OKI

The relocatable assembler for i-Core3

RASi3
User's manual

Program development support software

2005-5 making

* A change history is only the in-company version. It deletes with the shipment version.

Change history

Revision	Date	The contents of correction	Person in charge
The provisional version	04/7/30	New making	Kosaka
First edition	04/9/01	FIX of RASi3 specification	Kawano
The 2nd edition	04/9/03	Correction of explanation of a user symbol. Correction of the explanatory note of a STACKSEG pseudo-instruction.	Kawano
The 3rd edition	04/9/21	Explanation correction of an EQU pseudo-instruction	Kawano
The 4th Edition	05/4/01	Correction by architecture change. 5.9 Restriction of a basic instruction 6.7 Memory initialization pseudo-instruction 6.13 Optimization pseudo-instruction Update 6.14 C debugging pseud-instruction	kawano
The 5th Edition	05/05/26	6.2.2 STACKSEG : correct “_$$SP” to “_$$XSP or _$$YSP” 8.2 Error message list : update the error message	kawano

NOTICE

1. The information contained herein can change without notice owing to product and/or technical improvements. Before using the product, please make sure that the information being referred to is up-to-date.

2. The outline of action and examples for application circuits described herein have been chosen as an explanation for the standard action and performance of the product. When planning to use the product, please ensure that the external conditions are reflected in the actual circuit and assembly designs.

3. When developing and evaluating your product, please use our product below the specified maximum ratings and within the specified operating ranges including, but not limited to, operating voltage, power dissipation, and operating temperature.

4. **Oki assumes no responsibility or liability whatsoever for any failure or unusual or unexpected operation resulting from misuse, neglect, improper installation, repair, alteration or accident, improper handling, or unusual physical or electrical stress including, but not limited to, exposure to parameters beyond the specified maximum ratings or operation outside the specified operating range.**

5. Neither indemnity against nor license of a third party's industrial and intellectual property right, etc. is granted by us in connection with the use of the product and/or the information and drawings contained herein. No responsibility is assumed by us for any infringement of a third party's right which may result from the use thereof.

6. The products listed in this document are intended for use in general electronics equipment for commercial applications (e.g., office automation, communication equipment, measurement equipment, consumer electronics, etc.) and especially only for use in development and evaluation of control programs for equipment and systems. These products are not authorized for other use (as an embedded device and a peripheral device) and for use in any system or application that requires special or enhanced quality and reliability characteristics nor in any system or application where the failure of such system or application may result in the loss or damage of property, or death or injury to humans. Such applications include, but are not limited to, traffic and automotive equipment, safety devices, aerospace equipment, nuclear power control, medical equipment, and life-support systems.

7. Certain products in this document may need government approval before they can be exported to particular countries. The purchaser assumes the responsibility of determining the legality of export of these products and will take appropriate and necessary steps at their own expense for these.

8. No part of the contents contained herein may be reprinted or reproduced without our prior permission.

Table of Contents

1 Chapter Introduction....6

2 Chapter Introduction....8

2.1 THE FLOW OF PROGRAM DEVELOPMENT....8
2.2 DCL FILE....10
2.2.1 Discernment information on a target device....10
2.2.2 The range of the memory space which can be used....10
2.2.3 Access allowed to SFR....10
2.2.4 The reserved word showing an address....10
2.3 FILE SPECIFICATION....11
2.4 ENVIRONMENT VARIABLE....12
2.5 MEMORY SPACE....12
2.6 ADDRESS SPACE....13

3 Chapter Starting of RASi3....14

3.1 THE DEFAULT AT THE TIME OF FILE SPECIFICATION....15
3.2 THE OPERATION METHOD OF RASI3....16
3.3 OPTION SPECIFICATION BY A STARTING OPTION DEFINITION FILE....17
3.3.1 The specification method of a starting option definition file....18
3.3.2 The form of a starting option definition file....18

4 Chapter Option....19

4.1 OPTION LIST....19
4.2 THE FEATURE OF EACH OPTION....22
4.2.1 /PR,/NPR....22
4.2.2 /L,/NL....23
4.2.3 /S,/NS....24
4.2.4 /R,/NR....24
4.2.5 /E,/NE....26
4.2.6 /O,/NO....26
4.2.7 /NC....27
4.2.8 /DEF....28

4.2.9 /V 28
4.2.10 /PL 28
4.2.11 /PW 29
4.2.12 /I 30
4.2.13 /D,/ND 30
4.2.14 /SD,/NSD 31
4.2.15 /T 31
4.2.16 /A 32
4.2.17 /G 32
4.2.18 /BCODE, /BPRAM, /BXRAM, /BXROM, /BYRAM, /BYROM 33

5 Chapter Language specification 35

5.1 COMPOSITION OF A PROGRAM 35
5.1.1 A program and a sentence 35
5.1.2 Constituent factor 35
5.1.3 The kind of sentence 36
5.1.4 The end of a program 37
5.2 A CLASSIFICATION AND ATTRIBUTE OF A VALUE 38
5.2.1 An integer type, a decimal type and an address type 38
5.2.2 You sage type 38
5.3 CONSTANT 39
5.3.1 The number of settings 39
5.3.2 Decimal constant 40
5.3.3 Character constant 40
5.3.4 Character string constant 40
5.4 ESCAPE SEQUENCE 41
5.5 FULL-WIDTH CHARACTER 42
5.6 SYMBOL 42
5.6.1 Distinction of an alphabetic character 42
5.6.2 Reserved word 42
5.6.3 User symbol 43
5.7 EXPRESSION 49
5.7.1 The definition of an expression 49
5.7.2 The feature of a operator 49
5.7.3 A relocatable type and constant expression 53
5.7.4 Simple relocatable type 55
5.7.5 Operation rule 56
5.8 ADDRESSING 68

5.8.1 The form of addressing....68
5.8.2 The range and addressing check of a value....70
5.8.3 The conversion rule of the value at the time of describing a decimal to an immediate operand 76
5.9 RESTRICTION OF A BASIC INSTRUCTION....78
5.9.1 Address register use restrictions....エラー! ブックマークが定義されていません。
5.9.2 Delay instruction use restrictions....78
5.9.3 Repeat instruction use restrictions....78
5.9.4 Access to a stack....エラー! ブックマークが定義されていません。
5.9.5 Use restrictions of a loop instruction....79
5.9.6 Use restrictions of an EXIT instruction....79
5.9.7 Use restrictions of a PCSTK access instructionエラー! ブックマークが定義されていません。

6 Chapter False instruction....80
6.1 ASSEMBLING INITIAL-SETTING FALSE INSTRUCTION....84
6.1.1 TYPE false instruction....84
6.2 SEGMENT DEFINITION FALSE INSTRUCTION....85
6.2.1 SEGMENT false instruction....85
6.2.2 STACKSEG false instruction....86
6.3 SEGMENT CONTROL FALSE INSTRUCTION....87
6.3.1 CODESEG false instruction....87
6.3.2 PRAMSEG false instruction....87
6.3.3 XRAMSEG false instruction....88
6.3.4 XROMSEG false instruction....88
6.3.5 YRAMSEG false instruction....89
6.3.6 YROMSEG false instruction....89
6.3.7 RELSEG false instruction....90
6.4 LINKAGE CONTROL FALSE INSTRUCTION....90
6.4.1 EXTRN false instruction....90
6.4.2 PUBLIC false instruction....91
6.4.3 COMM false instruction....91
6.5 SYMBOL DEFINITION FALSE INSTRUCTION....92
6.5.1 EQU false instruction....92
6.5.2 = False instruction....93
6.5.3 DEFINE false instruction....94
6.6 ADDRESS CONTROL FALSE INSTRUCTION....94
6.6.1 ORG false instruction....94
6.7 MEMORY INITIALIZATION FALSE INSTRUCTION....95

6.7.1 DW false instruction 95
6.8 ASSEMBLING CONTROL FALSE INSTRUCTION 96
6.8.1 INCLUDE false instruction 96
6.8.2 END false instruction 96
6.9 CONDITION ASSEMBLING FALSE INSTRUCTION 97
6.9.1 IF, an IFE false instruction 97
6.9.2 IFDEF, an IFNDEF false instruction 98
6.9.3 IFB, an IFNB false instruction 98
6.10 LISTING CONTROL FALSE INSTRUCTION 98
6.10.1 TITLE false instruction 98
6.10.2 PAGE false instruction 99
6.10.3 PRN, a NOPRN false instruction 99
6.10.4 LIST, a NOLIST false instruction 100
6.10.5 SYM, a NOSYM false instruction 100
6.10.6 REF, a NOREF false instruction 101
6.10.7 ERR, a NOERR false instruction 101
6.10.8 OBJ, a NOOBJ false instruction 102
6.11 MACRO DEFINITION FALSE INSTRUCTION 102
6.11.1 MACRO false instruction 103
6.11.2 EXITM false instruction 103
6.11.3 LOCAL false instruction 105
6.11.4 REPT false instruction 105
6.11.5 IRP false instruction 105
6.12 SCOPE DEFINITION FALSE INSTRUCTION 106
6.12.1 SCOPE false instruction 106
6.13 OPTIMIZATION FALSE INSTRUCTION 106
6.13.1 GJMP, GJMPD, GJSR, a GJSRD false instruction 106
6.14 C DEBUGGING INFORMATION FALSE INSTRUCTION 107
6.14.1 CFILE false instruction 107
6.14.2 CFUNCTION, a CFUNCTIONEND false instruction 107
6.14.3 CARGUMENT false instruction 108
6.14.4 CBLOCK, a CBLOCKEND false instruction 108
6.14.5 CLABEL false instruction 108
6.14.6 CLINE false instruction 108
6.14.7 CGLOBAL false instruction 109
6.14.8 CLOCAL false instruction 109
6.14.9 CSLOCAL false instruction 109
6.14.10 CSTRUCTTAG, a CSTRUCTMEM false instruction 110

6.14.11 CUNIONTAG, a CUNIONMEM false instruction 110
6.14.12 CENUMTAG, a CENUMMEM false instruction 111
6.14.13 CTYPEDEF false instruction 111
6.14.14 CENVINFO false instruction 111
6.14.15 CMAINFO false instruction 112

7 Chapter List file 113

7.1 READING OF AN ASSEMBLY LIST 114
7.2 READING OF A CROSS REFERENCE LIST 115
7.3 READING OF A SYMBOL LIST 116

8 Chapter A message and end code 118

8.1 FORM OF AN ERROR MESSAGE 119
8.2 ERROR MESSAGE LIST 119
8.2.1 Fatal error message 119
8.2.2 Assembling error message 121
8.2.3 Warning message 129
8.3 END CODE 131

1 Introduction

This manual explains the usage of the relocatable assembler RASi3 for OKI original DSP core (i-Core2 and i-Core3).

RASi3 creates an object file, a list file, and an error file from the source file described by the assembly language.

A required system

The following environment is required in order to operate RASi3.

Hardware: IBM PC/AT compatible machine and clone machine
CPU beyond Pentium 90MHz
64MB or more of memory
10MB or more of hard disk

OS: Windows2000 and WindowsXp

RASi3 is the command line type tool which operates by the command prompt of Windows.

Related software

RASi3 has the following relational software. RASI3 and RLi3 are generically called a MACi3 assembler package.

- CCi3 compiler
- RLi3 Linka
- DTi3 debugger simulator

Related documents

The following document other than this manual is attached to RASi3. Please refer to if needed.

- MACi3.TXT
  The latest information which is not in this manual is described by this file.
- DCLi3.TXT
  Explanation of the DCL file which RASi3 uses is described by this file.

RASi3 has a related document in addition to the above-mentioned document. A related document refers to the hardware manual, instruction manual of a target device and the manual of an emulator, etc. The MACi3.TXT file explains the kind of document.

Notation

By this manual, in order to give explanation intelligible, some signs are used. The sign used by this manual and the meaning are as follows.

Sign	Meaning
SAMPLE	This character shows the message displayed on a screen, the example of an input of a command line, the example of the list file created, etc.
CAPITALS	The item expressed with the English capital letter shows that it inputs as a display.
itarics	The item expressed by italic is not display, but it is replaced to required information of an item.
[ ]	The contents of [] are the items inputted if needed. Omitting is also possible.
--	The item in fromt of this sign is repeatable if needed.
{choice1|choice2}	One is chosen and inputted out of a parenthesis ({}). One must input except for the case where [] is specified.
value1-value2	An input value shows the range of value1 to value2.
"Manual"	It shows a name of manual.
"Reference place"	It shows a reference place.
Ctrl+C	It indicates pressing the Ctrl key and the C key concurrently.
PROGRAM . . . PROGRAM	It is shown that a part of program is omitted.

In this manual, when "H" is added in the end of a numerical value, a value means a hexadecimal.

For example, when described as 1000H, 1000 (a decimal number 4096) of a hexadecimal is indicated.

2 Introduction

2.1 The flow of program development

Here, the flow of the work when developing the program created by the assembly language using a MACi3 assembler package is explained. Since this manual does not explain debugging of a program, please refer to the manual of the debugger to be used.

The flow of program development is shown in Fig. 1-1.

Please let the following explanation correspond with the number in a figure, and read it.

(1) A program is described using a general text editor. The file which described the program is called a source file (.ASM file).

(2) A source file is assembled using RASi3 and an object file (.OBJ file) is created. A list file (.LST file) is also created at this time. Moreover, an error message can also be outputted to a file.

(3) An object file (.OBJ file) can be registered into a library file (.LIB file) using the library feature of RLi3. The development performance of a program can be raised by registering a flexible program into a library file. A library file can be used as an input of RLi3. A list file (.LST file) can also be created. This is a file including the list of the object modules and public symbols which are registered into the library.

(4) All the object files that constitute a program using the linkage feature of RLi3 are combined, and one absolute object file (.ABS file) is created. RLi3 solves the external reference between object files, or assigns a logic segment to a memory. Moreover, a map file (.MAP) is also created.

Furthermore, an object file (.ABS file) is changed into a HEX file using the object conversion feature of RLi3. Please refer to the user's manual of RLi3 about the kind of HEX file, and a format.

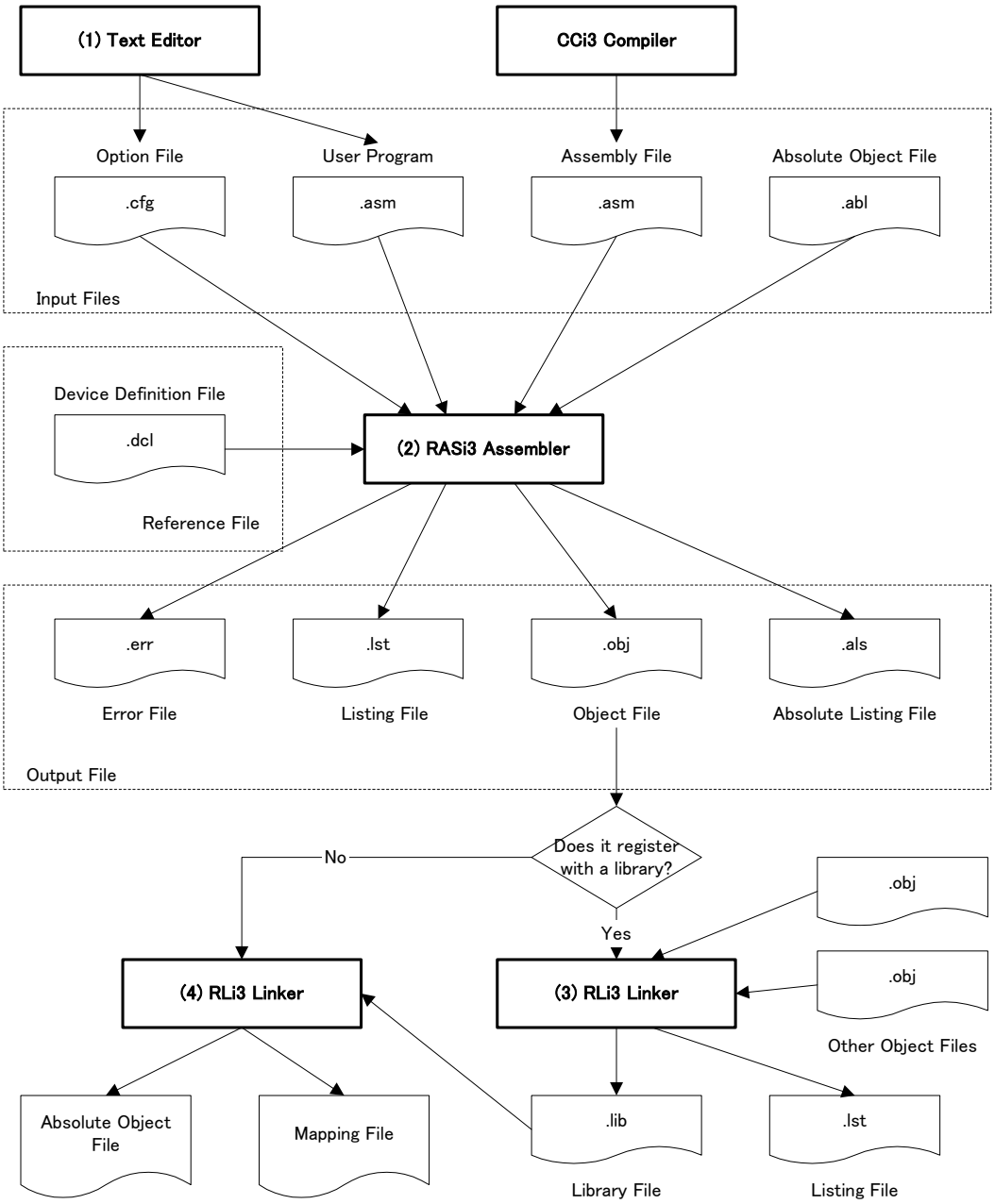


Fig. 1-1 Flow of program development

2.2 DCL file

RASi3 reads the file which defined information peculiar to the device of the object called a DCL file. RASi3 can respond to two or more devices according to this DCL file.

