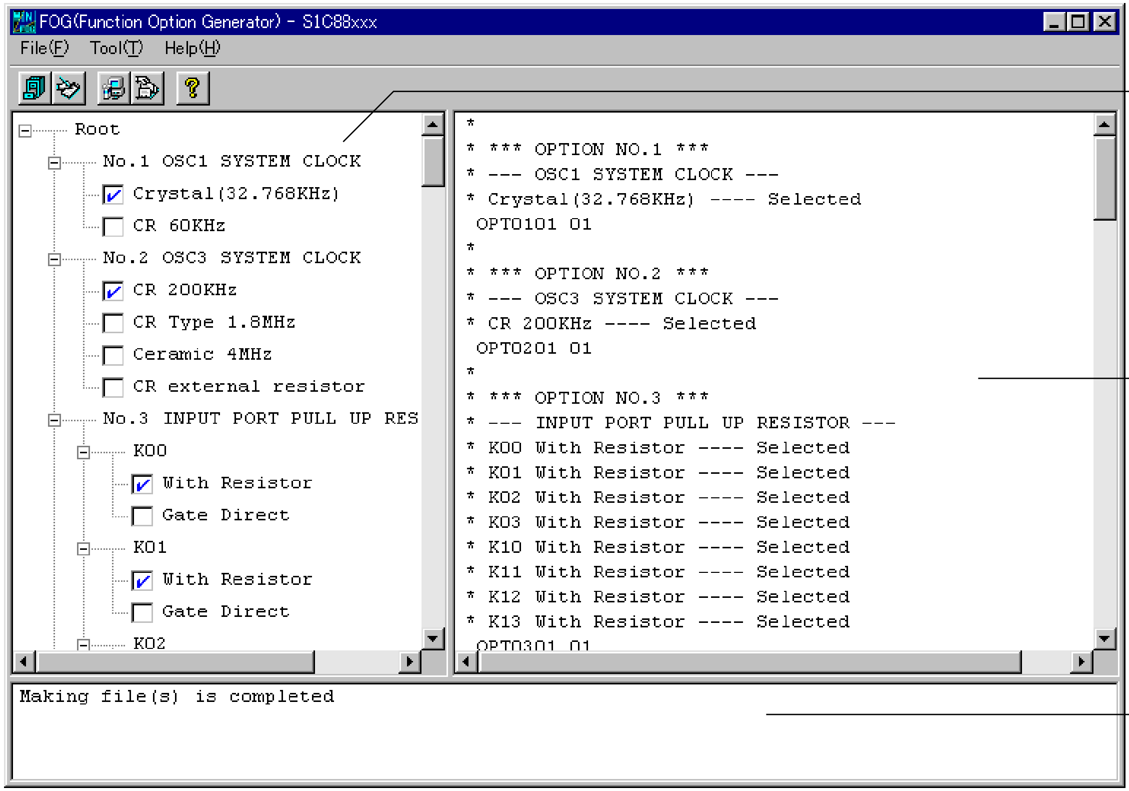
address	Specify absolute memory address
amode	Specify the addressing modes
assert	Error if assertion failed
attribute	Assign attributes to clusters, sections, stack or heap
block	Define physical memory area
bus	Specify address bus
chips	Specify cpu chips
cluster	Specify the order and placement of clusters
copy	Define placement of ROM-copies of data sections
cpu	Define cpu part
dst	Destination address
fixed	Define fixed point in memory map
gap	Reserve dynamic memory gap
heap	Define heap
label	Define virtual address label
layout	Start of the layout description
length	Length of stack, heap, physical block or reserved space
load_mod	Define load module (process)
map	Map a source address on a destination address
mau	Define minimum addressable unit (in bits)
mem	Define physical start address of a chip
memory	Define memory part
regsfr	Specify register file for use by debugger
reserved	Reserve memory
section	Define how a section must be located
selection	Specify attributes for grouping sections into clusters
size	Size of address space or memory
software	Define the software part
space	Define an addressing space or specify memory blocks
src	Source address
stack	Define a stack section
start	Give an alternative start label
table	Define a table section

Function Option Generator winfog (1)

Outline

The function option generator winfog is the software tool for creating the file necessary to generate mask patterns of several hardware specifications such as I/O port functions. In addition, simultaneously with this file, winfog can create a mask option setup file that are required when debugging programs with the ICE.

Windows



```
Root
  No.1 OSC1 SYSTEM CLOCK
    ☑ Crystal(32.768KHz)
    □ CR 60KHz
  No.2 OSC3 SYSTEM CLOCK
    ☑ CR 200KHz
    □ CR Type 1.8MHz
    □ Ceramic 4MHz
    □ CR external resistor
  No.3 INPUT PORT PULL UP RES
    K00
      ☑ With Resistor
      □ Gate Direct
    K01
      ☑ With Resistor
      □ Gate Direct
    K02
```

```
*
* *** OPTION NO.1 ***
* --- OSC1 SYSTEM CLOCK ---
* Crystal(32.768KHz) ---- Selected
 OPT0101 01
*
* *** OPTION NO.2 ***
* --- OSC3 SYSTEM CLOCK ---
* CR 200KHz ---- Selected
 OPT0201 01
*
* *** OPTION NO.3 ***
* --- INPUT PORT PULL UP RESISTOR ---
* K00 With Resistor ---- Selected
* K01 With Resistor ---- Selected
* K02 With Resistor ---- Selected
* K03 With Resistor ---- Selected
* K10 With Resistor ---- Selected
* K11 With Resistor ---- Selected
* K12 With Resistor ---- Selected
* K13 With Resistor ---- Selected
 OPT0301 01
```

```
Making file(s) is completed
```

Option list area

Lists mask options set in the device information definition file (s1c88xxx.ini). Use the check boxes in this area to select each option. A selected option has its check box marked by ✓.

Function option document area

Displays the contents of selected options in the function option document format. The contents displayed in this area are output to the function option document file. When you change any selected item in the option list area, the display in this area is immediately updated.

Message area

When you create a file by selecting [Generate] from the [Tool] menu or clicking the [Generate] button, this area displays a message showing the result of the selected operation.

Buttons

Tool bar

[Open] button
Opens a function option document file.

[Generate] button
Creates a file according to the selected contents of the option list.

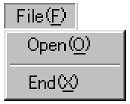
[Setup] button
Sets the date of creation, output file name and a comment included in the function option document file.

[Device INI Select] button
Loads the device information definition file (s1c88xxx.ini).

[Help] button
Displays the version of winfog.

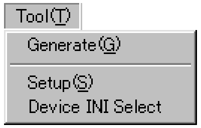
Menus

[File] menu



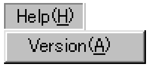
Open
Opens a function option document file.
End
Terminates winfog.

[Tool] menu



Generate
Creates a file according to the selected contents of the option list.
Setup
Sets the date of creation, output file name and a comment included in the function option document file.
Device INI Select
Loads the device information definition file (s1c88xxx.ini).

[Help] menu



Version
Displays the version of winfog.

Error Messages

File name error	Number of characters in the file name or extension exceeds the limit.
Illegal character	Prohibited characters have been entered.
Please input file name	File name has not been entered.
Can't open File : xxxx	File (xxxx) cannot be opened.
INI file is not found	Specified device information definition file (.ini) does not exist.
INI file does not include FOG information	Specified device information definition file (.ini) does not contain function option information.
Function Option document file is not found	Specified function option document file does not exist.
Function Option document file does not match INI file	Contents of the specified function option document file do not match device information definition file (.ini).
A lot of parameter	Too many command line parameters are specified.
Making file(s) is completed [xxxx is no data exist]	Finished creating the file, but the created file (xxxx) does not contain any data.
Can't open File: xxxx Making file(s) is not completed	File (xxxx) cannot be opened when executing Generate.
Can't write File: xxxx Making file(s) is not completed	File (xxxx) cannot be written when executing Generate.