A DCL file is a file of text format. The extension of a DCL file is always ".DCL." The name of the DCL file to refer to is specified in a source program. RASi3 stores the information on a DCL file in an object file. RLi3 receives the information on a DCL file from this object file.

Since the important information for initializing RASi3 is defined as the DCL file, please stop rewriting the contents of the DCL file absolutely. If the contents of the DCL file are rewritten, it may become impossible to process assembling normally.

The contents defined as the DCL file are explained below.

2.2.1 Identification information on a target device

The name of a target device is defined. RLi3 is confirming whether a modular link is possible with reference to the identification information on a target device, when linking.

2.2.2 The range of the memory space which can be used

RASi3 determines the useful range of program memory space and data memory space based on this information. Furthermore, the value of the operand which accesses the target memory is checked.

There is the following kind of the information about memory space.

(1) The address range of program memory (P memory) space and data memory (X memory, Y memory) space

(2) The address range of special areas, such as a SFR area

2.2.3 Access allowed to SFR

RASi3 checks access to a SFR area based on this information.

2.2.4 The reserved word showing an address

RASi3 gets the usage type which the value of the reserved word showing the address, and reserved word from this information. Specifically, the register name currently assigned to the SFR area is defined.

These reserved words can be used instead of an address by an operand.

2.3 File specification

In RASi3, a file is specified as an input or an output. This manual defines such file specification as follows.

<Drive:> <directory> <Base name> <. extension>

Moreover, the combination of a drive and a directory is called a path.

Example

```
C:¥ICORE¥MACI3¥SRC¥SAMPLE.ASM
```

Each part of the file specification in this example is as follows.

Name	Each part of file specification
Path	C:¥ICORE¥MACI3¥SRC¥
Drive	C:
Directory	¥ICORE¥MACI3¥SRC¥
Base name	SAMPLE
Extension	.ASM

A maximum of 255 characters can be specified and a LongFilename can also be specified as file designation.

The character which can be used for a file name and an extension is as follows.

A--Z a--z 0--9 _ (underscore) . (dot) ~ (tilde)

The operation is not guaranteed when characters other than the above are used. It depends for characters other than the above on OS which uses an assembler. However, a blank character is not concerned with OS but serves as an error.

Each input-and-output file designation is performed by the starting option or a pseudo-instruction. When specifying a file, depending on the kind of file, only specification or path, and file name of only a path can be specified. In this case, RASi3 applies default specification of file specification.

Moreover, when only . (dot) is specified as an extension, it is judged that RASi3 is a file without an extension.

2.4 Environment variable

RASi3 uses an environment variable DCLI3.

An environment variable DCLI3 is used when RASi3 searchs a DCL file. When there is no DCL file in the directory to which a current directory and RASi3.EXE exist, RASi3 uses an environment variable DCLI3, in order to search a DCL file.

The example of a setting of an environment variable DCLI3 is shown below.

```
SET DCLI3=C:¥ICORE¥DCL
```

2.5 Memory space

Memory space indicates the memory on the target device which RASi3 can manage. RASi3 can use the following memorys.

- Maximum of 256K word program memory (P memory)
- Maximum of 64K word data memory (X memory and Y memory)

- Program memory space

  Program memory space is an area for allocation instruction code required for program execution.

  The ROM area of P memory can be used as program memory space.
- Data memory space

  Data memory space is the space which holds initial data required for program execution, and the data under execution temporarily. The RAM area of X memory and Y memory can be used as data memory space. Moreover, although a ROM area can be specified as X memory and Y memory, it cannot be used as program space.

·

2.6 Address space

An address space is the space set as the object of each memory access addressing. The segment type is assigned to each address space.

The kind of address space and the corresponding segment type are as follows.

Address space	Object memory	Segment type
CODE address	The ROM area of P memory	CODE
PRAM address	The RAM area of P memory	PRAM
XRAM address	The RAM area of X memories	XRAM
XROM address	The ROM area of X memories	XROM
YRAM address	The RAM area of Y memory	YRAM
YROM address	The ROM area of X memories	YROM

3 Starting of RASi3

RASi3 creates an object file, a list file, and an error file from the source file described by the assembly language.

Information required for a rearrangeable object code, and a link and debugging is included in the object file.

The contents of the source file and the generated object code are contained in the list file. Furthermore, the name and value of a symbol which are used by the source file can be shown.

An error file consists of source statements which the error message and the error occurred, and if there is no output specification to a file, it will be displayed on up to a console.

3.1 The default file specification

When using RASi3, the specification of an input file and output file are required. A file is specified by the command line or the operand of a psuedo-instruction. There is the following kind of file specification of RASi3.

1. Specification of Source File
2. Specification of an include file
3. Specification of a option file
4. Specification of an object file
5. Specification of a list file
6. Specification of an error file
7. Specification of a DCL file

In the above-mentioned file specification, a drive and a directory are omissible. Except specification of a source file, an include file, and a option file, a base name is also omissible. The default when omitting a drive, a directory, a base name, or an extension is as follows.

Kind of file	Default path specification	Default file name	Extension
Source file	Current path [1]	An abbreviation is impossible.	.ASM
Option definition file	Current path	An abbreviation is impossible.	An abbreviation is impossible.
Include file	Search path [2]	An abbreviation is impossible.	An abbreviation is impossible.
DCL file	Search path	An abbreviation is impossible.	.DCL
Object file	The path of a source file	The file name of a source file	.OBJ
List file	The path of a source file	The file name of a source file	.LST

[1] A current path points out the working directory which performs RASi3.

[2] Please refer to an include file "an INCLUDE pseudo-instruction" and a DCL file "a TYPE pseudo-instruction" about search path.

Error file	The path of a source file	The file name of a source file	.ERR

3.2 The operation method of RASi3

This section explains how to perform RASi3.

It types with “RASi3” to the command prompt, a source file and an option are specified after that, and the Enter key is pressed. The format of a command line is as follows.

RASi3 [*options*] *source_file* [*options*]

The source file to assemble is specified as *Source_file*. It is used for options combining an option or option file specification. Before the alphabetic character showing an option, you have to attach a slash (/). Please insert a blank character between an option, and source files and options.

If it typed only with RASi3 and the Enter key was pressed, without specifying source_file, after the list of options will be displayed on a console in how to use RASi3, it returns to a command prompt.

Example

When a /S option is attached and it assembles source file MAIN.ASM, it types as follows.

```
RASi3  MAIN.ASM /S
```

When the extension of a source file name is omitted, RASi3 attached and processes extension “.ASM”. When the drive of a source file name is omitted, RASi3 considers that a source file is in the current drive. When the directory of a source file name is omitted, RASi3 considers that a source file is in a current directory. If a command line is inputted correctly, the starting message of RASi3 will be displayed on a screen. Then, the following message is displayed in order.

```
[dcl_file] loading...
 pass1...
 branch optimization...
 pass2...
```

RASi3 loads a DCL file to the beginning of assembling processing. The following message are displayed while loading the DCL file.

```
[dcl_file] loading...
```

dcl_file is the actually loaded DCL file name.

Assembling processing of RASi3 is divided into the processing called a pass 1 and a pass 2. RASi3 determines the value of a symbol, and the address of a program in pass 1 processing. In processing of a pass 2, the result of a pass 1 is used and an object file is created. If processing of a pass 1 starts, “pass 1…” will be displayed, and “pass 2…” will be displayed if processing of a pass 2 starts. Moreover, RASi3 optimizes branch instruction between processing of a pass 1 and a pass 2. “Branch optimization…” will be displayed if this optimization processing starts.

If the created program has an error, an error message will be displayed after that. Please refer to "6.9 Error message" about an error message.

After assembling is completed, RASi3 displays the following messages and returns to a command prompt.

```
List  File : MAIN.prn
Object File : MAIN.obj
Error  File : Console

Errors   : 0
Warnings : 0
Lines    : 100
Assembly End.
```

Three lines of the beginning are the name of each created file. The generated list file name is displayed on List File, the generated object file name is displayed on Object File, and the generated error file name (usually “Console”) is displayed on Error File.

The information on an assembling result is displayed following the display of a file name. The total of an error is displayed after Errors and the total of warning is displayed after Warnings. The number of lines of a source file is displayed after Lines.

Reference

All the messages that RASi3 displays on a screen are outputted to the standard output device. If the redirection feature of DOS is used, a message can be outputted to a file.

3.3 Option specification by a starting option definition file

Instead of describing specification and the option of a source file to a command line, there is also a method of reading an option from a text file. This text file is called an option definition file.

3.3.1 The specification method of a starting option definition file

In order to specify an option definition file, an option definition file is specified after a unit price sign (@). A blank character cannot be inserted between a unit price sign (@) and an option definition file.

Example 1:

When the option required to assemble source file MAIN.ASM is described by option definition file FOO.OPT, it types as follows.

RASi3 MAIN.ASM @FOO.OPT

Example 2:

Although source file specification and an option required in order to assemble MAIN.ASM to BAR.OPT are described, when adding a /S option in addition, it types as follows.

RASi3 @BAR.OPT /S

3.3.2 The form of an option definition file

In an option definition file, description of the following element is possible.

1. Specification of a source file
2. Specification of various options
3. Comment

Each element is divided by a blank (20H), or TAB (09H) and LF (0AH). CR (0DH) is omitted.

The number or the number of characters of an option which can be described to one line do not have restriction.

Moreover, it is possible to describe a comment. If a semicolon(;), a sharp (#), or //appears in a file, henceforth, even LF (0DH) will be interpreted as a comment and will be skipped. A block comment cannot be used.

Example :

The following is an example of the option definition file for assembling MAIN.ASM with a /E, /R, and /NL option.

```
;---------------------------------------
;   Sample of an option definition file (BAR.OPT)
;---------------------------------------
 MAIN.ASM     ; Specification of a source file
 /E           ; The output of an error file is confirmed.
 /R/NL        ; Output item change of a list file
```

4 Command Line Option

By specifying an option, operation of RASi3 and the form of an output file are controllable. All options start in an option head character, and an option name continues. There are some which can specify a parameter after that depending on the kind of option.

An option head character may specify whichever of a slash (/) or Haiphong (-). For convenience, the slash (/) is used in subsequent explanation.

Either a capital letter or a small letter can be used for an option name. A space cannot be inserted between an option head character and an option name and between an option name and a parameter. A pseudo-instruction with the completely same feature exists in some options.

4.1 Option list

The option which RASi3 prepares is shown below.

Option name	Form	Feature	A corresponding pseudo-instruction
PR	/PR[*filename*] *filename* : File specification	A list file is outputted. When filename is omitted, it becomes default file specification.	PRN
NPR	/NPR	A list file is not outputted.	NOPRN
L	/L[*line_switch*] *line_switch* = [0 | 1] 0 : Line Output OFF 1 : Line Output ON	An assembling list is outputted to a list file. line_switch controls the output of the source file line number displayed on an assembling list.	LIST
NL	/NL	An assembling list is not outputted to a list file.	NOLIST
S	/S	A symbol list is outputted to a list file.	SYM
NS	/NS	A symbol list is not outputted to a list file.	NOSYM
R	/R	A cross reference list is outputted to a list file.	REF
NR	/NR	A cross reference list is not outputted	NOREF

		to a list file.	
E	/E[*filename*] *filename* : File specification	An error file is outputted. When filename is omitted, it becomes default file specification.	ERR
NE	/NE	An error file is not outputted.	NOERR
O	/O[*filename*] *filename* : File specification	An object file is outputted. When filename is omitted, it becomes default file specification.	OBJ
NO	/NO	An object file is not outputted.	NOOBJ
NC	/NC	The capital letter and small letter of a user symbol are not distinguished.	Nothing
DEF	/DEF*symbol*[="*body*"] *symbol* : symbol to define. *body* : The value and character string by which a symbol is expanded.	A symbol is defined. A symbol is expanded by body when [="*body*"] is specified. When "="body"" is omitted, it is regarded as ["1"].	DEFINE
V	/V	The version of RASi3 is displayed. If this option is specified, only a version display will be performed and assembling will not be performed.	Nothing
PL	/PL[*num*] *num* : The number of lines of 1 page If num is omitted, it will be regarded as 60 lines.	The number of lines of 1 page of a list file is set as the value specified as num. A default is unrestricted.	PAGE
PW	/PW[*num*] *num* : The number of characters of one line If num is omitted, it will be regarded as 79 characters.	The number of characters of one line of a list file is set as the value specified as num. A default is unrestricted.	PAGE
I	/I*path* *path* : Include path	The search path of an include file is specified.	Nothing
D	/D	Assembly source level debugging information is outputted to an object file.	Nothing
ND	/ND	Assembly source level debugging information is not outputted to an object file.	Nothing

SD	/SD	C source level debugging information is outputted to an object file.	Nothing
NSD	/NSD	C source level debugging information is not outputted to an object file.	Nothing
T	/T*target_devuce* *target_device* : Target device name	A target device name is specified. Refer to the DCL file of specified target device name.dcl for RASi3.	TYPE
A	/A*file_name* *file_name* : .abl file name	An absolute list file (.als) is generated with reference to the absolute information file specified by file_name.	
G	/G*file_name* *file_name* : .abl file name	A relocatable object file (.obj) is generated with reference to the absolute information file specified by file_name.	
B	/B*mem*(*addr1,addr2*) *mem* : Memory space = CODE | PRAM | XRAM | XROM | YRAM | YROM *addr* : Range to extend	A memory is extended. The range specified by addr is added to the address space specified by mem.	Nothing
W	/W*num* *num* : warning number	Restrict output of warning message specified to *num.*	Nothing

4.2 The feature of each option

In an option definition file, description of the following element is possible.

4.2.1 /PR, /NPR

- Syntax

/PR[*list_file*]

/NPR

- Description

Use of a /PR option creates a list file. A list file name is specified as list_file. Please refer to "3.1 The default file specification" about the default in the case of omitting a part of file specification, when omitting an operand.

Use of a /NPR option does not create a list file. However, when the /A option is specified concurrently, a list file will be created even if there is specification of /NPR.

When omitting a /PR option and a /NPR option, a list file is created and a list file name becomes what changed the extension of a source file name into ".LST."