Warning Message

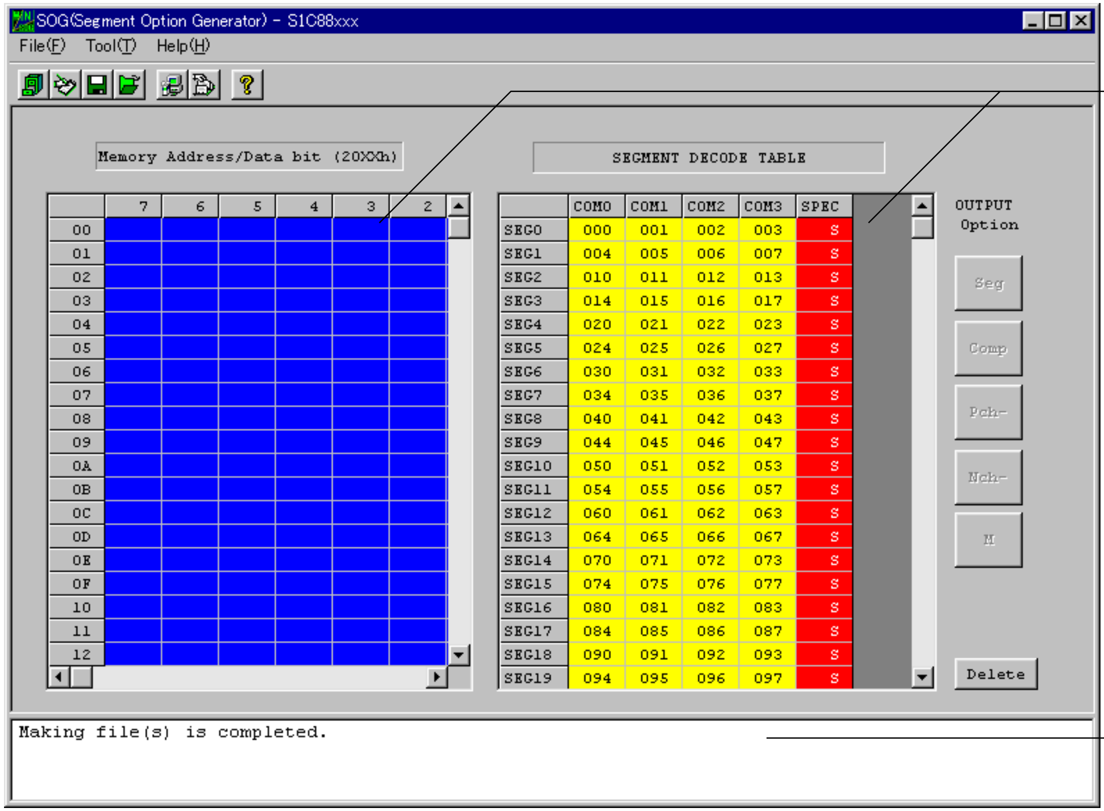
Are you file update? xxxx is already exist	Overwrite confirmation message (Specified file already exists.)

Segment Option Generator winsog (1)

Outline

The segment option generator winsog is the software tool for creating the file necessary to generate mask patterns of LCD output specifications and LCD output pin assignments. In addition, simultaneously with this file, winsog can create a mask option setup file that are required when debugging programs with the ICE.

Windows



	7	6	5	4	3	2
00						
01						
02						
03						
04						
05						
06						
07						
08						
09						
0A						
0B						
0C						
0D						
0E						
0F						
10						
11						
12						

	COM0	COM1	COM2	COM3	SPEC
SEG0	000	001	002	003	S
SEG1	004	005	006	007	S
SEG2	010	011	012	013	S
SEG3	014	015	016	017	S
SEG4	020	021	022	023	S
SEG5	024	025	026	027	S
SEG6	030	031	032	033	S
SEG7	034	035	036	037	S
SEG8	040	041	042	043	S
SEG9	044	045	046	047	S
SEG10	050	051	052	053	S
SEG11	054	055	056	057	S
SEG12	060	061	062	063	S
SEG13	064	065	066	067	S
SEG14	070	071	072	073	S
SEG15	074	075	076	077	S
SEG16	080	081	082	083	S
SEG17	084	085	086	087	S
SEG18	090	091	092	093	S
SEG19	094	095	096	097	S

```
Making file(s) is completed.
```

Option setup area

Comprised of a display memory map, a segment decode table, and buttons to select pin specifications. By clicking on cells in the display memory map and segment decode table, you can assign display memory addresses and bits.

Seg	Selects LCD segment output.
Comp	Selects DC-complementary output.
Pch-	Selects DC-Pch open-drain output.
Nch-	Selects DC-Nch open-drain output.
M	Selects segment/common shared output.
Delete	Clears selected segment assignments.

Message area

When you create a file by selecting [Generate] from the [Tool] menu or clicking the [Generate] button, this area displays a message showing the result of the selected operation.

Buttons

Tool bar

[Open] button
Opens a segment option document file.

[Save] button
Saves the current option settings to a file (segment assignment data file).

[Load] button
Loads a segment assignment data file.

[Generate] button
Creates a file according to the contents of segment options set.

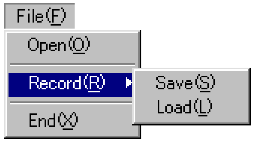
[Setup] button
Sets the date of creation or output file name or a comment included in the segment option document file.

[Device INI Select] button
Loads the device information definition file (s1c88xxx.ini).

[Help] button
Displays the version of winsog.

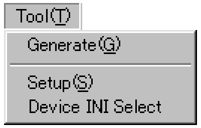
Menus

[File] menu



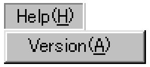
Open
Opens a segment option document file.
Record - Save
Saves the current option settings to a file (segment assignment data file).
Record - Load
Loads a segment assignment data file.
End
Terminates winsog.

[Tool] menu



Generate
Creates a file according to the contents of segment options set.
Setup
Sets the date of creation or output file name or a comment included in the segment option document file.
Device INI Select
Loads the device information definition file (s1c88xxx.ini).

[Help] menu



Version
Displays the version of winsog.

Error Messages

File name error	Number of characters in the file name or extension exceeds the limit.
Illegal character	Prohibited characters have been entered.
Please input file name	File name has not been entered.
Can't open File : xxxx	File (xxxx) cannot be opened.
INI file is not found	Specified device information definition file (.ini) does not exist.
INI file does not include SOG information	Specified device information definition file (.ini) does not contain segment option information.
Function Option document file is not found	Specified function option document file does not exist.
Function Option document file does not match INI file	Contents of the specified function option document file do not match device information definition file (.ini).
Segment Option document file is not found	Specified segment option document file does not exist.
Segment Option document file does not match INI file	Contents of the specified segment option document file do not match device information definition file (.ini).
Segment assignment data file is not found	Specified segment assignment data file does not exist.
Segment assignment data file does not match INI file	Contents of the specified segment assignment data file do not match device information definition file (.ini).
Can't open File: xxxx Making file(s) is not completed	File (xxxx) cannot be opened when executing Generate.
Can't write File: xxxx Making file(s) is not completed	File (xxxx) cannot be written when executing Generate.
ERROR: SPEC is not set Making file(s) is not completed	One or more SPEC cells are left blank when executing Generate.