- A corresponding pseudo-instruction

Instead of specifying a /PR option, you may describe a PRN pseudo-instruction in a program. Moreover, you may describe a NOPRN pseudo-instruction in a program instead of specifying a /NPR option. Priority is given to specification of an option when specifying both an option and a pseudo-instruction.

Please refer to "a PRN/NOPRN pseudo-instruction" about a PRN pseudo-instruction and a NOPRN pseudo-instruction.

- Example

RASi3 FOO.ASM /PROUTPUT.LST

A list file OUTPUT.LST is created in this example.

RASi3 FOO.ASM /NPR

A list file is not created in this example.

- Supplement

A /PR option and a /NPR option cannot be specified concurrently.

4.2.2 /L, /NL

- Syntax

/L

/NL

- Description

When a /L option is specified, each statement until a NOLIST pseudo-instruction appears in a program is outputted to an assembly list.

When a /NL option is specified, each statement until a LIST pseudo-instruction appears in a program is not outputted to an assembly list. However, a statement including an error will be outputted to an assembly list, even if the /NL option is specified.

A /NL option is specified by a default.

An assembly list is outputted to a list file. The contents outputted are explained by the "list file." Please refer to "a LIST/NOLIST pseudo-instruction" about a LIST pseudo-instruction and a NOLIST pseudo-instruction.

- A corresponding pseudo-instruction

The feature of these options, a LIST pseudo-instruction, and a NOLIST pseudo-instruction is almost the same. However, a LIST pseudo-instruction and a NOLIST pseudo-instruction have effect to the statement after the described line to specification of an option being available from the head of a program.

- Example

When outputting the contents of source file FOO.ASM to an assembly list and assembling them, it types as follows.

RASi3 FOO.ASM /L

When assembling without outputting the contents of source file FOO.ASM to an assembly list, it types as follows.

RASi3 FOO.ASM /NL

- Supplement

/L and /NL cannot be specified concurrently.

4.2.3 /S, /NS

- Syntax

/S

/NS

- Description

When a /S option is specified, the information on all user symbols is outputted to a symbol list. When a /NS option is specified, a symbol list is not created.

A symbol list is not created by a default.

A symbol list is outputted to a list file. The contents outputted are explained to the "list file."

- A corresponding pseudo-instruction

Instead of specifying a /S option, you may describe a SYM pseudo-instruction in a program. Moreover, you may describe a NOSYM pseudo-instruction in a program instead of specifying a /NS option. Priority is given to specification of an option when both an option and a pseudo-instruction are specified. Please refer to "a SYM/NOSYM pseudo-instruction" about a SYM pseudo-instruction and a NOSYM pseudo-instruction.

- Example

When outputting all the symbols currently used for source file FOO.ASM to a symbol list and assembling them, it types as follows.

```
RASi3 FOO.ASM /S
```

When assembling the contents of source file FOO.ASM, it types as follows, without creating a symbol list.

```
RASi3 FOO.ASM /NS
```

- Supplement

/S and /NS cannot be specified concurrently.

4.2.4 /R, /NR

- Syntax

/R

/NR

- Description

When a /R option is specified, the appearance line number in all user symbol is outputted to a cross reference list. When a /NR option is specified, a cross reference list is not created.

Correctly, the cross reference list created is influenced of the REF pseudo-instruction and NOREF pseudo-instruction described in a program. Even if the /R option is specified, when a NOREF pseudo-instruction is described in a program, a line number until a REF pseudo-instruction appears is not outputted to a cross reference list. On the other hand, even if the /NR option is specified, when a REF pseudo-instruction is described in a program, a line number until a NOREF pseudo-instruction appears is

outputted to a cross reference list. Therefore, the REF pseudo-instruction and the NOREF pseudo-instruction have the role of the switch of the output of a cross reference list.

However, generally there is almost no above usage. Therefore, you may think that a /R option creates a cross reference list and a /NR option does not create a cross reference list.

A cross reference list is not created by default.

A cross reference list is outputted to a list file. The contents outputted are explained to the "list file." Please refer to "a REF/NOREF pseudo-instruction" about a REF pseudo-instruction and a NOREF pseudo-instruction.

- A corresponding pseudo-instruction

The feature of these options, a REF pseudo-instruction, and a NOREF pseudo-instruction is almost the same. However, although specification of an option becomes available from the head of a program, REF and NOREF are available to the statement after the line which described the pseudo-instruction.

- Example

When creating and assembling the cross reference list of symbols currently used for source file FOO.ASM, it types as follows.

```
RASi3 FOO.ASM /R
```

When assembling source file FOO.ASM without making of a cross reference list, it types as follows.

```
RASi3 FOO.ASM /NR
```

- Supplement

/R and a /NR option cannot be specified concurrently.

4.2.5 /E, /NE

- Syntax

/E[*error_file*]

/NE

- Description

A /E option directs the output place of an error message to RASi3. If an error file name is specified as error_file, an error message will be outputted to the file. When an operand is omitted, please refer to "the default of file specification" about the default in the case of omitting a part of error file.

A /NE option directs to display an error message on a screen (standard output) to RASi3.

By a default, an error message is displayed on a screen.

Only an assembling error message and a warning message can control the output place of an error message by a /E option. When you also output a fatal error message and an internal processing error message to a file collectively, please use the redirection feature of DOS.

- A corresponding pseudo-instruction

Instead of specifying a /E option, you may describe an ERR pseudo-instruction in a program. Moreover, you may describe a NOERR pseudo-instruction in a program instead of specifying a /NE option. Priority is given to specification of an option when specifying both an option and a pseudo-instruction.

Please refer to "an ERR/NOERR pseudo-instruction" about an ERR pseudo-instruction and a NOERR pseudo-instruction.

- Example

RASi3 FOO.ASM /EERROR.LST

Making of an error file ERROR.LST is specified in this example.

- Supplement

/E and a /NE option cannot be specified concurrently.

4.2.6 /O, /NO

- Syntax

/O[*object_file*]

/NO

- Description

An object file is created when a /O option is used. An object file name is specified as object_file. Please refer to “the default of file designation” about the default when omitting a part of file designation, when an operand is omitted.

Use of a /NO option does not create an object file.

When a /O option and a /NO option are omitted, a print file is created and an object file name becomes what changed the extension of a source file name into ".OBJ."

- A corresponding pseudo-instruction

Instead of specifying a /O option, you may describe an OBJ pseudo-instruction in a program. Moreover, you may describe a NOOBJ pseudo-instruction in a program instead of specifying a /NO option. Priority is given to specification of an option when specifying both an option and a pseudo-instruction.

Please refer to "an OBJ/NOOBJ pseudo-instruction" about an OBJ pseudo-instruction and a NOOBJ pseudo-instruction.

- Example

```
RASi3 FOO.ASM /OOUTPUT.OBJ
```

In this example, it is pointing to making of an object file OUTPUT.OBJ.

```
RASi3 FOO.ASM /NO
```

In this example, it specifies not creating an object file.

- Supplement

/O and a /NO option cannot be specified concurrently.

4.2.7 /NC

- Syntax

```
/NC
```

- Description

When a /NC option is specified, the capital letter and small letter of an alphabetic character which are used for the symbol are no longer distinguished. In this case, if spelling of a symbol is the same, even if proper use of a capital letter and a small letter differs, it will be managed as the same symbol. If a /NC option is specified, RASi3 will be managed after changing into a capital letter all the alphabetic characters currently used for the symbol. It is stored in the symbol information on a list file or an object file by the name changed into the capital letter.

When a /NC option is omitted, the capital letter and small letter of an alphabetic character are distinguished.

Only the user symbol defined in programs, such as a label and a segment name, and the SFR symbol defined in a DCL file distinguish the capital letter and small letter of an alphabetic character, and is controlled.

Reserved word, such as an instruction and a pseudo-instruction, is not concerned with option specification, and cannot distinguish the capital letter and small letter of an alphabetic character.

- Example

When assembling source file FOO.ASM without distinguishing a capital letter and a small letter, it types as follows.

```
RASi3 FOO.ASM /NC
```

4.2.8 /DEF

- Syntax

/DEF*symbol*[=*body*]

- Description

A /DEF option defines a macro symbol. A blank character cannot be inserted between symbol and (=), and between (=) and body.

When “=body” is omitted, body is assigned to the macro symbol “symbol”. “1” is assigned to the macro body when “=body” is omitted.

- A corresponding pseudo-instruction

Instead of specifying this option, it is also possible to define a macro symbol using a DEFINE pseudo-instruction.

- Example

When assembling source file FOO.ASM, defining a macro body "TYPE (MXXXXXX)" as the macro symbol READDCL and defining "1" as the macro symbol ONE, it types as follows.

RASi3 FOO.ASM /DEFREADDCL=TYPE(MXXXXXX) /DEFONE

4.2.9 /V

- Syntax

/V

- Description

A /V option displays the version information of RASi3.

Assembling is not performed when a /V option is specified.

- A corresponding pseudo-instruction

There is no pseudo-instruction corresponding to a /V option.

4.2.10 /PL

- Syntax

/PL[*page_length*]

- Description

A /PL option specifies the number of lines of each page of a list file.

The number of lines of each page is specified as page_length with the number of settings. This number of lines contains the header of a list file, the blank line before and behind that, etc. The value specified as page_length does not have restriction in particular. When page_length is omitted, the number of lines of a page is set as 60.

By the default, the number of lines of each page of a list file is set up without any restriction.

- A corresponding pseudo-instruction

Instead of specifying a /PL option, you may describe a PAGE pseudo-instruction in a

program. The same setup as a /PL option can be performed by specifying the number of lines as the 1st operand of a PAGE pseudo-instruction.

Priority is given to specification of an option when specifying both a PAGE pseudo-instruction and a /PL option.

- Example

RASi3 FOO.ASM /PL100

In this example, the number of lines of each page of a list file is specified as 100 lines.

4.2.11 /PW

- Syntax

/PW[*page_width*]

- Description

A /PW option specifies the number of characters of each line of a list file.

The number of characters of each line is specified as page_width with the number of settings. This number of characters means the number of single byte characters. The value specified as page_width does not have restriction in particular. When a value is omitted, the number of characters of one line is set as 79.

By the default, the number of characters of each line of a list file is set up without any restriction.

- A corresponding pseudo-instruction

Instead of specifying a /PW option, you may describe a PAGE pseudo-instruction in a program. The same setup as a /PW option can be performed by specifying the number of characters as the 2nd operand of a PAGE pseudo-instruction.

Priority is given to specification of an option when specifying both a PAGE pseudo-instruction and a /PW option.

- Example

RASi3 FOO.ASM /PW132

In this example, the number of characters of each line of a list file is specified as 132 characters.

4.2.12 /I

- Syntax

/I*include_path*

- Description

A /I option specifies the path of a file read by INCLUDE pseudo-instruction. Two or more paths can be specified by describing two or more /I options.

RASi3 searches an include file in order of the following.

(1) An include file is searched from a current directory. The file will be read if the target file exists in a current directory.

(2) If the path of an include file is specified as the /I option when the target file does not exist in a current directory, the target file will be searched from the path. A file is searched with the order described when two or more /I options are specified.

Refer to "the INCLUDE pseudo-instruction" for the details about an INCLUDE pseudo-instruction.

- Example

When assembling source file FOO.ASM and searching an include file in order of a current directory, C:¥USR¥SHARE¥INC, and C:¥USR¥PRV¥INC, it types as follows.

RASi3 FOO.ASM /IC:¥USR¥SHARE¥INC /IC:¥USR¥PRV¥INC

4.2.13 /D, /ND

- Syntax

/D

/ND

- Description

A /D option outputs assembly level debugging information to an object file. If this debugging information is included in the object file, a program can be debugged symbolically.

A /ND option does not output assembly level debugging information to an object file.

By a default, debugging information is not outputted to an object file.

- A corresponding pseudo-instruction

There is no pseudo-instruction corresponding to /D and /ND.

- Example

RASi3 FOO.ASM /D

In this example, assembly level debugging information is outputted to the object file.

- Supplement

/D and a /ND option cannot be specified concurrently.

4.2.14 /SD, /NSD

- Syntax

/SD

/NSD

- Description

A /SD option is specified when a source file is created by CCi3 compiler. If this option is specified, RASi3 will analyze C debugging pseudo-instruction embedded at the assembly source file which CCi3 compiler created, and will create an object file including C source-level debugging information. By specifying this option, it is available of C source-level debugging.

When a /NSD option is specified, C source-level debugging information is not outputted to an object file. In this case, C source-level debugging cannot be performed.

By a default, C source-level debugging information is not outputted to an object file.

- A corresponding pseudo-instruction

There is no pseudo-instruction corresponding to /SD and a /NSD option.

- Example

RASi3 CCFOO /SD

In this example, source file CCFOO.ASM which CCi3 compiler created is assembled, and an object file including C source level debugging information is created.

4.2.15 /T

- Syntax

/T*target_device*

- Description

A /T option specifies a target device name.

The name of a target device is specified as target_device.

- A corresponding pseudo-instruction

Instead of specifying a /T option, a target device can also be specified using a TYPE pseudo-instruction. Priority is given to specification of an option when both an option and a TYPE pseudo-instruction are specified. Please refer to "a TYPE pseudo-instruction" about a TYPE pseudo-instruction.

- Example

When assembling source file FOO.ASM by setting a target device to MXXXXX, it types as follows.

RASi3 FOO.ASM /TMXXXXX

4.2.16 /A

- Syntax

/A[*abl_file*]

- Description

A /A option creates an absolute list file.

An absolute list file is a list file which does not have relocatable information without including indefinite instruction code information and address information at all.

An ABL file name is specified as abl_file. An ABL file is a binary format file with information required in order to create an absolute list file, and it is created by RLi3 Linker.

Also when using a /A option, the specification method of the file name by a /PR option does not change. However, although the default extension of the usual list file is “.LST”, the default extension of an absolute list file is “.ALS”.

Refer to the "absolute listing feature" for the details of the making method of an absolute print file.

- Example

When creating the absolute list file of source file FOO.ASM, it types as follows. ABL file APRINFO.ABL is read in this example.

RASi3 FOO.ASM /AAPRINFO

4.2.17 /G

- Syntax

/G*file_name*

- Description

A /G option generates a relocatable object file with reference to the ABL file specified by file_name.

- Example

When creating a relocatable object file with reference to the absolute list file of source file FOO.ASM, it types as follows. ABL file APRINFO.ABL is read in this example.

RASi3 FOO.ASM /GAPRINFO

4.2.18 /BCODE, /BPRAM, /BXRAM, /BXROM, /BYRAM, /BYROM

- Syntax

/BCODE(*start_address,end_address*)

/BPRAM(*start_address,end_address*)

/BXRAM(*start_address,end_address*)

/BXROM(*start_address,end_address*)

/BYRAM(*start_address,end_address*)

/BYROM(*start_address,end_address*)

- Description

These options are options for specifying the kind and area of the memory, when a user adds a memory on an address space manageable by RASi3. The kind of memory carried in the target device cannot be redefined. The start address and end address of an area are specified as start_address and end_address, respectively.

A /BCODE option adds a CODE area (ROM area on P address space). The ranges of the address which can be specified are the arbitrary areas of 00000H to 3FFFFH.

A /BPRAM option adds the RAM area on P address space. The ranges of the address which can be specified are the arbitrary areas of 00000H to 3FFFFH.

A /BXRAM option adds the RAM area on X address space. The ranges of the address which can be specified are the arbitrary areas of 0000H to 0FFFFH.

A /BXROM option adds the ROM area on X address space. The ranges of the address which can be specified are the arbitrary areas of 0000H to 0FFFFH.

A /BYRAM option adds the RAM area on Y address space. The ranges of the address which can be specified are the arbitrary areas of 0000H to 0FFFFH.

A /BYROM option adds the ROM area on Y address space. The ranges of the address which can be specified are the arbitrary areas of 0000H to 0FFFFH.

4.2.19 /W

- Syntax

/W*warn_num*

- Description

A /W option restrict the output of warning message specified by *warn_num*.