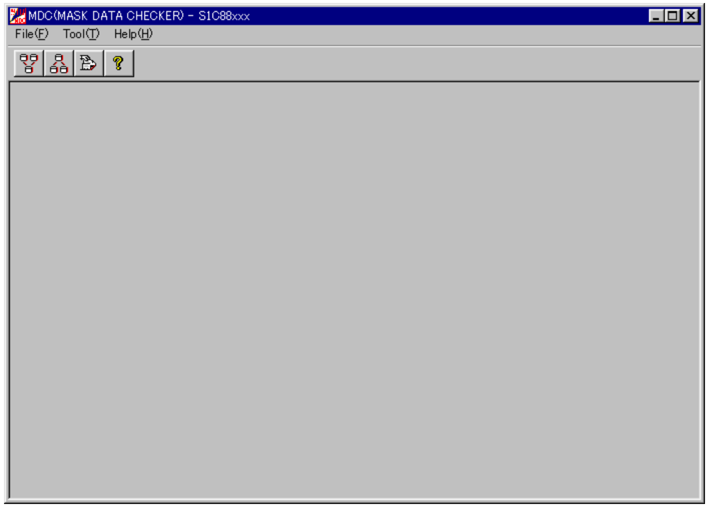
Warning Message

Are you file update? xxxx is already exist	Overwrite confirmation message (Specified file already exists.)

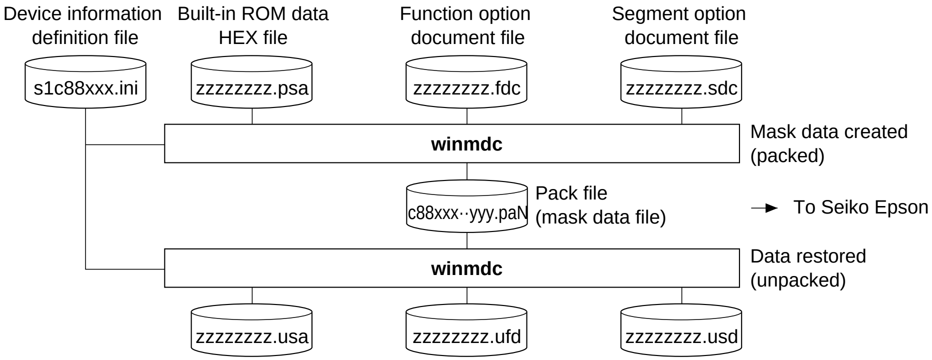
Mask Data Checker winmdc (1)

Outline

The Mask Data Checker winmdc checks the format of the internal ROM HEX files generated by the program unused area filling utility fil88xxx and the option document files generated by the function option generator winfog and segment option generator winsog, and create a file necessary to generate mask patterns. winmdc also has a function for restoring the created mask data file into the original file format.



Flowchart


Device information definition file
Built-in ROM data HEX file
Function option document file
Segment option document file
s1c88xxx.ini
zzzzzzzz.psa
zzzzzzzz.fdc
zzzzzzzz.sdc
winmdc
Mask data created (packed)
c88xxx··yyy.paN
Pack file (mask data file)
To Seiko Epson
winmdc
Data restored (unpacked)
zzzzzzzz.usa
zzzzzzzz.ufd
zzzzzzzz.usd

Mask Data Checker winmdc (2)

Buttons

Tool bar

[Pack] button
Packs the ROM data file and option document file to create a mask data file for presentation to Seiko Epson.

[Unpack] button
Restores files in the original format from a packed file.

[Device INI Select] button
Loads the device information definition file (s1c88xxx.ini).

[Help] button
Displays the version of winmdc.

Menus

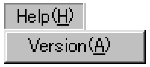
[File] menu

End
Terminates winmdc.

[Tool] menu

Pack
Packs the ROM data file and option document file to create a mask data file for presentation to Seiko Epson.
Unpack
Restores files in the original format from a packed file.
Device INI Select
Loads the device information definition file (s1c88xxx.ini).

[Help] menu



Version
Displays the version of winmdc.

I/O Error Messages

File name error	Number of characters in the file name or extension exceeds the limit.
Illegal character	Prohibited characters have been entered.
Please input file name	File name has not been entered.
INI file is not found	Specified device information definition file (.ini) does not exist.
INI file does not include MDC information	Specified device information definition file (.ini) does not contain MDC information.
Can't open file : xxxx	File (xxxx) cannot be opened.
Can't write file: xxxx	File (xxxx) cannot be written.

ROM Data Error Messages

Hex data error: Not S record.	Data does not begin with "S".
Hex data error: Data is not sequential.	Data is not listed in ascending order.
Hex data error: Illegal data.	Invalid character is included.
Hex data error: Too many data in one line.	Too many data entries exist in one line.
Hex data error: Check sum error.	Checksum does not match.
Hex data error: ROM capacity over.	Data is large. (Greater than ROM size)
Hex data error: Not enough the ROM data.	Data is small. (Smaller than ROM size)
Hex data error: Illegal start mark.	Start mark is incorrect.
Hex data error: Illegal end mark.	End mark is incorrect.
Hex data error: Illegal comment.	Model name shown at the beginning of data is incorrect.

Function Option Data Error Messages

Option data error : Illegal model name.	Model name is incorrect.
Option data error : Illegal version.	Version is incorrect.
Option data error : Illegal option number.	Option No. is incorrect.
Option data error : Illegal select number.	Selected option number is incorrect.
Option data error : Mask data is not enough.	Mask data is insufficient.
Option data error : Illegal start mark.	Start mark is incorrect.
Option data error : Illegal end mark.	End mark is incorrect.

Segment Option Data Error Messages

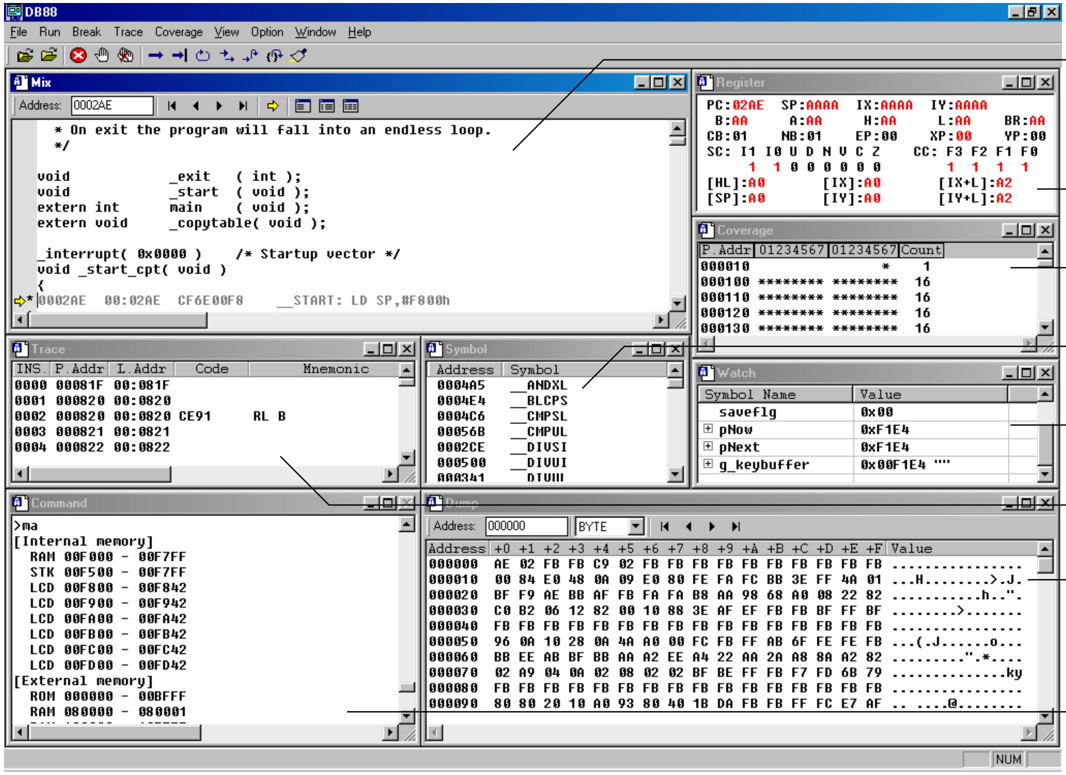
LCD segment data error : Illegal model name.	Model name is incorrect.
LCD segment data error : Illegal version.	Version is incorrect.
LCD segment data error : Illegal segment No.	Segment No. is incorrect.
LCD segment data error : Illegal segment area.	Display memory address is out of range.
LCD segment data error : Illegal segment output specification.	Specified output mode is incorrect.
LCD segment data error : Illegal data in this line.	Data is not hex number or output mode.
LCD segment data error : Data is not enough.	Segment data is insufficient.
LCD segment data error : Illegal start mark.	Start mark is incorrect.
LCD segment data error : Illegal end mark.	End mark is incorrect.