- Example

When the following warining message in FOO.ASM is restricted, warning_number 60 is specify to *warn_num*.

Warning 60:A page address may be changed within a loop

RASi3 FOO.ASM /W60

5 Language specification

In this chapter, the specification of the source file of RASi3 assembler is explained.

5.1 Composition of a program

Here, the component of a program is explained.

5.1.1 A program and a sentence

A program is the aggregate of one or more sentences. A sentence is a set of the character finished as a line feed code (0AH), and EOT (1AH) or EOF (end of a physical file). A return code (0DH) is skipped.

The number of the sentences of one program is restricted to a maximum of 9,999,999 lines per 1 assembling. Moreover, although the length of a sentence does not have restriction, the character outputted to an assembling list is from the head of a sentence to 256 characters.

5.1.2 Component

The sentence in a program consists of elements shown below.

Separator, Special sign, Operator sign, Constant, Symbol, Comment, Block comment

Explanation of each element is shown below.

- Separator

  Separators are one or more blanks (20H) or TAB (09H) for dividing the element which adjoins in a sentence.

- Special sign

  A special sign is the character which gives a special meaning to the element of order by the existence. The special sign which RASi3 prepares for below is shown. Each part explains the meaning of each sign.

: ; , . [ ] ? :: // /* */ ++

- Operator sign

  A operator sign is a sign of one character or two characters used as a operator. The

operator sign which RASi3 prepares for below is shown.

+	-	*	/	%	&&	||	!	&	|	^	~		
<<	>>	>	>=	<	<=	==	!=	( )					

- Constant

  A constant is description which gives a certain fixed value. A constant is classified into the number of settings, a character constant, and a character string constant.

- Symbol

  A symbol is the character string of one or more characters which consists of an alphabetic character, a number, an _(underscore), ?, and $. However, a number cannot be used for the 1st character for the purpose of distinction with the number of settings. A symbol is classified into a reserved word and a user symbol. A reserved word is a symbol which RASi3 prepares beforehand, and a user symbol is a symbol which a programmer defines in a program.

- Comment

  A comment is description from “//” or “::” which appeared first in the sentence, to the end (LF). However, “//” and “::” in a character constant and a character-string constant do not have the effect as a comment start. In a comment, it is possible to describe a 2-byte character. A comment does not have influence of what on an assemble result, either.

  Even if “/*” is in a comment, there is no effect of a start of a block comment. The comment described by “::” behind instruction statement in the macro body is deleted at the time of macro expansion.

- Block comment

  A block comment is description which starts in '/*' and finishes with '*/'. The kind of character contained in a block comment does not have restriction. Even if LF is contained in the block comment, RASi3 does not interpret it as the end of a sentence. Moreover, '/*' in a character string constant and '*/' are not judged to be the start of a block comment, and an end. Nesting is possible for a block comment and a nesting level does not have restriction.

5.1.3 The kind of sentence

The kind of sentence has the following three.

Basic instruction statement, Pseudo-instruction statement, Meaningless writing

Below, each statement is explained.

Words/phrases: *LABEL*= label, *MNEM*= mnemonic, *OPR*= operand, *CMNT*= comment

■ Basic instruction statement

Basic instruction statement expresses the machine instruction of CPU symbolically. The sentence showing a basic instruction is called basic instruction statement. There is the following in the description method of basic instruction statement. RASi3 judges a semicolon(;) to be the termination of basic instruction statement.

[*LABEL*:] *MNEM* [*OPR*...] ;
[*LABEL*:] *MNEM* [*OPR*...] *MNEM* [*OPR*...] ;

In RASi3, two instructions are described to one line as mentioned above, and there is the description method which generates one code. Moreover, such description can be divided into two sentences as follows, and can also be described.
[*LABEL*:] *MNEM* [*OPR*...]
MNEM [*OPR*...] ;

Label	A label is an address symbol with the address and the segment attribute of the segment which instruction statement sets. A label must be described to a beginning of a sentence, and must describe a colon (:) following a label symbol.
Mnemonic	A mnemonic is a reserved word showing the kind of basic instruction.
Operand	An operand is the addressing notation which a mnemonic requires. The number and kind of operand were decided according to each basic instruction, and may be unnecessary.

■ Pseudo-instruction statement

RASi3 prepares a pseudo-instruction uniquely. The description method of a pseudo-instruction statement changes with each pseudo-instructions.

■ Meaningless writing

Meaningless writing is a sentence which does not include an instruction. The form of meaningless writing is shown below.

[*LABEL*:] [*CMNT*]

5.1.4 The end of a program

RASi3 considers that the following sentence is the end of a program. RASi3 does not

conduct analysis, even if there is character data after it.

- END pseudo- instruction statement
- The sentence which finishes it as EOT (1AH) or EOF (end of a physical file)

5.2 A classification and attribute of a value

Here, the treatment of the value of an expression, a constant, a symbol, and a reserved word is explained.

5.2.1 An integer type, a decimal type and an address type

A value can be classified into a numerical value type and an address type. A numerical value type has integer type and a decimal type.

An integer type and a decimal type express the mere constant which is not an address. Each internal expression is expressed with 32bit without a sign, and 64bit with a sign.

An address type expresses the address on a certain space. An address type internal expression is expressed with 32bit with a sign.

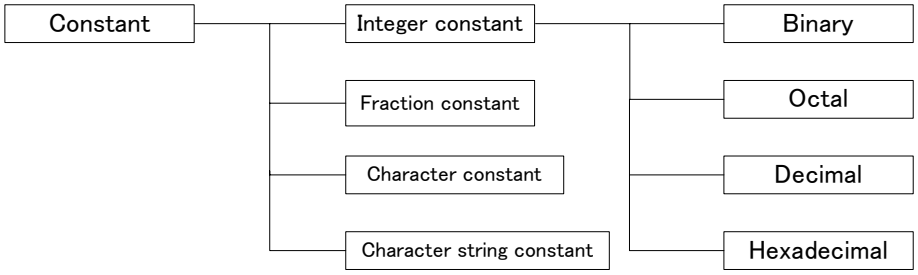
5.2.2 Usage type

A usage type is an attribute expressing the use purpose of a value. The kind of usage type and a meaning are shown below.

Usage type	Meaning
CODE	The address on a CODE address space
PRAM	The address on a PRAM address space
XRAM	The address on a XRAM address space
XROM	The address on a XROM address space
YRAM	The address on a YRAM address space
YROM	The address on a YROM address space
NUMBER	Integer
FLOAT	Decimal fraction

5.3 Constant

The constants which can be managed by RASi3 are an integer constant, a fraction constant, a character constant, and a character-string constant. Furthermore, there are a decimal number, a hexadecimal number, an octal number, and a binary number in an integer constant.



5.3.1 Integer constant

An integer constant is expression showing the value of integer type. The character of the beginning of an integer constant is a number. This is for clarifying distinction with a symbol.

There are four kinds of integer constants, a binary number, an octal number, a decimal number, and a hexadecimal. The last character (radix specifier) determines the kind of integer constant. Moreover, when a radix specifier is omitted, it is regarded as a decimal number. The value of an integer constant is a maximum of 0 FFFF_FFFFH (4,294,967,295). The integer constant expressing the value exceeding this value is an error, and a value is unfixed.

The following table show the kind of an integer constant, the character set which can be described, and radix specifier.

Kind of an integer constant	The character set which can be described	Radix specifier
Hexadecimal number	0 1 2 3 4 5 6 7 8 9 A B C D E F a b c d e f _	H h
Decimal number	0 1 2 3 4 5 6 7 8 9 _	D d
Octal number	0 1 2 3 4 5 6 7 _	O o Q q
Binary number	0 1 _	B b

An underscore (_) may appear in the arbitrary parts after the 2nd character, and does not affect the value of an integer constant.

The usage type of an integer constant is NUMBER.

5.3.2 Fraction constant

A fraction constant is expression showing a decimal fraction type value. The character of the decimal fraction type beginning is a number or decimal point (.). In the case of decimal point, a number must be in the following character.

The range of the value of a fraction constant is 1.7E-308 to 1.7E+308.

The example of format of a fraction constant which can be described is shown below.

```
12.34 , 12. , .34 , 1.2e23 , 1.2e+23 , 1.2e-23
```

The usage type of a fraction constant is FLOAT.

5.3.3 Character constant

A character constant is the character of one or more characters enclosed by the single quotation mark ('). An escape sequence can be used for a character constant. RASi3 interprets a character constant as the numerical value of 1 byte. 2 bytes or more of character and the value beyond 100H are errors, and a value is unfixed.

The usage type of a character constant is NUMBER.

5.3.4 Character string constant

A character string constant is zero or more character the character string of less than 255 characters enclosed by double quotes ("). An escape sequence can be used for a character-string constant. RASi3 interprets a character string constant as the numerical value with which the single byte code was located in a line.

5.4 Escape sequence

An escape sequence is description which starts ¥ (5CH). This is a character constant and a character string constant, and is the feature prepared in order to express the character which cannot be displayed. The following tables are the lists of the escape sequences which can be used by RASi3.

Notation	Value
¥ooo	ooo is an octal number to three characters. A value must be 0 or less FFH.
¥xhh	hh is a hexadecimal number to two characters.
¥Xhh	hh is a hexadecimal number to two characters.
¥a	07H
¥b	08H
¥f	0CH
¥n	0AH
¥r	0DH
¥t	09H
¥v	0BH
¥¥	5CH
¥'	27H
¥"	22H
¥char	“char” indicates ASCII characters other than a, b, f, n, r, t, and v, and is changed into the ASCII code corresponding to a character.
¥j	"j" is a multi-byte character (2-byte code), and although this escape sequence can be used for a character string constant, it cannot use it for a character constant.

5.5 Multi-byte character

In RASi3, use of a Multi-byte character is possible within a comment, a block comment, and a character string constant.

The multi-byte character consists of 2 bytes of codes. The kind of multi-byte character code which can recognize RASi3 is a Shift JIS code.

The code of the multi-byte character which RASi3 recognizes is shown below.

	Shift JIS code
The 1st byte of multi-byte	81H-9FH 0E0H-0FCH
The 2nd byte of multi-byte	40H-7EH 80H-0FCH

5.6 Symbol

A symbol is the character string of one or more characters which consists of an alphabetic character, a number, _ (underscore), ?, and $. However, the 1st character must not be a number. This is for performing distinction on analysis with an integer constant, and a fraction constant. Although the number of characters of a symbol does not have restriction, it ignores after it, without recognizing the number of characters which RASi3 recognizes only to the 32nd character.

Symbols include a reserved word and a user symbol.

5.6.1 Distinction of an alphabetic character

RASi3 does not distinguish the capital letter and small letter of an alphabetic character of a reserved word.

Although RASi3 distinguishes the capital letter and small letter of a user symbol by a default, it does not distinguish when a /NC option is specified.

5.6.2 Reserved word

Since the usage of reserved word was decided beforehand, it cannot be redefined as a user symbol. Moreover, one symbol have two or more kinds of features. The feature is explained according to the kind of reserved word below.

- Basic instruction symbol

  It is a symbol showing the instruction of a target device.
- Pseudo-instruction symbol

  It is a symbol showing the kind of pseudo-instruction.

■ Operator symbol

It is a symbol with the feature of a operator.

LONG SHORT INT SIN COS TAN EXP LOG LOG10 SQRT ABS POW BITREV Q0 Q1 Q2 Q3 Q4 Q5 Q6 Q7 Q8 Q9 Q10 Q11 Q12 Q13 Q14 Q15

■ Register symbol

It is a symbol showing a register or a flag and has a meaning especially by addressing description of a basic instruction.

P X Y YA A0 A1 A0L A1L RA0 RA1 RB0 RB1 GP0 GP1 IA0 IA1 IB0 IB1 CR MR SR ISR GR0 GR1 A0S A1S MD0 MD1 PC PPC PH LC LE PL SCR PCSTK LESTK LCSTK PPCSTK DR0 DR1 DR2 DR3 DR4 DR5 DR6 DR7 DR0G DR1G DR2G DR3G DR4G DR5G DR6G DR7G DR0H DR1H DR2H DR3H DR4H DR5H DR6H DR7H DR0L DR1L DR2L DR3L DR4L DR5L DR6L DR7L

■ Instruction auxiliary symbol

It is a symbol showing the flag state in a basic instruction etc.

LS LC EQ NE CS CC MI PL LE GT LT GE VS VC

■ Pseudo-instruction auxiliary symbol

It is the symbol described as an operand of a specific pseudo-instruction.

■ Current location sign

It is a sign showing the present address of a current segment. It depends for an attribute on the described segment. In this manual, although described as the location symbol, the location counter symbol, etc., it is altogether the same.

$

5.6.3 User symbol

A user symbol is a symbol which a programmer newly defines in a program. A user symbol describes a label or is defined by a /DEF option or the following pseudo-instruction.

SEGMENT, EQU, =, MACRO, SCOPE, DEFINE, COMM, EXTRN

A reserved word cannot be defined as a user symbol.

5.6.3.1 The kind of user symbol

A user symbol is classified into four kinds. The module under following explanation means one file in the program which consisted of two or more source files.

Kind of symbol	Contents
Local symbol	It is the symbol defined by a label or EQU, and = pseudo-instruction. This is a symbol showing the address or a numerical value, is available only within the defined module and cannot be referred to from other modules.
Segment symbol	It is the symbol defined by the SEGMENT pseudo-instruction. A relocatable segment is indicated.
Communal symbol	It is the symbol defined by the COMM pseudo-instruction. A share area with other modules is indicated.
External symbol	It is the symbol defined by the EXTRN pseudo-instruction. This means referring to the symbol declared by the PUBLIC pseudo-instruction, or a communal symbol.
A special user symbol	It is the symbol and temporary symbol which were defined by MACRO, SCOPE, and DEFINE pseudo-instruction.

5.6.3.2 A special user symbol

In a user symbol, the symbol defined by the MACRO pseudo-instruction is called a macro symbol, the symbol defined within SCOPE is called a scope symbol, and the symbol defined by the DEFINE pseudo-instruction is called a DEFINE symbol. Moreover, the symbol % was described to be at the head of the symbol is called a temporary symbol, and has a special feature.

■ Macro symbol

The symbol defined by the MACRO pseudo-instruction is called a macro symbol. The definition method of a macro symbol is as follows.

macro_sym MACRO [*opr1* [, *opr2*] ....]
:
body
:
ENDM

In a macro pseudo-instruction, the end of macro definition is expressed with an ENDM pseudo-instruction, and the instruction statement described between the MACRO pseudo-instruction and the ENDM pseudo-instruction is called the macro body.

In the macro body, it is possible to use a symbol or to call oneself and other macroes. The newest definition becomes available when the same macro symbol is redefined.

■ Scope symbol

The symbol defined within the scope of a SCOPE pseudo-instruction is called a scope symbol. The definition method of a scope symbol is as follows.

scope_tag SCOPE [*global_sym1* [*global_sym2*] .....]
:
scope_sym (symbol definition sentence)
:
ENDC

The SCOPE pseudo-instruction and the ENDC pseudo-instruction indicate beginning and the end of a scope, respectively. scope_tag indicates a scope name and scope_sym indicates a scope symbol name. global_sym 1 and 2 .... indicate the global symbol used within a scope.

When a SCOPE pseudo-instruction with the same scope_tag already exists, they are managed as the same scope. The scope symbol name defined within the scope becomes available only within the same scope.

The method of describing in the case of referring to a scope symbol and a global symbol inside a scope and outside a scope is shown below.

Symbol type	Inside of a scope	Outside of a scope
The symbol defined within the scope (scope_tag1)	scope_sym	scope_tag1.scope_sym
The symbol defined within the different scope (scope_tag2)	scope_tag2.scope_sym	scope_tag2.scope_sym
Global symbol	.global_sym	global_sym or .global_sym
The global symbol enumerated by the SCOPE pseudo-instruction	global_sym	global_sym or .global_sym

・Example

```
global_sym      EQU 10h

scope_tag1      SCOPE  global_sym
scope_sym       EQU    30h
                DW     global_sym    =>    10h (value of the operand of DW)
                DW     scope_sym     =>    30h
                ENDC

scope_tag2      SCOPE
scope_sym       EQU    40h
                DW     .global_sym              =>    10h
                DW     scope_tag1.scope_sym     =>    30h
                ENDC

                DW     global_sym               =>    10h
                DW     scope_tag1.scope_sym     =>    30h
                DW     scope_tag2.scope_sym     =>    40h
```

■ DEFINE symbol

The symbol defined by the DEFINE pseudo-instruction is called a DEFINE symbol. The character string (DEFINE substance) of a maximum of 255 characters is assigned to the DEFINE symbol. When a DEFINE symbol is described in a statement, RASi3 analyzes by displacing a DEFINE symbol to DEFINE substance.

A DEFINE symbol can be referred to only after a definition. Moreover, a DEFINE symbol may be contained in DEFINE substance. In RASi3, in order to prevent the hang-up by refer to the self, nesting of DEFINE is restricted to 8 level.

■ Temporary symbol

% is attached to the temporary symbol at the head of a symbol name.

A temporary symbol can be used only within a code segment. Moreover, a temporary symbol is defined as a label and referred to as an operand of branch instruction. If '<' or '>' is added when referring to a temporary symbol, referring to the back or forward addressing can be chosen, and it is possible to register two or more same symbol names. It means that '<' refers to the temporary symbol of the nearest back, and means that '>' refers to the temporary symbol of the nearest front.

・Example

```
%tmp_sym:          ---------(1)
%tmp_sym:          ---------(2)
    JMP    >%tmp_sym     ---------(3) is referred to.
    JMP    <%tmp_sym     ---------(2) is referred to.
    %tmp_sym:     ---------(3)
  %tmp_sym:       ---------(4)
```

5.6.3.3 The attribute and usage type of a symbol

■ Public attribute

Although a local symbol cannot be referred to from an external module, it can usually be referred to from an external module by making a public declaration. A PUBLIC pseudo-instruction is used for this declaration.

The attribute which can be referred to from an external module is called a public attribute, and a public symbol is called for a symbol with a public attribute.

■ The usage type of a symbol

A usage type is given to a symbol at the time of a definition. The next table shows the usage type of a symbol according to the definition method.

The definition method	Usage type
Label	The usage type of the segment which belongs
EQU, = pseudo-instruction	The usage type of an operand
SEGMENT pseudo-instruction	Segment type specification of an operand
COMM pseudo-instruction	Segment type specification of an operand
EXTRN pseudo-instruction	Usage type specification of an operand
MACRO, SCOPE pseudo-instruction	NUMBER

* Note: The symbol defined by MACRO and SCOPE pseudo-instruction cannot actually be used as an expression, although NUMBER is expressed as the usage type of a symbol list.

5.6.3.4 An absolute symbol and a relocatable symbol

In a user symbol, the symbol which a value fixes during an assemble is called an absolute symbol, and a relocatable symbol is called for the symbol which a value fixes during a link.

An absolute symbol is defined as follows.

- Constant expression is specified and defined as the operand of EQU and = pseudo-instruction.
- A definition is given as a label which belongs to an absolute segment (segment defined by CODESEG, PRAMSEG, XROMSEG, XRAMSEG, YROMSEG, and YRAMSEG pseudo-instruction).

A relocatable symbol is defined as follows.

- It defines by SEGMENT pseudo-instruction. (Segment symbol)
- It defines by COMM pseudo-instruction. (Communal symbol)
- It defines by EXTRN pseudo-instruction. (External symbol)
- A definition is given as a label which belongs to a relocatable segment (segment defined by the RELSEG pseudo-instruction). (Simple relocatable symbol)
- The expression which uses simple relocatable one is specified and defined as the operand of EQU and = pseudo-instruction.

5.7 Expression

Here, each feature of the constituent factor of an expression and a operator and an operation rule are explained.

5.7.1 The definition of an expression

An expression consists of an operation clause and a operator. The operation clause itself has the character of an expression. An operation clause is a basic clause showing a value, and can specify the following element.

Integer constant　Fraction constant　Character constant　User symbol Location counter sign

An expression can be classified into a numerical type and an address type.

・Numerical type

It is an expression expressing a numerical type value. A usage type is surely set to NUMBER or FLOAT.

・Address type

It is an expression expressing an address type value. A usage type is surely except NUMBER and FLOAT.

The usage type of an expression is determined by the kind of the operation clause included in an expression, and performed operation.

5.7.2 The feature of a operator

The feature of all the operators currently prepared for RASi3 is explained.

The following term is used in explanation.

Left term	It is the expression located in the left of a operator.
Right term	It is the expression located in the right of a operator.
True	It means that the value (it is an offset value in the case of an address type) of an expression is except zero.
False	It means that the value (it is an offset value in the case of an address type) of an expression is 0.
TRUE	Numerical value type 1 is meant.
FALSE	Numerical value type 0 is meant.

5.7.2.1 Arithmetic operator

Operator	Form	Feature
+	+ Right term	Positive (unary operator)
	Left term + Right term	Addition
-	- Right term	Negative (unary operator)
	Left term – Right term	Subtraction
*	Left term * Right term	Multiplication
/	Left term / Right term	Division
%	Left term % Right term	Modulo caluculation

5.7.2.2 Logical operator

Operator	Form	Feature
&&	Left term && Right term	TRUE if both expressions are ture; FALSE otherwise.
||	Left term || Right term	TRUE if either expression is true; FALSE otherwise.
!	! Right term	TRUE if the expression is true; FALSE if false.

5.7.2.3 Bit logical operator

Operator	Form	Feature
&	Left term & Right term	Logical AND
|	Left term | Right term	Logical OR
^	Left term ^ Right term	Exclusive OR
~	~ Left term	Bit inversion
>>	Left term >> Right term	Shifts left term to the right by the number of bits given by right term. Zeros are shifted in from the most significant bit.
<<	Left term << Right term	Shifts left term to the left by the number of bits given by right term. Zeros are shifted in from the least significant bit.

5.7.2.4 Relational operator

A relational operation performs numerical size comparison and equivalent comparison. The operation of numerical value types or the address type comrade with

the same usage type is allowed.

Operator	Form	Feature
>	Left term > Right term	Returns TRUE if left term is grater than right term; otherwise returns FALSE.
>=	Left term > = Right term	Returns TRUE if left term is grater than or equal to right term; otherwise returns FALSE.
<	Left term < Right term	Returns TRUE if left term is less than right term; otherwise returns FALSE.
<=	Left term <= Right term	Returns TRUE if left term is less than or equal to right term; otherwise returns FALSE.
==	Left term == Right term	Returns TRUE if left term is equal to right term; otherwise returns FALSE.
!=	Left term != Right term	Returns TRUE if left term is not equal to right term; otherwise returns FALSE.

5.7.2.5 Special operator

A special operator is a special operator which can be used by RASi3.

Operator	Form	Feature
LONG	LONG Right term	Addressing of a basic instruction is managed as long. *
SHORT	SHORT Right term	Addressing of a basic instruction is managed as short. *
SIN	SIN (Right term)	A numerical value type SIN operation result is got.
COS	COS (Right term)	A numerical value type COS operation result is got.
TAN	TAN (Right term)	A numerical value type TAN operation result is got.
EXP	EXP (Right term)	A numerical value type exponential function is got.
LOG	LOG (Right term)	A numerical value type natural logarithm is got.
LOG10	LOG10 (Right term)	Numerical value type common logarithm is got.
SQRT	SQRT (Right term)	A numerical value type square root is got.

ABS	ABS (Right term)	A numerical value type absolute value is got.
INT	INT (Right term)	A numerical value type decimal fraction is got.
POW	POW (Right term1, Right term2)	The number of Right term2 power of Right term1.
BITREV	BITREV (Right term1, Right term2)	The bit of right term1 corresponding to right term2 is reversed.
Qn (n = 0-15)	Qn (Right term)	The number of bits of fraction part is set to n, and the decimal fraction of a decimal number is changed into a hexadecimal. Warning is outputted when the changed value is larger than 65535.

※ Only when it describes as an expression to the operand of the basic instruction from which the size of the machine code generated according to the size of addressing changes, a LONG operator and a SHORT operator have a meaning. It is ignored when described by the other part. When a LONG operator is described in the expression of an operand, the instruction is changed into the machine code of 2-word size, and it is changed into the machine code of 1-word size when a SHORT operator is described.
An example is shown below.

Example

```
    RELSEG      segsym
            // Pseudo-instruction which specifies the start of a  relocatable segment
rel_sym:

    MOV   RA0,      LONG    rel_sym ;
            // It is changed into 2 word machine code.
    MOV   RA0,      SHORT   rel_sym ;
            // It is changed into 1 word machine code.

    JMP   100H+LONG         rel_sym ;
            // It is changed into 2 word machine code.
    JMP   100H+SHORT        rel_sym ;
            // It is changed into 1 word machine code.

    MOVX  X,        [LONG rel_sym] ;
            // It is changed into 2 word machine code.
```

```
MOVX  X,      [SHORT rel_sym] ;
        // It is changed into 1 word machine code.
```

It becomes warning when a SHORT operator is described to the constant value of long size.

Example

```
MOV   RA0,    SHORT 0FF00H ;
```

5.7.3 A relocatable expression and integer constant expression

A relocatable expression is an expression containing a relocatable symbol, or an expression containing $ in a relocatable segment (current location symbol), and is an expression which a value does not fix during an assemble. On the other hand, constant expression is an expression which a value fixes during an assemble.

The instruction which can use a relocatable expression for an operand is as follows.

- Basic instruction
- DW pseudo-instruction (The right term when a DUP expression is used)

When a relocatable expression is used for addressing of a basic instruction, an assembler cannot generate the fixed machine code. In this case, the information for computing a fixed value to an object file is generated, and a linker is solved.

Depending on a operator, a relocatable expression can be specified as an operand. In this case, unsolved operation information is generated by the object file.

The syntax of a relocatable expression is shown below.

RELOP expression		Relocatable type with unsolved operation
REL expression		Relocatable type without unsolved operation
RELOP expression	: : =	(RELOP expression)
	|	+RELOP type
REL expression	::=	REL expression + Constant expression
	|	Constant expression + REL expression
	|	REL expression - Constant expression
	|	(REL expression)
	|	+REL expression
	|	REL term

```
REL term            : : =   Simple relocatable symbol
                    |       Segment symbol
                    |       External symbol
                    |       Communal symbol
                    |       $

Constant expression :: = Unary operator  Constant expression
                    | Constant expression binary operator Constant expression
                    | (constant expression)
                    | Absolute term
*1                  | REL expression - REL expression
```

*1 The type of an expression needs to be the same by the left expression and a right expression. The type of an expression means the kind of symbol (simple relocatable type, segment type, communal type and external type) contained in an expression.

```
Unary operator      ::=     +
                    |       -
                    |       ~
                    |       !
                    |       LONG
                    |       SHORT

Binary operator     : : =   +
                    |       -
                    |       *
                    |       /
                    |       %
                    |       &
                    |       ^
                    |       |
                    |       <<
                    |       >>
                    |       <
                    |       <=
                    |       >
                    |       >=
                    |       ==
                    |       !=
                    |       &&
                    |       | |
```

Absolute term	:: =	Integer constant
	|	Character constant
	|	Absolute symbol
	|	Address symbol
	|	$

[Note]

$ in a REL term means what was described within the relocatable segment.

$ in a Absolute term means what was described within the absolute segment.

5.7.4 Simple relocatable type

A simple relocatable expression is a relocatable expression whose REL term is a simple relocatable symbol.

A simple relocatable expression can be described to the operand of the following instructions.

- The instruction which can use a relocatable expression for an operand
- A part of pseudo-instructions

  EQU

  =

  CODESEG

  PRAMSEG

  XRAMSEG

  XROMSEG

  YRAMSEG

  YROMSEG

5.7.5 Operation rule

Here, the priority of a operator and the result of an expression are explained.

5.7.5.1 The priority of a operator

Each operator has a priority. RASi3 evaluates an expression previously from the higher one of a priority. When a priority is the same, it evaluates according to unity.

The conversion table of a operator, its priority, and unity is shown below. A priority turns into a high rank in this table like what has a small number.

Priority	Operator	Unity
1	( )	From the left to the right
2	SIN, COS, TAN, EXP, LOG, LOG10, SQRT, ABS, INT, POW, BITREV, LONG, SHORT, ! ~ and + (unary) - (unary) Qn	From the right to the left
3	* , / , %	From the left to the right
4	+ (Binary operator) - (Binary operator)	From the left to the right
5	>> , <<	From the left to the right
6	< , <= , > , >=	From the left to the right
7	== , !=	From the left to the right
8	&	From the left to the right
9	^	From the left to the right
10	|	From the left to the right
11	&&	From the left to the right
12	||	From the left to the right

5.7.5.2 Evaluation of an expression

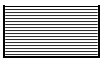
Here, the evaluation rule of the expression of RASi3 is explained according to a operator. Description restrictions of an expression and succession of each attribute are also covered by the rule defined here.

■ The sign to be used

Absnum	A usage type NUMBER, absolute numerical type is meant.
Absaddr	An absolute address type is meant.
Relnum	A relocatable usage type NUMBER numerical type is meant.
Reladdr	A relocatable address type is meant.
Float	A usage type FLOAT numerical type is meant.
Wn	The kind of warning is indicated.

It indicates becoming an error. An operation result is unfixed.

It indicates becoming warning. Operation is performed.



It indicates that evaluation changes with usage types of an expression. (Notes) explain the evaluation rule in this case outside the limit.

■ Succession of an attribute

When the evaluation result of an expression is an address type, the attribute which the original address type has is inherited.

■ The range check of a value

Though the value of an address type is over the range specified in a corresponding address space, in expression analysis processing, it is not managed as an error. The range check of a value is performed into the analysis of addressing.

■ The kind of warning

There are the following kinds of warning to the form of an expression. The warning number used here is for explanation, and differs from a formal error code.

Warning number	The contents of the error
W1	It is the operation which has a meaning only to NUMBER.
W2	It is the operation which has a meaning only to FLOAT.
W3	It is the operation which has a meaning only to a numerical expression.
W4	The right expression is not a numerical expression.
W5	The right expression is not a NUMBER.
W6	Neither the right expression nor left expression is NUMBER.
W7	Operation different usage type.

(1) Arithmetic operator

Expression type / Right expression	+ (Unary)	- (Unary)
Absnum	Absnum	Absnum
Relnum	Relnum	
Absaddr	Absaddr	Absaddr
Reladdr	Reladdr	
float	float	float

Expression type		+	-	*	/	%
Left expression	Right expression					
Absnum	Absnum	Absnum	Absnum	Absnum	Absnum	Absnum
	Relnum	Relnum				
	Absaddr	Absaddr	Absaddr	Absnum	Absanum	Absnum
	Reladdr	Reladdr				
	float	float	float	float	float	
Relnum	Absnum	Relnum	Relnum			
	Relnum		* Notes 2			
	Absaddr	Reladdr	Reladdr			
	Reladdr					
	float					
Absaddr	Absnum	Absaddr	Absaddr	Absnum	Absnum	Absnum
	Relnum	Reladdr				
	Absaddr	Absnum W6	* Notes 1	Absnum W6	Absnum W6	Absnum W6
	Reladdr	Relnum W6				
	float					
Reladdr	Absnum	Reladdr	Reladdr			
	Relnum					
	Absaddr	Relnum W6	* Notes 3			
	Reladdr		* Notes 4			
	float					
float	Absnum	float	float	float	flaot	
	Relnum					
	Absaddr					
	Reladdr					
	float	float	float	float	float	

Notes 1: If it becomes when a usage type is the same, it is set to Absnum, and it will be set to Absnum W7 when different.

Notes 2: When right and left have the same ID, it is set to Absnum, and in the case of others, becomes an error.

Notes 3: If it becomes when a usage type is the same, it is set to Relnum, and it will be set to Relnum W7 when different.

Notes 4: When right and left have the same SEG-ID, EXT-ID, and COM-ID, it is set to Absnum, and in the case of others, it becomes an error.