Debugger db88 (1)

Outline

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

Windows



```
 * On exit the program will fall into an endless loop.
 */

void           _exit    ( int );
void           _start   ( void );
extern int     main     ( void );
extern void    _copytable( void );

_interrupt( 0x0000 )    /* Startup vector */
void _start_cpt( void )
{
*0002AE   00:02AE   CF6E00F8     __START: LD SP,#F800h
```

```
PC:02AE  SP:AAAA  IX:AAAA  IY:AAAA
 B:AA     A:AA     H:AA     L:AA    BR:AA
CB:01    NB:01    EP:00    XP:00    YP:00
SC: I1 I0 U D N V C Z    CC: F3 F2 F1 F0
     1  1 0 0 0 0 0 0         1  1  1  1
[HL]:A0        [IX]:A0        [IX+L]:A2
[SP]:A0        [IY]:A0        [IY+L]:A2
```

```
P.Addr 01234567 01234567 Count
000010                *    1
000100 ******** ********  16
000110 ******** ********  16
000120 ******** ********  16
000130 ******** ********  16
```

INS.	P.Addr	L.Addr	Code	Mnemonic
0000	00081F	00:081F		
0001	000820	00:0820		
0002	000820	00:0820	CE91	RL B
0003	000821	00:0821		
0004	000822	00:0822		

Address	Symbol
0004A5	__ANDXL
0004E4	__BLCPS
0004C6	__CMPSL
00056B	__CMPUL
0002CE	__DIVSI
000500	__DIVUI
000341	__DIVUL

Symbol Name	Value
saveflg	0x00
⊞ pNow	0xF1E4
⊞ pNext	0xF1E4
⊞ g_keybuffer	0x00F1E4 ""

```
>ma
[Internal memory]
  RAM 00F000 - 00F7FF
  STK 00F500 - 00F7FF
  LCD 00F800 - 00F842
  LCD 00F900 - 00F942
  LCD 00FA00 - 00FA42
  LCD 00FB00 - 00FB42
  LCD 00FC00 - 00FC42
  LCD 00FD00 - 00FD42
[External memory]
  ROM 000000 - 00BFFF
  RAM 080000 - 080001
```

```
Address +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +A +B +C +D +E +F Value
000000  AE 02 FB FB C9 02 FB FB FB FB FB FB FB FB FB FB ................
000010  00 84 E0 48 0A 09 E0 80 FE FA FC BB 3E FF 4A 01 ...H........>.J.
000020  BF F9 AE BB AF FB FA FA B8 AA 98 68 A0 08 22 82 ...........h..".
000030  C0 B2 06 12 82 00 10 88 3E AF EF FB FB BF FF BF ........>.......
000040  FB FB FB FB FB FB FB FB FB FB FB FB FB FB FB FB ................
000050  96 0A 10 28 0A 4A A0 00 FC FB FF AB 6F FE FE FB ...(.J......o...
000060  BB EE AB BF BB AA A2 EE A4 22 AA 2A A8 8A A2 82 .........".*....
000070  02 A9 04 0A 02 08 02 02 BF BE FF FB F7 FD 6B 79 ..............ky
000080  FB FB FB FB FB FB FB FB FB FB FB FB FB FB FB FB ................
000090  80 80 20 10 A0 93 80 40 1B DA FB FB FF FC E7 AF .. ....@........
```

[Source] window

Displays the program with disassemble codes, source codes or disassemble and source codes.

[Register] window

Displays register values and memory data pointed by the registers.

[Coverage] window

Displays coverage data.

[Symbol] window

Displays symbol information.

[Watch] window

Displays the monitored symbol values.

[Trace] window

Displays traced data.

[Dump] window

Displays the contents of the memory.

[Command] window

Used to enter debug commands and display the execution results.

Buttons

Tool bar buttons

[Load File] button
Loads a program file or a function option file into the debugger.

[Load Parameter] button
Loads a parameter file into the debugger.

[Key Break] button
Forcibly breaks execution of the target program.

[Break] button
Sets or clears a breakpoint at the address where the cursor is located in the [Source] window.

[Break All Clear] button
Clears all break conditions.

[Go] button
Executes the program from the current PC address.

[Go to Cursor] button
Executes the program from the current PC address to the cursor position in the [Source] window.

[Go after Reset] button
Resets the CPU and then executes the program after fetching the reset vector.

[Step] button
Executes one instruction step at the current PC address.

[Next] button
Executes one step at the current PC address. The subroutines are executed as one step.

[Step Exit] button
Executes the program to exit the current subroutine.

[Reset CPU] button
Resets the CPU.

Buttons in the [Source] window

[Disassemble] button
Switches the [Source] window into disassemble display mode.

[Source] button
Switches the [Source] window into source display mode.

[Mix] button
Switches the [Source] window into mix display mode.

[Find] button
Searches the specified strings in the [Source] window.

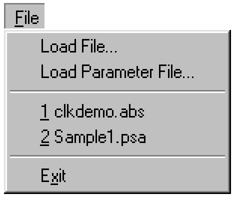
[Find Next] button
Searches the specified strings toward the end of the program.

[Find Previous] button
Searches the specified strings toward the beginning of the program.

[Watch] button
Registers the symbol selected in the [Source] window to the [Watch] window.

Menu

[File] menu

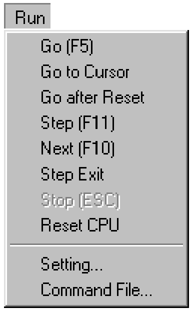


Load File...
Loads a program file or a function option file into the debugger.

Load Parameter File...
Loads a parameter file into the debugger.

Exit
Terminates the debugger.

[Run] menu



Go
Executes the program from the current PC address.

Go to Cursor
Executes the program from the current PC address to the cursor position in the [Source] window.

Go after Reset
Resets the CPU and then executes the program after fetching the reset vector.

Step
Executes one instruction step at the current PC address.

Next
Executes one step at the current PC address. The subroutines are executed as one step.

Step Exit
Executes the program to exit the current subroutine.

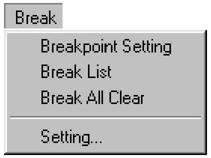
Stop
Forcibly breaks execution of the target program.

Reset CPU
Resets the CPU.

Setting...
Sets options related to program execution.

Command File...
Reads a command file and executes the debug commands written in it.

[Break] menu



Breakpoint Setting
Sets or clears breakpoints and break conditions.