After the operation of num and float changes num to float, it is calculated.

(2) Logical operator

Expression type		&&	||
Left expression	Right expression		
Absnum	Absnum	Absnum	Absnum
	Relnum		
	Absaddr	Absnum	Absnum
	Rsladdr		
	float	Absnum	Absnum
Relnum	Absnum		
	Relnum		
	Absaddr		
	Rsladdr		
	float		
Absaddr	Absnum	Absnum	Absnum
	Relnum		
	Absaddr	Absnum	Absnum
	Rsladdr		
	float	Absnum	Absnum
Reladdr	Absnum		
	Relnum		
	Absaddr		
	Rsladdr		
	float		
float	Absnum	Absnum	Absnum
	Relnum		
	Absaddr	Absnum	Absnum
	Rsladdr		
	float	Absnum	Absnum

Expression type / Right expression	!
Absnum	Absnum
Relnum	
Absaddr	Absnum
Reladdr	
float	Absnum

(3) Bit logical operator

Expression type		&	|	^	<<	>>
Left expression	Right expression					
Absnum	Absnum	Absnum	Absnum	Absnum	Absnum	Absnum
	Relnum					
	Absaddr	Absnum	Absnum	Absnum	Absnum W5	Absnum W5
	Reladdr					
	float					
Relnum	Absnum					
	Relnum					
	Absaddr					
	Reladdr					
	float					
Absaddr	Absnum	Absnum	Absnum	Absnum	Absnum	Absnum
	Relnum					
	Absaddr	Absnum	Absnum	Absnum	Absnum W5	Absnum W5
	Reladdr					
	float					
Reladdr	Absnum					
	Relnum					
	Absaddr					
	Reladdr					
	float					
float	Absnum					
	Relnum					
	Absaddr					
	Reladdr					
	float					

Expression type / Right expression	~
Absnum	Absnum
Relnum	
Absaddr	Absnum
Reladdr	
float	

(4) Relational operator

Expression type		> >= < <= == !=
Left expression	Right expression	
Absnum	Absnum	Absnum
	Relnum	
	Absaddr	Absnum
	Reladdr	
	float	Absnum
Relnum	Absnum	
	Relnum	
	Absaddr	
	Reladdr	
	float	
Absaddr	Absnum	Absnum
	Relnum	
	Absaddr	* Notes 2
	Reladdr	
	float	Absnum
Reladdr	Absnum	
	Relnum	
	Absaddr	
	Reladdr	
	float	
float	Absnum	Absnum
	Relnum	
	Absaddr	Absnum
	Reladdr	
	float	Absnum

* Notes 2:

If it becomes when a usage type is the same, it is set to Absnum, and it will be set to Absnum W7 when different.

(5) Special operator

Expression type / Right expression	LONG	SHORT	SIN	COS	TAN	EXP
Absnum	Absnum	Absnum	float	float	float	float
Relnum	Relnum	Relnum				
Absaddr	Absaddr	Absaddr	float W4	float W4	float W4	float W4
Reladdr	Reladdr	Reladdr				
float			float	float	float	float

Expression type / Righ expression	LOG	LOG10	SQRT	ABS	INT	*Qn* (n = 0-15)
Absnum	float	float	float	float	Absnum	AbsnumW2
Relnum						
Absaddr	flaot W4	float W4	float W4	float W4	Absnum W4	Absnum W2
Reladdr						
float	float	float	float	float	Absnum	Absnum

Expression type		POW	BITREV
Right expressio n1	Right expression 2		
Absnum	Absnum	float	Absnum
	Relnum		
	Absaddr	float W4	Absnum W5
	Reladdr		
	float	float	
Relnum	Absnum		
	Relnum		
	Absaddr		
	Reladdr		
	float		
Absaddr	Absnum	float W3	Absnum W1
	Relnum		
	Absaddr	float W3	Absnum W5
	Reladdr		
	float	float W3	
Reladdr	Absnum		
	Relnum		
	Absaddr		
	Reladdr		
	float		
float.	Absnum	float	Absnum W1
	Relnum		
	Absaddr	float W4	Absnum W5
	Reladdr		
	float	float	

5.8 Addressing

Here, the range of a value and an addressing check are explained.

5.8.1 The form of addressing

The table of addressing which can be described to the operand of a basic instruction is shown.

Description	Meaning
[xxx_10]	Direct page addressing (10bit short)
[xxx_16]	Direct page addressing (16bit long)
xxx_16	Absolute addressing (16 bits)
xxx_18	Absolute addressing (18 bits)
[ar]	Address register indirectnessing
[ar] ++	Post increment
[ar] --	Post decrement
[ar] + ix	Renewal of an index
[ar] + disp	Renewal of the De Dis placement with a mark disp : 8 bits
imm_2	Immediate (2 bits)
imm_4	Immediate (4 bits)
imm_5	Immediate (5 bits)
imm_6	Immediate (6 bits)
imm_8	Immediate (8 bits)
imm_16	Immediate (16 bits)
[disp_9]	9-bit Displacement
[disp_13]	13-bit Displacement
.cc	Conditional execution
r	Register

The register which can be specified as an operand
RA0,RA1,RB0,RB1,IA0,IA1,IB0,IB1,LC,LE,MR,MD0,GP0,GP1,GP2,GP3,PC,PPC,PCSTK,PPCSTK,LCSTK,LESTK, DR0,DR1,DR2,DR3,DR4,DR5,DR6,DR7, DR0H,DR1H,DR2H,DR3H,DR4H,DR5H,DR6H,DR7H, DR0L,DR1L,DR2L,DR3L,DR4L,DR5L,DR6L,DR7L, DR0G,DR1G,DR2G,DR3G,DR4G,DR5G,DR6G,DR7G

The condition identifier which can be specified as an operand	
Condition identifier	The state of a flag
LS	L=1
LC	L=0
EQ	Z=1
NE	Z=0
CS	C=1
CC	C=0
MI	N=1
PL	N=0
GE	(N^V)=0
LE	((N^V)|Z)=1
GT	((N^V)|Z)=0
LT	(N^V)=1
VS	V=1
VC	V=0

C:Carry Flag N:Negative Flag V:Overflow Flag
L:Latched Overflow Flag Z:Zero Flag

5.8.2 The range of value and addressing check

RASi3 performs the following checks to the operand described by the basic instruction.

- The range check of a value

  The value of the expression described by the operand confirms whether be within the limits of the value of each addressing. An error is outputted when a value is exceeds the range.

- Usage type check

  The usage type of operand confirms whether suit with the usage type which can be described. Warning is outputted when it does not suit.

- Address check

  The value of the expression described by the operand confirms whether be in the memory area specified as the address space. An error is outputted when a value exceeds the range.

Below, the usage type table in which the range of the value for every addressing, an object instruction, and description are possible is shown. Each addressing is described by the position of * in a table.

Moreover, please refer to the specification of I-Core3 about the operand which can be described to r, r1, r2, .cc, A, Ad and ea.

■ xxx_16 , xxx_18

Addressing	The range of a value	Object instruction	Usage type	Address space
xxx_16	0H – 0FFFFH	JMP.cc *	NUMBER	CODE
		JMPD.cc *	CODE	
		JSR.cc *		
		JSRD.cc *		
		DO *	NUMBER	CODE
			CODE	
xxx_18	0H–3FFFFH	JMP.cc *	NUMBER	CODE
		JMPD.cc *	**CODE**	
		JSR.cc *		
		JSRD.cc *		

■ [xxx_16]

Addressing	The range of a value	Object instruction	Usage type	Address space
[xxx_16]	0H - 0FFFFH	MOVX r, *	NUMBER	XRAM/XROM
		SELX r, *	XRAM	XRAM
		TSTX *, imm_4	XROM	XROM
		MOVX *, r	NUMBER	XRAM
		SELX *, r	XRAM	
		SETX *, imm_4		
		CLRX *, imm_4		
		MOVY r, *	NUMBER	YRAM/YROM
		SELY r, *	YRAM	YRAM
		TSTY *, imm_4	YROM	YROM
		MOVY *, r	NUMBER	YRAM
		SELY *, r	YRAM	
		SETY *, imm_4		
		CLRY *, imm_4		

■ [ar] + disp

Addressing	The range of a value	Object instruction	Usage type	Address space
[ar] + disp	-1000H - 0FFFH	MODFY.cc *	With no specification	It does not check.
		MOVX r , * SELX r, * TSTX *, imm_4	NUMBER XRAM XROM	
		MOVX * , r SELX * , r SETX *, imm_4 CLRX *, imm_4	NUMBER XRAM	
		MOVY r , * SELY r , * TSTY *, imm_4	NUMBER YRAM YROM	
		MOVY * , r SELY * , r SETY *, imm_4 CLR"Y *, imm_4	NUMBER YRAM	

■ imm_16

Addressing	The range of a value	Object instruction	Usage type	Address space
imm_16	0H - 0FFFFH	MOV r , * AND A, r, * ANDH A, r, * OR A, r, * ORH A, r, * XOR A, r, * XORH A, r, *	None	It does not check.

	-8000H - 7FFFH	ADD A, r, *
		ADDH A, *, im2
		SUB A, r, *
		SUBH A, *, im2
		CMP A, *
		CMPH A, *
		THRU A, *
		THRUH A, *
		MAX A, r, *
		MAXH A, r, *
		MIN A, r, *
		MINH A, r, *		

	0H - 0FFFFH	SFUOP1 A,*,im2	NUMBER	

■ imm_2, imm_4, imm_5, imm_6, imm_8

Addressing	The range of a value	Object instruction	Usage type	Address space
imm_2	0H - 3H	MVSFU A, sr, *	NUMBER	It does not check.
		SFUOP2 A,r,im16,*	NUMBER	It does not check.
imm_4	0H - 0FH	TSTX ea , * TSTY ea , * SETX ea , * SETY ea , * CLRX ea , * CLRY ea , *	NUMBER	It does not check.
imm_5	0H - 1FH	SALH.cc A , * SARH.cc A , * SHLH.cc A , * SHRH.cc A , *	NUMBER	It does not check.
imm_6	0H - 3FH	SAL A , * SAR A , * SHL A , * SHR A , *	NUMBER	It does not check.
		INS A,r1,r2,im6,* INSU A, r, im6, * INSS A, r, im6, * EXTR A,r1,r2,im6,* EXTRU A, r, im6, * EXTRS A, r, im6, *	NUMBER	It does not check.
		INS A,r1,r2,*,im6 INSU A, r, *, im6 INSS A, r, *, im6 EXTR A,r1,r2,*,im6 EXTRU A, r, *, im6 EXTRS A, r, *, im6	NUMBER	It does not check.
imm_8	0H - 0FFH	ANDH A, r, * ORH A, r, * XORH A, r, *	NONE	It does not check.

■ [disp_9] , [disp_13]

Addressing	The range of a value	Object instruction	Usage type	Address space
[disp_9]	-100H - 0FFH	JMP *	NUMBER	CODE
		JSR *	CODE	
[disp_13]	-1000H - 0FFFH	JMP.cc *	NUMBER	CODE
		JMPD.cc *	CODE	
		JSR.cc *		
		JSRD.cc *		

5.8.3 Conversion rule of the value when describing a fraction to an immediate operand

Here, the conversion rule of the value when describing a fraction value is explained to the operand of an immediate value.

- When a value is 1.0 or more

Overflow warning is outputted.

In the case of imm_12 and imm_16	7FFFh
imm_9	0FFh
imm_5	0Fh
imm_4	7h
DW pseudo-instruction	7FFFFFh

- When a value is less than -1.0

Overflow warning is outputted.

In the case of imm_12 and imm_16	8000h
imm_9	100h
imm_5	10h
imm_4	8h
DW pseudo-instruction	800000h

- It is the case of -0.000061 or more smaller than 0

(When it is smaller than 0 and larger than -0.00000011921 in DW)

Overflow warning is outputted.

In the case of imm_12 and imm_16	8000h
imm_9	100h
imm_5	10h
imm_4	8h
DW pseudo-instruction	800000h

- Other case

The value which shifted the numerical value of the head following decimal point so that it might come to the most significant bit of an operand when a fraction was changed into a binary.

An example is shown below.

Example MOV X, 0.2 ;

↓

0.0001100110011001100110011001..(B

↓

1999h

Since this example is operand imm_12 and imm_16 of a MOV instruction, it is changed into the value shifted so that below decimal point when changing 0.2 into a binary number might become the 16-bit most significant from the 1st place. The changed value is set to 1999h.

The shifted value is changed as follows.

In the case of imm_9, it becomes the 9-bit most significant.

In the case of imm_5, it becomes the 5-bit most significant.

In the case of imm_4, it becomes the 4-bit most significant.

In the case of the operand of DW, it becomes the 24-bit most significant.

5.9 Restriction of a basic instruction

Here, description restrictions when describing a basic instruction are explained.

In the following explanation, the case which (G) and (D) abbreviate to the case which describes the character in a parenthesis, respectively exists.

(.cc) has the case which is not described to be the case which describes a condition identifier.

In addition, when there is no description especially, RASi3 outputs an error to these restrictions.

5.9.1 MOVXY instruction use restriction

In the case of addressing from which addressing of a MOVXY instruction is except address register indirect, and both the X side Y sides differ mutually, it forbids specifying the same base register.

Moreover, in the case of memory read both the X and Y sides forbid specifying the same destination register.

5.9.2 Parallel instruction use restrictions

In a parallel instruction, it forbids specifying the same destination register by a transfer instruction and operation instruction.

5.9.3 Repeat instruction use restrictions

The following instructions cannot be described in the next instruction (instruction for a repeat) of a REP instruction.

- The instruction which changes the value of LC

  MOVXY

  MOVX MOVY

  MOV
- Programmed control instruction

  (G)JMP(D)(.cc) (G)JSR(D)(.cc)

  RET(I)(D)(.cc) RETICE(.cc)

  REP TRAP

  DO EXIT(.cc)

5.9.4Use restrictions of a loop instruction

The following contents cannot be described to the end address of DO instruction.

- Specification of the expression containing a usage type of CODE label, or the expression containing a front reference symbol
- Specification of the next address
- Specification of the address same when performing a multiplex loop
- And use of an instruction of the following in an address or the address in front of two of them

(G)JMP(D)(.cc)	(G)JSR(D)(.cc)
RET(I)(D)(.cc)	RETICE(.cc)
REP	TRAP
DO	EXIT(.cc)

- The following instructions are used in front of the four address of the end address.
  - The instruction which changes the value of LC

    MOVXY

    MOVX MOVY

    MOV
  - The instruction which changes the value of LE

    MOV

- Use of DO instruction over program space.
  - Prohibition of arrangement of the end address to a different relocatable segment
  - Use of the instruction which changes a PPC register within the DO loop

    When the following instructions which may change a PPC register are used within the DO loop, RASi3 outputs warning.

(G)JMP(D)(.cc)	(G)JSR(D)(.cc)
RET(I)(D)(.cc)	RETICE(.cc)

5.9.5 Use restrictions of program control instruction

Let the following instructions be disables within 2 instructions from immediately after (G)JMP(D), (G)JSR(D), RET(I)(D), RETICE, EXIT instruction.

(G)JMP(D)(.cc)	(G)JSR(D)(.cc)
RET(I)(D)(.cc)	RETICE(.cc)
DO	EXIT
REP	TRAP

6 Pseudo-instruction

In this section, the details of the pseudo-instruction which can be specified by RASi3 are explained.

A pseudo-instruction is an instruction which RASi3 prepares uniquely, and aims at performing control of management of a program, or assembling processing.

The list of the pseudo-instructions which can be used by RASi3 is shown in the following table.

Classification	Instruction	Feature	Description restrictions	A corresponding option
Assembler initial setting	TYPE	A target device is specified.	The head of a program	/T
Segment definition	SEGMENT	A relocatable segment is defined.		
	STACKSEG	A stack segment is defined.		
Segment control	CODESEG	It changes to an absolute CODE segment.		
	PRAMSEG	It changes to an absolute PRAM segment.		
	XRAMSEG	It changes to an absolute XRAM segment.		
	XROMSEG	It changes to an absolute XROM segment.		
	YRAMSEG	It changes to an absolute YRAM segment.		
	YROMSEG	It changes to an absolute YROM segment.		
	RELSEG	It changes to a relocatable segment.		
Linkage control	EXTRN	An external reference symbol is defined.		
	PUBLIC	A public declaration of a symbol is made.		
	COMM	A communal symbol is defined.		
Symbol definition	EQU, =	A user symbol is defined.		
	DEFINE	A user symbol is defined.		/DEF
Address control	ORG	A start address is specified.	Inside of a segment	
Memory initialization	DW	A memory is initialized.	Only inside of CODE, XROM, and a YROM segment	
		A memory area is secured.	All segments	

Assembling control	INCLUDE	A file is read.		
	END	The end of a program is defined.		
Condition assembling	IF	Condition assembling is performed.		
	IFE			
	IFDEF			
	IFNDEF			
	IFB			
	IFNB			
	ELSE			
	ENDIF			
Listing control	TITLE	The title name of a list file is specified.		
	PAGE	Form feed specification and the number of lines of a list file, and the number of characters of one line are specified.		/PL /PW
	PRN	A list file is outputted.		/PR
	NOPRN	A list file is not outputted.		/NPR
	LIST	An assembling list is generated.		/L
	NOLIST	An assembling list is not generated.		/NL
	SYM	A symbol list is generated.		/S
	NOSYM	A symbol list is not generated.		/NS
	REF	A cross reference list is generated.		/R
	NOREF	A cross reference list is not generated.		/NR
	ERR	An error list file is outputted.		/E
	NOERR	An error list file is not outputted.		/NE
	OBJ	An object file is outputted.		/O
	NOOBJ	An object file is not outputted.		/NO
Macro definition	MACRO	A macro symbol is defined.		
	LOCAL	A local symbol is defined in the macro body.	The head of a macro body	
	REPT	The imperative sentence from REPT to ENDM is repeated by the number of times of specification.		

	IRP	The imperative sentence from IRP to ENDM is repeated by the number of times of specification. In this time, a temporary parameter is transposed to a real parameter.		
	ENDM	The end of macro body is defined.		
	EXITM	Terminate of macro.	The inside of a macro body.	
Scope definition	SCOPE	A scope is defined.		
Optimization	GJMP[.cc]	It changes into the optimal conditional branch.	Inside of a CODE segment	
	GJMPD[.cc]			
	GJSR[.cc]	It changes into the optimal subroutine call.	Inside of a CODE segment	
	GJSRD[.cc]			
C debugging information	CFILE	The file information on the C language is given.		
	CFUNCTION	The function starting position of the C language is shown.	After CFILE	
	CFUNCTIONEND	The function end position of the C language is shown.	After CFUNCTION	
	CARGUMENT	The function argument definition information on the C language is given.	After CFUNCTION	
	CBLOCK	The block starting position of the C language is shown.	After CFUNCTION	
	CBLOCKEND	The block end position of the C language is shown.	After Corresponding CBLOCK	
	CLABEL	The label definition information on the C language is given.	After CFUNCTION	
	CLINE	The line number of the C language is given.	After CFUNCTION	
	CGLOBAL	The global variable definition information on the C language is given.		
	CSGLOBAL	The static global variable definition information on the C language is given.	Just before a corresponding area	
	CLOCAL	The local variable definition information on the C language is given.	Between CBLOCK and CBLOCKEND	
	CSLOCAL	The static local variable information on the C language is given.	Between CBLOCK and CBLOCKEND	
	CSTRUCTTAG	The structure tag definition information on the C language is given.		

	CSTRUCTMEM	The structure member definition information on the C language is given.	After CSTRUCTTAG	
	CUNIONTAG	The union tag definition information on the C language is given.		
	CUNIONMEM	The union member definition information on the C language is given.	After CUNIONTAG	
	CENUMTAG	The tag definition information on the enumerated type variable of the C language is given.		
	CENUMMEM	The member definition information on the enumerated type variable of the C language is given.	After CENUMTAG	
	CTYPEDEF	The definition information on typedef of the C language is given.		
	CENVINFO	The information on compile environment is given.		
	CMACINFO	The macro information on the C language is given.		

6.1 Assembling initial-setting pseudo-instruction

An assembler initialization pseudo-instruction is for setting assembling conditions to RASi3. Therefore, it is necessary to describe an assembler initial-setting pseudo-instruction to the beginning of a program.

6.1.1 TYPE pseudo-instruction

- Syntax

TYPE (*device_name*)

device_name : Target device name

- Description

A TYPE pseudo-instruction is a pseudo-instruction for specifying the DCL file name corresponding to the target target device. RASi3 reads the information on a DCL file that device_name is used into a base name and it uses ".DCL" as an extension.

A DCL file is searched in the following order.

(1) Current directory

(2) The directory where started RASi3.EXE exists

(3) The directory specified as the environment variable DCLi3

RASi3 reads the contents of the DCL file before assembling processing. Even if an error is in the contents of the DCL file, it reads to the end of a DCL file. RASi3 will be ended if all DCL errors to occur are displayed. If reading of a DCL file is normal, assembling of a source file will be started continuously.

・Supplement

You have to specify a TYPE pseudo-instruction or a /T option. Moreover, you have to specify a TYPE pseudo-instruction at the head of a program.

If a TYPE pseudo-instruction is specified twice or more, RASi3 will output a fatal error.

6.2 Segment definition pseudo-instruction

Segment definition pseudo-instructions include the STACKSEG pseudo-instruction which defines the SEGMENT pseudo-instruction and stack segment which define a relocatable segment.

6.2.1 SEGMENT pseudo-instruction

・Syntax

segment_symbol SEGMENT *seg_type* [*boundary*] [, *access*] [, *link_atr*]

・Description

A SEGMENT pseudo-instruction defines a relocatable segment. A relocatable segment can be defined as one program to 65535 pieces. The number called segment ID in order of a definition is given to each segment.

segment_symbol is used for discernment of a relocatable segment. This segment_symbol is specified as the operand of a RELSEG pseudo-instruction. Moreover, segment_symbol can also be used for the operand of an instruction. In this case, segment_symbol indicates the base address of a relocatable segment and the value under assemble is set to 0.

seg_typ

The segment type showing the kind of address space which assigns a relocatable segment is specified as *seg_typ*. Only one can be specified out of the following segment type.

Segment type	Allocated Memory space
CODE	CODE address space
PRAM	PRAM address space
XRAM	XRAM address space
XROM	XROM address space
YRAM	YRAM address space
YROM	YROM address space

boundary

The boundary value of a head address when a relocatable segment is assigned is specified as *boundary*. This is called boundary value attribute of a logical segment. The integer constant is specified as boundary.

An abbreviation of boundary specifies 1.

access

The keyword (SHORT10, SHORT12 and LONG16) showing the attribute of the address range is specified as *access*. The meaning of each keyword is as follows.

The attribute of the address range	Allocated address range
SHORT10	0H - 3FFH
SHORT12	0H - 0FFFH
LONG16	0H - 0FFFFH

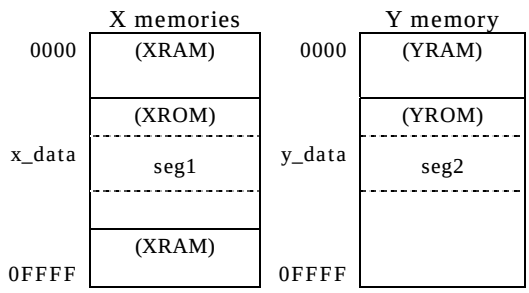
A segment symbol is described in link_atr (linkage attribute). In the same source file, the two segment which has the pair of linkage attribute is allocated the segment start address of respectively same value by Linker.

- Example

```
seg1  SEGMENT  XROM ,  seg2
seg2  SEGMENT  YROM ,  seg1

      RELSEG   seg1
x_data:
      DW   10h,  20h
  --→

      RELSEG   seg2
y_data:
      DW   30h,  40h
```



In this case, seg1 and seg2 are assigned to the address with respectively same X memory and Y memory.

- Supplement

In the following description, RASi3 outputs an assembling error.

(1) When a segment symbol with segment type on the same memory space is described

(2) When a linkage attribute is specified to a segment type CODE segment

(3) When the segment name which is not defined in the source file is described

6.2.2 STACKSEG pseudo-instruction

- Syntax

STACKSEG [*seg_type* ,] *size*

- Description

A STACKSEG pseudo-instruction defines a stack segment. *seg_type* is the segment which assigns a stack area and can specify XRAM or YRAM. The size of a stack area is specified as *size*.

If a STACKSEG pseudo-instruction is specified, RASi3 defines a stack segment which are named as a $XSTACK or a $YSTACK. Although $XSTACK or $YSTACK is one of the relocatable segments, it cannot specify $XSTACK or $YSTACK as the operand of a RELSEG pseudo-instruction.

- Supplement

The initial value of a stack pointer, i.e., the end address of a stack segment, can be referred to by _$$XSP or _$$YSP. _$$XSP or _$$YSP is the symbol prepared in order to access SP register, and in case use it, you need to make an external reference declaration by EXTRN pseudo-instruction.

6.3 Segment control pseudo-instruction

Segment control pseudo-instructions include CODESEG, PRAMSEG, XRAMSEG, XROMSEG, YRAMSEG and YROMSEG pseudo-instruction which define an absolute segment, and RELSEG pseudo-instruction which defines a relocatable segment.

6.3.1 CODESEG pseudo-instruction

- Syntax

CODESEG [*address*]

CODESEG *address* OVL *real_address*

- Description

CODESEG pseudo-instruction declares the start of an absolute CODE segment definition. The definition with an OVL identifier is described when defining the absolute segment for overlay.

The start address of the logical segment to define is specified as *address*. The *address* is the constant expression which does not include forward reference. The value of a location counter is updated by the specified address by *address*. When *address* is omitted, a segment is started from the address following the end address of the absolute segment defined by the last CODESEG pseudo-instruction. When there is no CODESEG pseudo-instruction before, a segment is started from the minimum value of a CODE address space.

When using overlay, the value of a label etc. is solved by the value of address during an assemble, but it is assigned to real_address of a XROM area in practice.

- Supplement

The value specified as address must be within the limits of the CODE address space defined by the DCL file. However, when OVL is specified, the value specified to address must be within the limits of a PRAM address space.

6.3.2 PRAMSEG pseudo-instruction

- Syntax

PRAMSEG [*address*]

- Description

PRAMSEG pseudo-instruction declares the start of an absolute PRAM segment definition.

The start address of a logical segment is specified as *address*. The *address* is the constant expression which does not contain forward reference. The value of a location counter is updated by the specified address by *address*. When *address* is omitted, a segment is started from the address following the end address of the absolute segment defined by the last PRAMSEG pseudo-instruction. When there is no PRAMSEG pseudo-instruction before, a segment is started from the minimum value of a PRAM address space.

- Supplement

The value specified as address must be within the limits of the PRAM address space defined by the DCL file.

6.3.3 XRAMSEG pseudo-instruction

- Syntax

XRAMSEG [*address*]

- Description

A XRAMSEG pseudo-instruction declares the start of an absolute XRAM segment definition.

The start address of a logical segment is specified as *address*. The *address* is the constant expression which does not contain forward reference. The value of a location counter is updated by the specified address by *address*. When *address* is omitted, a segment is started from the address following the end address of the absolute segment defined by the last XRAMSEG pseudo-instruction. When there is no XRAMSEG pseudo-instruction before, a segment is started from the minimum value of a XRAM address space.

- Supplement

The value specified as address must be within the limits of the XRAM address space defined by the DCL file.

6.3.4 XROMSEG pseudo-instruction

- Syntax

XROMSEG [*address*]

- Description

A XROMSEG pseudo-instruction declares the start of an absolute XROM segment definition.

The start address of a logical segment is specified as *address*. The *address* is the constant expression which does not contain forward reference. The value of a location counter is updated by the specified address by *address*. When *address* is omitted, a segment is started from the address following the end address of the absolute segment defined by the last XROMSEG pseudo-instruction. When there is no XROMSEG pseudo-instruction before, a segment is started from the minimum value of a XROM

address space.

- Supplement

The value specified as address must be within the limits of the XROM address space defined by the DCL file.

6.3.5 YRAMSEG pseudo-instruction

- Syntax

YRAMSEG [*address*]

- Description

A YRAMSEG pseudo-instruction declares the start of an absolute YRAM segment definition.

The start address of a logical segment is specified as *address*. The *address* is the constant expression which does not contain forward reference. The value of a location counter is updated by the specified address by *address*. When *address* is omitted, a segment is started from the address following the end address of the absolute segment defined by the last YRAMSEG pseudo-instruction. When there is no YRAMSEG pseudo-instruction before, a segment is started from the minimum value of a YRAM address space.

- Supplement

The value specified as address must be within the limits of the YRAM address space defined by the DCL file.

6.3.6 YROMSEG pseudo-instruction

- Syntax

YROMSEG [*address*]

- Description

A YROMSEG pseudo-instruction declares the start of an absolute YROM segment definition.

The start address of a logical segment is specified as *address*. The *address* is the constant expression which does not contain forward reference. The value of a location counter is updated by the specified address by *address*. When *address* is omitted, a segment is started from the address following the end address of the absolute segment defined by the last YROMSEG pseudo-instruction. When there is no YROMSEG pseudo-instruction before, a segment is started from the minimum value of a YROM address space.

- Supplement

The value specified as address must be within the limits of the YROM address space defined by the DCL file.

6.3.7 RELSEG pseudo-instruction

- Syntax

RELSEG *seg_sym*

- Description

A RELSEG pseudo-instruction declares the start of a relocatable segment definition.

The segment symbol of a relocatable segment is specified as seg_sym. The seg_sym must be defined by the SEGMENT pseudo-instruction before the description position of a RELSEG pseudo-instruction. A relocatable segment is started from the 0^{th} address during an assemble. It is assigned to an actual address space by Rli3 linker.

6.4 Linkage control pseudo-instruction

A linkage control pseudo-instruction is used when mainly splitting and creating a program to two or more files.

6.4.1 EXTRN pseudo-instruction

- Syntax

EXTRN *usage_type* [*access*] : *symbol* [*symbol* --]

- Description

An EXTRN pseudo-instruction declares a external symbol. In order to refer to the symbol defined by the external module, it is necessary to surely declare by EXTRN pseudo-instruction.

The usage type of a external symbol is specified as *usage_type*. To *usage_type*, only one can be specified out of the following segment type.

Segment type	Allocated Memory space
CODE	It has an address on a CODE address space.
PRAM	It has an address on a PRAM address space.
XRAM	It has an address on a XRAM address space.
XROM	It has an address on a XROM address space.
YRAM	It has an address on a YRAM address space.
YROM	It has an address on a YROM address space.
NONE	It has the address which does not pinpoint space.
NUMBER	It has an integer type.
FLOAT	It has a fraction type.

The keyword of the attribute (SHORT10, SHORT12 and LONG16) of the address range is specified as *access*. The meaning of each keyword is as follows.

The attribute of the address range	Allocated address range
SHORT10	0H - 3FFH
SHORT12	0H - 0FFFH
LONG16	0H - 0FFFFH

- Supplement

The *usage_type* and *access* must be the same types as the symbol defined by the external module. When unmatched, RLi3 outputs an error.

The symbol already defined cannot be specified as *symbol*. Moreover, an EXTRN pseudo-instruction cannot refer a segment symbol.

6.4.2 PUBLIC pseudo-instruction

- Syntax

PUBLIC *symbol* [*symbol*--]

- Description

A PUBLIC pseudo-instruction declares a local symbol as a public symbol. By declaring a local symbol as public, the symbol can be used from other source files.

A local symbol and public declaration of the symbol may define whichever first.

- Supplement

In order to refer to a public symbol from other source files, you have to declare the external symbol of the same name in the source file to refer to using an EXTRN pseudo-instruction.

The public symbol of the same name cannot be defined in two or more source files.