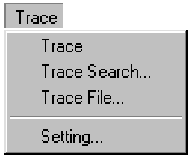
Break List
Displays all the break conditions that have been set.

Break All Clear
Clears all break conditions.

Setting...
Sets break options.

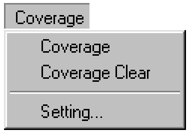
Menu

[Trace] menu



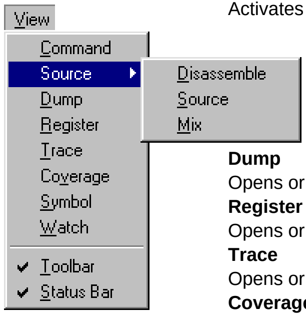
Trace
Displays the trace information.
Trace Search...
Searches trace information from the trace memory.
Trace File...
Saves the specified range of the trace information to a file.
Setting...
Sets a trace mode.

[Coverage] menu



Coverage
Displays the coverage information acquired in the ICE.
Coverage Clear
Clears the coverage information.
Setting...
Selects coverage options.

[View] menu

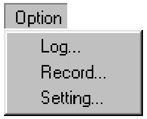


Command
Activates the [Command] window.

Source (Disassemble, Source, Mix)
[Opens or activates the [Source] window and displays the program from the current PC address in the display mode selected from the sub menu items.

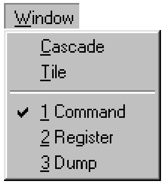
Dump
Opens or activates the [Dump] window and displays the memory contents.
Register
Opens or activates the [Register] window and displays the register values.
Trace
Opens or activates the [Trace] window and displays the trace data.
Coverage
Opens or activates the [Coverage] window and displays the coverage information.
Symbol
Opens or activates the [Symbol] window and displays the symbol information.
Watch
Opens or activates the [Watch] window and displays the symbol value.
Toolbar
Shows or hides the toolbar.
Status Bar
Shows or hides the status bar.

[Option] menu



Log...
Starts or stops logging.
Record...
Starts or stops recording of commands executed.
Setting...
Sets system options.

[Window] menu



Cascade
Cascades the opened windows.
Tile
Tiles the opened windows.

This menu shows the currently opened window names. Selecting one activates the window.

[Help] menu

About DB88...
Displays an About dialog box for the debugger.

Debug Commands

Memory operation

<table>
<tr><td>dd</td><td>[<addr1> [<addr2>] [{-B|-W|-L|-F|-D}]]
[<addr1> <@size>] [{-B|-W|-L|-F|-D}]]</td><td>Dump memory data</td></tr>
<tr><td>de</td><td>[<addr> <data1> [..<data16>]]</td><td>Enter memory data</td></tr>
<tr><td>df</td><td>[<addr1> <addr2> <data>]</td><td>Fill memory area</td></tr>
<tr><td>dm</td><td>[<addr1> <addr2> <addr3>]
[<addr1> <@size> <addr3>]</td><td>Copy memory area</td></tr>
<tr><td>ds</td><td><addr1> {<addr2>|@<byte>}...
...{"<str>"|<data>[:{B|W|L}]} [S=<step>]</td><td>Search memory data</td></tr>
</table>

Register operation

<table>
<tr><td>rd</td><td></td><td>Display register values</td></tr>
<tr><td>rs</td><td>[<reg> <value>]</td><td>Modify register value
reg={PC|SP|IX|IY|A|B|HL|BR|CB|EP|XP|YP|
SC|I1|I0|U|D|N|V|Z|C}</td></tr>
</table>

Program execution

<table>
<tr><td>g</td><td>[<addr>]</td><td>Execute successively from current PC</td></tr>
<tr><td>gr</td><td>[<addr>]</td><td>Reset CPU and execute successively</td></tr>
<tr><td>s</td><td>[<step>]</td><td>Single stepping from current PC</td></tr>
<tr><td>n</td><td>[<step>]</td><td>Single stepping with skip function/subroutine</td></tr>
<tr><td>se</td><td></td><td>Exit from function/subroutine</td></tr>
</table>

CPU reset

<table>
<tr><td>rst</td><td></td><td>Reset CPU</td></tr>
</table>

Break

<table>
<tr><td>bp</td><td>{-|+|_} <addr></td><td>Set software breakpoints</td></tr>
<tr><td>bpa</td><td><addr1> <addr2></td><td>Set software break area</td></tr>
<tr><td>bpr</td><td></td><td>Clear software breakpoints</td></tr>
<tr><td>bc</td><td>[<addr>]</td><td></td></tr>
<tr><td>bpc</td><td>[<addr>]</td><td></td></tr>
<tr><td>bas</td><td>{0|1|2|3}</td><td>Set sequential break mode</td></tr>
<tr><td>ba</td><td><ch> <addr> [<count>]
<ch> {-|+|_}</td><td>Set hardware breakpoints</td></tr>
<tr><td>bar</td><td></td><td>Clear hardware breakpoints</td></tr>
<tr><td>bd</td><td><ch> [A=<addr>][D=<data>][{R|W|}]
<ch> {-|+|_}</td><td>Set hardware data break condition</td></tr>
<tr><td>bdr</td><td></td><td>Clear hardware data break condition</td></tr>
<tr><td>bl</td><td></td><td>Display all break conditions</td></tr>
<tr><td>bac</td><td></td><td>Clear all break conditions</td></tr>
</table>

Program display

<table>
<tr><td>u</td><td>[<addr>]</td><td>Disassemble code display</td></tr>
<tr><td>sc</td><td>[<addr>]</td><td>Source display</td></tr>
<tr><td>m</td><td>[<addr>]</td><td>Mix display</td></tr>
</table>

Symbol information

<table>
<tr><td>sy</td><td>[/a]</td><td>Display symbol list</td></tr>
<tr><td>w</td><td><symbol> [;{H|D|Q|B}] [/A]</td><td>Display symbol information</td></tr>
</table>

Load file

<table>
<tr><td>lf</td><td>[<file>]</td><td>Load program/option HEX file</td></tr>
<tr><td>par</td><td>[<file>]</td><td>Load parameter file</td></tr>
</table>

Trace

<table>
<tr><td>td</td><td>[<cycle>]</td><td>Display trace information</td></tr>
<tr><td>ts</td><td>[{pc|dr|dw} <addr>]</td><td>Search trace information</td></tr>
<tr><td>tf</td><td>[<file> [<cycle1> [<cycle2>]]]</td><td>Save trace information</td></tr>
</table>

Coverage

<table>
<tr><td>cv</td><td>[<addr1> [<addr2>]]</td><td>Display coverage information</td></tr>
<tr><td>cvc</td><td></td><td>Clear coverage information</td></tr>
</table>

Command file, logging

<table>
<tr><td>com</td><td><file> [<interval>]]</td><td>Load and execute command file</td></tr>
<tr><td>cmw</td><td>[<file>]</td><td>Load and execute command file with interval</td></tr>
<tr><td>rec</td><td>[<file>]</td><td>Record executed commands to file</td></tr>
<tr><td>log</td><td>[<file>]</td><td>Logging</td></tr>
</table>

Map information

<table>
<tr><td>ma</td><td></td><td>Display map information</td></tr>
</table>

FPGA operation

<table>
<tr><td>xfer</td><td></td><td>Erase FPGA</td></tr>
<tr><td>xfwr</td><td><file> ;{H|S} [;N]</td><td>Write FPGA data</td></tr>
<tr><td>xfcp</td><td><file> ;{H|S}</td><td>Compare FPGA data</td></tr>
<tr><td>xdp</td><td><addr1> [<addr2>]</td><td>Dump FPGA data</td></tr>
</table>

Quit

<table>
<tr><td>q</td><td></td><td>Quit debugger</td></tr>
</table>

Help

<table>
<tr><td>?</td><td></td><td>Display command usage</td></tr>
</table>

Debugger Messages

Debugger error

Error : Address out of range : use 0x000000 - 0xffffff	The specified address is outside the valid range.
Error : Address out of range, use 0 - 0x7FFFFF	The address specified here is outside the program memory area.
Error : Address out of range, use 0 - 0xFFFFFF	The address specified here is outside the data memory area.
Error : Cannot open device (ICE88UR)	Failed to connect to the ICE.
Error : Cannot open file	Cannot open the file.
Error : Checksum error	Checksum resulted in an error.
Error : Coverage mode is off or the coverage mode is not supported	Coverage mode is turned off or the ICE being used does not support coverage mode.
Error : Data out of range, use 0 - 0xFF	The specified value is outside the valid range of data.
Error : DLL Initialization error	Failed to initialize DLL.
Error : End address < start address	The end address specified here is smaller than the start address.
Error : End index < start index	The end cycle specified here is smaller than the start cycle.
Error : Error file type (extension should be CMD)	The specified file extension is not effective as a command file.
Error : Error file type (extension should be PAR)	The specified file extension is not effective as a parameter file.
Error : Failed ICE88UR initialization	Failed to initialize the ICE.
Error : Failed to initialize DLL : %s	Failed to initialize DLL.
Error : Failed to Load DLL	Failed to load DLL needed to start DB88.
Error : Failed to open : %s	Could not open the file.
Error : Failed to read BA	Error occurred when reading the BA register.
Error : Failed to read BR	Error occurred when reading the BR register.
Error : Failed to read CB	Error occurred when reading the CB register.
Error : Failed to read CC	Error occurred when reading the CC register.
Error : Failed to read EP	Error occurred when reading the EP register.
Error : Failed to read file : %s	Error occurred when reading the file.
Error : Failed to read HL	Error occurred when reading the HL register.
Error : Failed to read NB	Error occurred when reading the NB register.
Error : Failed to read PC	Error occurred when reading the PC register.
Error : Failed to read SC	Error occurred when reading the SC register.
Error : Failed to read SP	Error occurred when reading the SP register.
Error : Failed to read X	Error occurred when reading the X register.
Error : Failed to read Y	Error occurred when reading the Y register.
Error : Failed to road DLL : %s	Failed to load DLL.
Error : Failed to write BA	Error occurred when writing to the BA register.
Error : Failed to write BR	Error occurred when writing to the BR register.
Error : Failed to write CB	Error occurred when writing to the CB register.
Error : Failed to write CC	Error occurred when writing to the CC register.

Debugger error

Error : Failed to write EP	Error occurred when writing to the EP register.
Error : Failed to write HL	Error occurred when writing to the HL register.
Error : Failed to write NB	Error occurred when writing to the NB register.
Error : Failed to write PC	Error occurred when writing to the PC register.
Error : Failed to write SC	Error occurred when writing to the SC register.
Error : Failed to write SP	Error occurred when writing to the SP register.
Error : Failed to write X	Error occurred when writing to the X register.
Error : Failed to write Y	Error occurred when writing to the Y register.
Error : ICE88UR Diagnostic error	Detected an error during ICE self-diagnostic processing.
Error : Ice88ur Initialization failed	Failed to initialize the ICE.
Error : Ice88ur is already running	ICE88UR.EXE is up and running.
Error : ICE88UR is turned off	Power to the ICE is turned off.
Error : Illegal initialization packet data	Initialization packet data is in error.
Error : Incorrect number of parameters	The number of parameters for the command is illegal.
Error : Incorrect r/w option, use r/w/*	The R/W option specified here is invalid.
Error : Incorrect register name, use PC/SP/IX/IY/A/B/HL/BR/CB/EP/XP/YP/SC	The register name specified here is invalid.
Error : Index out of range, use 0 - 8191	The specified trace cycle number is outside the valid range.
Error : Initialization failed! Please quit and restart!	Failed to initialize DB88. Please restart DB88.
Error : Input address does not exist	The address specified here has no breakpoints set.
Error : Invalid command	The command entered here is invalid.
Error : Invalid data pattern	The data pattern entered here is invalid.
Error : Invalid display unit, use -B/-W/-L/-F/-D	The display unit specified here is invalid.
Error : Invalid DLL ModuleID	DLL identification error
Error : Invalid file name	The specified file extension is not effective as a program file or function option file.
Error : Invalid fsa file	The FSA file is invalid.
Error : Invalid hexadecimal string	This is an invalid hexadecimal string.
Error : Invalid value	The value entered here is invalid.
Error : Maximum nesting level(5) is exceeded, cannot open file	Command files have been nested exceeding the nesting limit.
Error : Memory ranges in %s are invalid or the file is not exist	The memory range of the CPU INI file is invalid.
Error : No symbol information	No symbol information is found.
Error : Number of steps out of range, use 0 - 65535	The specified number of steps exceeds the limit.
Error : The Memory Area cannot include the boundary between 0x00FFFF and 0x010000	The specified area overlaps the 0x00FFFF–0x010000 address boundary.

Debugger Messages

Debugger error

Error : The Memory Area must be above 0x10000, and longer than 256 bytes	Any memory area specified above 0x010000 must be greater than 256 bytes in size.
Error : This command is not supported in current mode	The trace and coverage commands are not effective when trace or coverage is turned off.
Error : Unable to get the coverage area number	Failed to get the coverage area number.
Error : Unable to get the coverage mode	Failed to get coverage information.
Error : Unable to set SelfFlash check function	Could not set the SelfFlash check function.
Error : Unable to set the coverage area number	Failed to set the coverage area number.
Error : Unable to set the coverage mode	Failed to set coverage mode.
Error : Wrong Command line parameter	The startup parameters are incorrect.
Please load the selfflash library program	Please load the SelfFlash library program.
Warning : 64 break addresses are already set	The total number of breakpoints specified here exceeds 64.
Warning : Break address already exists	The specified address has a breakpoint already set.
Warning : Identical break address input	Two or more instances of the same address are specified on the command line.
Warning : Memory may be modified by SelfFlash	Memory contents may have been modified by the SelfFlash program.
Warning : SelfFlash program area is out of the current software pc break area. Please clear the break point(Address)	The SelfFlash program area does not match the currently set software break area. Please clear the breakpoint set at (Address).

ICE error

Error : Cannot be run in Free-Run mode	The ICE is operating in free-run mode.
Error : Cannot fine specified data	The specified data could not be found.
Error : ICE88UR is still keep a conservative mode	The ICE is operating in maintenance mode.
Error : ICE88UR power off execution abort	Power to the ICE main unit is off. Execution was aborted.
Error : Insufficient memory for loading program	Failed to allocate memory for the program.
Error : Vdd down or no clock	The power supply voltage for the target system is low, the target system is not powered on, or no clocks are supplied to the target system.
Error : Verify error	A verify error occurred.
ICE88UR system error : ?? illegal packet	Detected an illegal packet.
ICE88UR system error : Command timeout	Detected a command time-out.
ICE88UR system error : Firmware packet error	Detected an error in EB: Firmware packet.
ICE88UR system error : Master reset	Detected MR: master reset.
ICE88UR system error : Not connected	The ICE is not connected or powered on.
ICE88UR system error : Not ready	The ICE is not ready.
Internal error : ICE88UR does not support this command version	The current version of the ICE does not support this command.
Internal error : Illegal error code fetched. System crash possible	Nonexistent error code has been encountered.
Processing terminated by hitting ESC-key	Processing terminated because the ESC key was pressed.

Outline

The structured preprocessor sap88 adds the macro functions to the cross assembler asm88.

The sap88 expands the macro and structured control statements included in the specified S1C88 assembly source file into a format that can be assembled by the asm88, and outputs it. At this time, the sap88 also executes the processing for including of the modularized S1C88 assembly source files and conditional assembly.

Startup Command

```
sap88 [flags] <file name>
```

Flags

-d<macro>	A character-string macro is defined prior to reading in an input file. <macro>: <character-string macro name> = <substitution character string> or <character-string macro name>
-l<label>	The front character string of a label name that is created at the time of the expansion of the structured control statement is designated. It is "L" in default.
-o<file name>	An output file name is turned to *. The default status is standard output.
-q	Does not output any message related to processing of the structured preprocessor.

Error Messages

unexpected EOF in ~	The file is terminated in the middle of ~.
can't include ~	~ cannot be included.
illegal ~	~ is incorrect.
illegal define	"define" statement is incorrect.
illegal expression at ~	~ in the expression is incorrect.
illegal undef	"undef" statement is incorrect.

Pseudo-Instructions

INCLUDE	<file>	Another file insertion
<macro> [<macro>]	MACRO [<param>,...] <statements> [EXITM] <statements> ENDM	Macro definition
DEFINE	<macro> [<character string>]	Character-string macro definition
LOCAL	[<label>,...]	Definition of local label
PURGE	[<macro>]	Macro deletion
UNDEF	<macro>	Deletion of a character string macro
IRP ENDR	<param>,<arg>[,<arg>...] <statements>	Repetition by character strings
IRPC ENDR	<param>,<arg> <statements>	Repetition by characters
REPT ENDR	<expression> <statements>	Repetition by the specified number of times
IFC ELSEC ENDIF	<condition> <statements>[<statements>]	Conditional assembly by conditional expression
IFDEF ELSEC ENDIF	<name> <statements>[<statements>]	Conditional assembly by the name either defined or undefined
IFNDEF ELSEC ENDIF	<name> <statements>[<statements>]	Conditional assembly by the name either undefined or defined

Outline

The cross assembler asm88 converts an assembly source file to machine language by assembling the assembly source file in which the macros are expanded by the structured preprocessor sap88. The asm88 deals with the relocatable assembly for modular development.

In the relocatable assembly, the relocatable object file to link up with the other modules using the linker link88 is created.

Startup Command

```
asm88 [flags] <file names>
```

Flags

-all	Outputs all symbols including local symbols to a symbol table.
-c	Differentiates capital and small letters within the input source.
-l	Prohibits the creation of an assembly list file.
-o<file name>	Creates output files with the name <file name>.
-q	Does not output any messages related to the assembly processing.
-RAM<size>	Sets the RAM capacity in byte units with <size>.
-ROM<size>	Sets the ROM capacity in byte units with <size>.
-sig<number>	Character numbers of symbols that are significant can be set with a <number> value.
-suf<ext>	Changes the extension of the input file to <ext> (a separator "." is not included).
-x	Prohibits the creation of a cross reference list file.

Pseudo-Instructions

CODE			Definition of CODE section
DATA			Definition of DATA section
DB	<exp>[,<exp>...]		Reserve/constant setting of the byte unit data area
DW	<exp>[,<exp>...]		Reserve/constant setting of the word (2-byte) unit data area
DL	<exp>[,<exp>...]		Reserve/constant setting of the long word (4-byte) unit data area
ASCII	<exp>[,<exp>...]		ASCII text storing in memory
PARITY			Setting/resetting of parity bit
<name>	EQU	<exp>	Name value setting
<name>	SET	<exp>	Name value setting
ORG	<exp>		Changing of location counter value
EXTERNAL	<symbol>[,<symbol>]		Symbol external definition declaration
PUBLIC	<symbol>[,<symbol>]		Global declaration of symbol
LINENO	<exp>		Change of line number for assembly list file
SUBTITLE	<title>		Subtitle setting to assembly list file
SKIP			Suppresses all initialization codes output that exceed 4 bytes to assembly list file
NOSKIP			Outputs all initialization codes to assembly list file
LIST			Assembly list file output
NOLIST			Prohibition of assembly list file output
EJECT			Form feed of assembly list file
END	[<label>]		Assembly stop

Error Messages

Fatal errors

can't create <file>	<file> cannot be created.
can't open <file>	<file> cannot be opened.
can't read tmp file	Temporary file cannot be read.
can't write tmp file	Temporary file cannot be written.
namelist full	Name list table is full.
no i/p file	There is no input file specification.
insufficient memory	There is not enough memory.
can't seek on vmem file	Seeking of virtual memory file has failed.
can't seek to end of vmem file	Cannot reach the end of virtual memory file.
no swappable page	There is no swap space.
read error on vmem file	Reading of virtual memory file has failed.
write error on vmem file	Writing to virtual memory file has failed.

Severe errors

<numeric label> already defined	The numeric label has been defined previously.
<identifier> wrong type	An illegal identifier has appeared.
<token> expected	A token is needed.
' missing	A quotation mark is missing.
attempted division by zero	Attempt has been made to divide by zero.
attempt to redefine <identifier>	Attempt has been made to redefine an identifier.
constant expected	A constant expression is required.
end expected	There is no end instruction.
encountered too early end of line	The line has terminated in the middle.
field overflow	The field to be secured has overflowed.
invalid branch address	An external defined symbol is used for the operand of the short branch instruction.
invalid byte relocation	The byte relocation is invalid.
invalid character	Three is an illegal character.
invalid flag	The flag is invalid.
invalid operand	The operand is invalid.
invalid relocation item	The relocation item is invalid.
invalid register	The register is invalid.
invalid register pair	The register combination is invalid.
invalid symbol define	The symbol definition is invalid.
invalid word relocation	The word relocation is invalid.
new origin incompatible with current psect	There is an absolute origin within the relocatable section (relocatable mode).
non terminated string	The termination of a string cannot be located.
<identifier> not defined	Undefined identifier has appeared.
missing numeric expression	A numeric expression is missing.
cars or jrs out of range	Branch destination by cars or jrs is out of range.
carl or jrl out of range	Branch destination by carl or jrl is out of range.

Severe errors

operand expected	There is no operand.
psect name required	A section name must be specified.
phase error <identifier>	The label address is different between pass 1 and pass 2.
CODE or DATA missing	There is no section setting pseudo-instruction.
ROM capacity overflow	ROM capacity has overflowed.
RAM capacity overflow	RAM capacity has overflowed.
relocation error in expression	A relocation error has appeared within the expression.
<identifier> reserved word	<identifier> is a reserved word.
syntax error <token> expected	Syntax error due to insufficient token(s)
syntax error <token> unexpected	Syntax error due to excess token(s)
syntax error - invalid identifier <identifier>	Syntax error due to an illegal identifier
syntax error <token> invalid in expression	Syntax error due to an illegal token
system error < > <token>	System error due to an illegal token
unsupported instruction	Unsupported instruction has appeared.
unsupported operand	Unsupported operand has appeared.

Warnings

directive is ignored in relocatable mode	The pseudo-instruction is skipped because it is in the relocatable mode.
possibly missing relocatability	Relocatability may lose.
constant overflow	Seven or more digits has been defined for the name.
expected operator	There is no operator (BOC, LOC, POD, LOD).

Outline

The link88 links multi-section relocatable object files for the S1C88 and creates an absolute object file. The absolute object file is used to create a program data HEX file that is used for debugging with the ICE by inputting to the binary/HEX converter hex88. It will also be used to create absolute symbol information (rel88) after linking the relocatable assembled file.

Startup Command

```
link88 [global flags] [local flags] [<drive name>:]
```

Flags

Global flags

-c	Distinguishes capital and small letters used for symbols within the relocatable object file.
-cd	Does not output the code part for the DATA section.
+dead	Outputs a list of dead wood symbols on the CRT, that is, symbols that have been defined, but are not referred as absolute.
-max<size>	Sets the maximum section size at <size> bytes.
-o<file name>	Writes the output module on the file <file name>.
-q	Does not output any message related to link processing.

Local flags

+code	Begins a new CODE section, then processes the local flag for that section.
+data	Begins a new DATA section, then processes the local flag for that section.
-m<size>	Sets the maximum size of the individual segment as <size> bytes.
-p<addr>	Sets the physical address of the beginning of the section as <addr>.

Error Messages

bad file format: 'FILE NAME'	Format of the input file 'FILE NAME' is incorrect.
bad relocation item	There is long integer type relocation information.
bad symbol number: 'NUMBER'	'NUMBER' is detected as illegal symbol code.
can't create 'FILE NAME'	The file 'FILE NAME' cannot be created.
can't create tmp file	Temporary file cannot be created.
can't open: 'FILE NAME'	The input file 'FILE NAME' cannot be opened.
can't read binary header: 'FILE NAME'	Header of the file 'FILE NAME' cannot be read.
can't read file header: 'FILE NAME'	First two bytes of the file 'FILE NAME' cannot be read.
can't read symbol table: 'FILE NAME'	Symbol table cannot be read from the file 'FILE NAME'.
can't read tmp file	Temporary file cannot be read.
can't write output file	Cannot write into output file.
can't write tmp file	Cannot write into temporary file.
field overflow	Branch destination by cars or jrs is out of range.
inquiry phase error: 'SYMBOL NAME'	Symbol value of the 'SYMBOL NAME' is different between pass 1 and pass 2.
link: early EOF in pass2	Unexpected EOF is detected during pass 2 processing.
multiply defined 'SYMBOL NAME'	'SYMBOL NAME' is multiply defined.
no object files	No input object files exist.
no relocation bits: 'FILE NAME'	The relocation information corresponding to the file 'FILE NAME' is suppressed.
'SECTION NAME' overflow	The section size in the 'SECTION NAME' exceeds the upper limit value.
phase error: 'SYMBOL NAME'	Symbol value of the 'SYMBOL NAME' is different between pass 1 and pass 2.
previous reference blocked: 'SYMBOL NAME' range error	The information related relocation bit width is unmatched.
read error in pass2	Read error is generated during pass 2 processing.
undefined 'SYMBOL NAME'	'SYMBOL NAME' has not been defined.

Outline

The rel88 checks the multi-section relocatable objects. The files that become the object of such checks are relocatable object files output by the cross assembler asm88 and absolute object files output by the link88. The rel88 can be used to check the size and configuration of relocatable object files and to output symbol information in absolute object files output from the link88.

Startup Command

```
rel88 [flags] <file names>
```

Flags

-a	Sorts outputs in alphabetical order of the symbol names.
+dec	Outputs symbol values and segment sizes in decimal numbers.
-d	Outputs all defined symbols within each file, one per line.
-g	Outputs global symbols only.
+in	akes <file names> from standard input and adds them to command line.
+sec	Outputs the physical address and size of each section of multi-segment output files.
-v	Sorts the inside of section by symbol values. The aforementioned -d flag is tacitly specified.

Error Messages

can't read binary header	Reading of the object header excluding magic number and configuration byte has failed.
can't read header	Reading of the first two bytes of the object header (magic number and configuration byte) has failed.
can't read symbol table	Reading of the symbolic table in the object has failed.

Outline

The symbolic table file generator sym88 converts a symbolic information file (file_name.ref) generated in file redirect with the symbol information generating utility rel88 to a symbolic table file (file_name.sy) that can be referenced in the ICE. Loading the symbolic table file and the corresponding relocatable assembly program file in the ICE makes symbolic debugging possible.

Startup Command

```
sym88 <file name>
```

Error Message

No Input File	Input file ".ref" has not been specified.

Outline

The hex88 converts an absolute object file created by the link88 into a hexadecimal data conversion format (program data HEX file). This system adopted Motorola S record format.

Startup Command

```
hex88 [-o<file name>] <file name>
```

Flags

-o<file name>	Writes the output module for the file <file name>.

Error Messages

bad file format	Input file format is incorrect.
can't read <input file>	Reading of the <input file> has failed.
can't write <output file>	Writing to the <output file> has failed.

EPSON

International Sales Operations

AMERICA

EPSON ELECTRONICS AMERICA, INC.

HEADQUARTERS
2580 Orchard Parkway
San Jose, CA 95131, U.S.A.
Phone: +1-800-228-3964 Fax: +1-408-922-0238

SALES OFFICE
Northeast
301 Edgewater Place, Suite 210
Wakefield, MA 01880, U.S.A.
Phone: +1-800-922-7667 Fax: +1-781-246-5443

EUROPE

EPSON EUROPE ELECTRONICS GmbH

HEADQUARTERS
Riesstrasse 15 Muenchen Bayern
80992 GERMANY
Phone: +49-89-14005-0 Fax: +49-89-14005-110

ASIA

EPSON (CHINA) CO., LTD.

7F, Jinbao Bldg., No.89 Jinbao St., Dongcheng District
Beijing 100005, CHINA
Phone: +86-10-6410-6655 Fax: +86-10-6410-7320

SHANGHAI BRANCH
7F, Block B, Hi-Tech Bldg., 900, Yishan Road
Shanghai 200233, CHINA
Phone: +86-21-5423-5522 Fax: +86-21-5423-5512

EPSON HONG KONG LTD.

20/F, Harbour Centre, 25 Harbour Road
Wanchai, Hong Kong
Phone: +852-2585-4600 Fax: +852-2827-4346
Telex: 65542 EPSCO HX

EPSON (CHINA) CO., LTD.

SHENZHEN BRANCH
12/F, Dawning Mansion, Keji South 12th Road
Hi-Tech Park, Shenzhen
Phone: +86-755-2699-3828 Fax: +86-755-2699-3838

EPSON TAIWAN TECHNOLOGY & TRADING LTD.

14F, No. 7, Song Ren Road
Taipei 110
Phone: +886-2-8786-6688 Fax: +886-2-8786-6660

EPSON SINGAPORE PTE., LTD.

1 HarbourFront Place
#03-02 HarbourFront Tower One, Singapore 098633
Phone: +65-6586-5500 Fax: +65-6271-3182

SEIKO EPSON CORPORATION

KOREA OFFICE
50F, KLI 63 Bldg., 60 Yoido-dong
Youngdeungpo-Ku, Seoul, 150-763, KOREA
Phone: +82-2-784-6027 Fax: +82-2-767-3677

GUMI OFFICE
2F, Grand B/D, 457-4 Songjeong-dong
Gumi-City, KOREA
Phone: +82-54-454-6027 Fax: +82-54-454-6093

SEIKO EPSON CORPORATION
SEMICONDUCTOR OPERATIONS DIVISION

IC Sales Dept.
IC International Sales Group
421-8, Hino, Hino-shi, Tokyo 191-8501, JAPAN
Phone: +81-42-587-5814 Fax: +81-42-587-5117

S5U1C88000C Manual II
(Integrated Tool Package for S1C88 Family)
Workbench/Development Tools/Assembler Package Old Version

SEIKO EPSON CORPORATION
SEMICONDUCTOR OPERATIONS DIVISION

■ **EPSON Electronic Devices Website**

http://www.epson.jp/device/semicon_e

Document code: 411391600
First issue October, 2001
Printed March, 2008 in Japan Ⓛ B