6.4.3 COMM pseudo-instruction

- Syntax

symbol COMM *seg_type size* [*boundary*] [*access*]

- Description

A COMM pseudo-instruction defines a shared symbol (communal symbol). A shared symbol defines a common data area by two or more source files.

The shared symbol defined by two or more source files indicates the head address of a common data area. RLi3 determines the address of the data area where a shared symbol is defined.

The syntax of a COMM pseudo-instruction is similar of the SEGMENT pseudo-instruction. However, the size of the area to secure is specified just after *seg_type*.

seg_typ

The segment type showing the kind of address space which assigns a relocatable segment is specified as *seg_typ*. Only one can be specified out of the following segment type.

Segment type	Allocated Memory space
PRAM	PRAM address space
XRAM	XRAM address space
XROM	XROM address space
YRAM	YRAM address space
YROM	YROM address space

boundary

The boundary value of a head address in case a relocatable segment is assigned is specified as *boundary*. This is called boundary value attribute of a logic segment. The integer constant is specified as *boundary*.

An abbreviation of boundary specifies 1.

access

The keyword of the attribute (SHORT10, SHORT12 and LONG16) of the address range is specified as *access*. The meaning of each keyword is as follows.

The attribute of the address range	The range of the address [allocation / address]
SHORT10	0H - 3FFH
SHORT12	0H - 0FFFH
LONG16	0H - 0FFFFH

- Supplement

It will become an error if the same shared symbol is declared twice or more in one source file.

When the sizes of the shared symbol of each source file differ, the greatest area in the size to specify is assigned to memory.

6.5 Symbol definition pseudo-instruction

A symbol definition pseudo-instruction is a pseudo-instruction for defining a symbol and giving a value or an address value to the symbol.

6.5.1 EQU pseudo-instruction

- Syntax

symbol EQU *simple_expression*

- Description

An EQU pseudo-instruction defines a local symbol. The symbol to define is specified as *symbol* and the simple type which does not include forward reference is specified as *simple_expression*.

The symbol defined by the EQU pseudo-instruction has the value and attribute of simple_expression. That is, the symbol to define is an absolute symbol when simple_expression is constant expression. When simple_expression is a simple relocatable expression, the symbol to define is a simple relocatable symbol. Moreover, if simple_expression is a numerical value type expression, the usage type of a symbol is NUMBER or FLOAT, and if simple_expression is an address type, a symbol has the property of the address of simple_expression.

- Supplement

The symbol already defined cannot be specified as *symbol*.

When the usage type of the symbol defined by this pseudo-instruction is except NUMBER, FLOAT, and NONE, RASi3 confirms whether the usage type of the kind of memory and a symbol to the address is right. When the kind and usage type of memory are not right, RASi3 displays warning.

6.5.2 = Pseudo-instruction

- Syntax

symbol = *simple_expression*

- Description

The = pseudo-instruction defines a local symbol. The symbol to define is specified as *symbol* and the simple type which does not include forward reference is specified as *simple_expression*.

The symbol defined by the = pseudo-instruction has the value and attribute of simple_expression. That is, the symbol defined is an absolute symbol when simple_expression is constant expression. When simple_expression is a simple relocatable expression, the symbol defined is a simple relocatable symbol. Moreover, if simple_expression is a numerical value type expression, the usage type of a symbol will serve as NUMBER or FLOAT, and if simple_expression is an address type, a symbol has the property of the address of simple_expression.

- Supplement

The symbol defined by the = pseudo-instruction can be redefined by an = pseudo-instruction.

6.5.3 DEFINE pseudo-instruction

- Syntax

DEFINE *def_sym* "*def_body*"

- Description

A DEFINE pseudo-instruction defines a macro symbol.

A DEFINE pseudo-instruction assigns *def_body* to *def_sym*. If symbol appears in source statement after this definition, RASi3 will be replaced to def_body of it, and will assemble it. The number of characters which can be specified as def_body is a maximum of 255 bytes.

In def_body, you may describe another macro symbol. In this case, RASi3 performs still more macroscopic replacement, when replacing the first macro. As for a maximum of 8 level, the macroscopic nest is allowed.

- Supplement

A macro symbol can be referred to only after a DEFINE pseudo-instruction defines.

6.6 Address control pseudo-instruction

6.6.1 ORG pseudo-instruction

- Syntax

ORG *address*

- Description

An ORG pseudo-instruction resets the value of the location counter of the logical segment which belongs as the value of address. The features of an ORG pseudo-instruction differ by the case where a logic segment is an absolute, and the case where a logic segment is relocatable.

(1) When a logic segment belongs to an absolute segment

The value of a location counter is set to address by the constant expression which does not include forward reference.

Constant expression must be a value more than the head address of the logic segment which belongs. Moreover, the value of constant expression must be in the target address space.

(2) When a logic segment belongs to a relocatable segment

The value of a location counter is specified as address by the simple formula which does not include forward reference.

When a simple type is an address type, the present location counter is changed into the value of an expression.

When a simple type is constant expression, the present current location is changed for the value of an expression as offset from the head of a segment.

When a simple relocatable symbol is contained in simple expression, the

relocatable segment to which the simple relocatable symbol belongs must be the present relocatable segment.

6.7 Memory initialization pseudo-instruction

6.7.1 DW pseudo-instruction

- Syntax

[*symbol*] DW *expression* [, *expression* --]

expression = General formula

= *count* DUP (*expression* [, *expression* --])

- Description

DW pseudo-instruction initializes a memory by a word unit. The *expression* (a general formula or duplicate type) can be specified as an operand. The number of operands does not have restriction.

The symbol specified by *symbol* is defined as a label with the start address and segment type of the memory area initialized by DW pseudo-instruction.

You may include forward reference in *expression*. 1-word data is specified as *expression*.

The duplicate type specified as *expression* is used when initializing the continuous range with the same value. Only the number of times which specified initialization by the value of expression described by the operand of a duplicate type by count is repeated.

When '?' is specified as *expression*, only reservation of an area is performed without initializing.

- Supplement

The range of the value which can be described at a expression must be 0FFFFH from -0FFFFH. When the range is exceeded, RASi3 displays warning.

DW pseudo-instruction accompanied by initialization can be described only to CODE, XROM, and YROM segment.

DW pseudo-instruction without initialization can be described to PRAM, XRAM, YRAM segment. When DW pseudo-instruction is described in other segments, RASi3 displays an error.

The nest level of a duplicate type is unrestricted.

6.8 Assembling control pseudo-instruction

6.8.1 INCLUDE pseudo-instruction

- Syntax

  INCLUDE (*file_name*)

- Description

An INCLUDE pseudo-instruction reads the include file specified by *file_name*. Description of an INCLUDE pseudo-instruction inserts the contents of the include file in the position.

An INCLUDE pseudo-instruction is further used in an include file, and another file can be inserted. A maximum of 8 levels can nest an INCLUDE pseudo-instruction.

When an END pseudo-instruction is detected in an include file, RASi3 stops the assemble after the END pseudo-instruction in the include file, and returns to assemble processing of the source file which called the include file.

An include file is searched in following order.

(1) Current directory

(2) /I Search path specified as the option

6.8.2 END pseudo-instruction

- Syntax

  END

- Description

An END pseudo-instruction is a pseudo-instruction for informing RASi3 about the end of a program. RASi3 assembles even an END pseudo-instruction. Even if it has described the source statement after an END pseudo-instruction, RASi3 ignores it. Moreover, when an END pseudo-instruction is in an include file, RASi3 stops assembling after the END pseudo-instruction in the include file, and returns to assembling processing of the source file which called the include file.

6.9 Condition assembling pseudo-instruction

If a condition assemble pseudo-instruction is used, only when certain conditions are satisfied, it can control to assemble the arbitrary blocks of a program. As a result, one source program can be used for two or more purposes.

A condition assemble feature is realized by describing a condition assemble pseudo-instruction. The syntax of a condition assemble pseudo-instruction is as follows.

```
IFxxx    conditional_operand
         true_conditional_body
[ELSE
         false_conditional_body]
ENDIF
```

IFxxx indicates one of the next condition assembling pseudo-instructions.

IF IFE IFDEF IFNDEF IFB IFNB

conditional_operand is the expression and symbol which give the truth conditions of condition assembling. The contents specified as *conditional_operand* change with condition assembling pseudo-instructions.

true_conditional_body and *false_conditional_body* indicate the block of a source statement. When conditions are truth, the statement block of *true_conditional_body* is assembled. When conditions are false, the statement block of *true_conditional_body* is skipped. *false_conditional_body* is assembled when there is an ELSE pseudo-instruction at this time.

To *true_conditional_body* or *false_conditional_body*, you may describe a condition assembling pseudo-instruction further. As for a condition assemble pseudo-instruction, a maximum of 15 level can nest.

6.9.1 IF, IFE pseudo-instruction

- Syntax

IF *expression*

IFE *expression*

- Description

IF pseudo-instruction will assemble true_conditional_body, if the value of expression is truth, and if it is a false, it assembles false_conditional_body.

An IFE pseudo-instruction will assemble true_conditional_body, if denial of the value of expression is truth, and if it is a false, it assembles false_conditional_body.

The *expression* is the constant expression which does not include forward reference.

If expression contains forward reference, and if a grammatical error is in expression,

conditions are judged to be a false.

6.9.2 IFDEF, IFNDEF pseudo-instruction

- Syntax

  IFDEF *symbol*

  IFNDEF *symbol*

- Description

The *symbol* is a symbol except a reserved word.

If symbol specified as the IFDEF pseudo-instruction is defined before this source statement, conditions become truly. Conditions become false, if symbol is not defined in the program, or if it defines after this source statement.

Conditions become false if symbol specified as the IFNDEF pseudo-instruction is defined before this source statement. If symbol is not defined in the program, or if it defines after this source statement, conditions become truly.

6.9.3 IFB, IFNB pseudo-instruction

- Syntax

  IFB <*argument*>

  IFNB <*argument*>

- Description

The *argument(s)* are arbitrary character strings.

If argument specified as the IFB pseudo-instruction is null statement, conditions will become truly. Conditions will become false if argument is not null statement.

If argument specified as the IFNB pseudo-instruction is not null statement, conditions will become truly. Conditions will become false if argument is null statement.

6.10 Listing control pseudo-instruction

6.10.1 TITLE pseudo-instruction

- Syntax

  TITLE "*string*"

- Description

A TITLE pseudo-instruction specifies the title of a print file. This title is outputted to the header of each page of a print file.

The character string of less than 70 characters made into a title is specified as *string*. If the character string exceeding 70 characters is specified as *string*, it will be ignored after the 71st character.

Although two or more TITLE pseudo-instructions can be used into a program, available TITLE is specified at the end. An abbreviation of a TITLE pseudo-instruction does not output a title to the header of a print file.

6.10.2 PAGE pseudo-instruction

- Syntax

PAGE [*line*] [, *column*]

- Description

The PAGE pseudo-instruction without an operand has a different feature from a PAGE pseudo-instruction with an operand.

The PAGE pseudo-instruction which does not specify an operand inserts a form feed in a print file forced. It becomes the next page from the line the PAGE pseudo-instruction was described to be.

As for the operand of a PAGE pseudo-instruction, the number of lines of each page of a print file and the number of characters of each line are specified. The number of lines of 1 page is specified as line, and the number of characters of one line is specified as column. Both line and column are the constant expression which does not contain forward reference.

The range of the value of *line* is from 10 to 65535. If a value smaller than 10 is specified to *line*, it will be rectified by 10, and it will be rectified by 65535 if a larger value than 65535 is specified.

The range of the value of *column* is from 79 to 255. If a value smaller than 79 is specified to *column*, it will be rectified by 79, and it will be rectified by 255 if a larger value than 255 is specified.

- Supplement

A PAGE pseudo-instruction is ignored in the range with an available NOLIST pseudo-instruction.

6.10.3 PRN, NOPRN pseudo-instruction

- Syntax

PRN [(*file_specification*)]

NOPRN

- Description

These pseudo-instructions control making of a print file.

A print file is created when a PRN pseudo-instruction is specified. A print file name is specified to file_specification.

A print file is not created when a NOPRN pseudo-instruction is specified.

A print file is created if a PRN pseudo-instruction and a NOPRN pseudo-instruction are omitted. As for the print file name in this case, the extension of a source file name replaces ".PRN."

- Supplement

These pseudo-instructions specify either only once. If specified 2 times or more, it was specified effectively first. Moreover, priority is given to /PR and a /NPR option over specification of a pseudo-instruction.

6.10.4 LIST, NOLIST pseudo-instruction

- Syntax

LIST

NOLIST

- Description

These pseudo-instructions control the output of the assembling list to a print file.

An assemble list is a list of a program and object code. By using a LIST pseudo-instruction and a NOLIST pseudo-instruction, the range of the program outputted to an assemble list can be specified.

Description of a LIST pseudo-instruction outputs the program from the next line to an assemble list.

Description of a NOLIST pseudo-instruction stops outputting the program from the next line to an assemble list. However, source statement including an error or warning is outputted to an assemble list regardless of these pseudo-instructions.

When a LIST pseudo-instruction and a NOLIST pseudo-instruction are omitted, all programs are outputted to an assembling list.

- Supplement

If a /L option is specified to RASi3, each statement until a NOLIST pseudo-instruction appears in a program is outputted to an assemble list. That is, it becomes the same operation as the case where a LIST pseudo-instruction is described at the head of a program.

If a /NL option is specified to RASi3, each statement until a LIST pseudo-instruction appears in a program is not outputted to an assemble list. That is, it becomes the same operation as the case where a NOLIST pseudo-instruction is described at the head of a program.

6.10.5 SYM, NOSYM pseudo-instruction

- Syntax

SYM

NOSYM

- Description

These pseudo-instructions control the output of the symbol list to a print file.

The information on the user symbol used for the program is included in a symbol list. By using a SYM pseudo-instruction and a NOSYM pseudo-instruction, it is specified whether a symbol list is outputted.

If a SYM pseudo-instruction is specified, the information on all user symbols is outputted to a symbol list. A symbol list is not created if a NOSYM pseudo-instruction is specified.

A symbol list is not outputted if a SYM pseudo-instruction and a NOSYM pseudo-instruction are omitted.

- Supplement

When two or more these pseudo-instructions were specified, it was specified effectively first. Moreover, priority is given to /S and a /NS option rather than a pseudo-instruction.

6.10.6 REF, NOREF pseudo-instruction

- Syntax

REF

NOREF

- Description

These pseudo-instructions control the output of the cross reference list to a print file.

The line number for which the user symbol defined by the program and each user symbol were used is contained in a cross reference list. By using a REF pseudo-instruction and a NOREF pseudo-instruction, the user symbol outputted to a cross reference list is controllable.

In the range until a NOREF pseudo-instruction appears from the next line of the line which described the REF pseudo-instruction, the symbol defined or referred to is outputted to a cross reference list.

In the range until a REF pseudo-instruction appears from the next line of the line which described the NOREF pseudo-instruction, the symbol defined or referred to is not outputted to a cross reference list.

- Supplement

If a /R option is specified to RASi3, each statement until a NOREF pseudo-instruction appears in a program is outputted to a cross reference list. This is the same operation as the case where a REF pseudo-instruction is described at the head of a program.

If a /NR option is specified to RASi3, each statement until a REF pseudo-instruction appears in a program is not outputted to a cross reference list. This is the same operation as the case where a NOREF pseudo-instruction is described at the head of a program.

6.10.7 ERR, NOERR pseudo-instruction

- Syntax

ERR [(*file_specification*)]

NOERR

- Description

These pseudo-instructions control making of an error file.

If an ERR pseudo-instruction is specified, an error file will be created and an error message will be outputted to the file. An error file name is specified as file_specification.

Specification of a NOERR pseudo-instruction does not create an error file. In this case, an error message is outputted to a standard error output.

If an ERR pseudo-instruction and a NOERR pseudo-instruction are omitted, an error file is not created but an error message is outputted to a standard error output.

- Supplement

If two or more these pseudo-instructions were specified, it was specified effectively first. Moreover, priority is given to specification of /E and a /NE option over specification of a pseudo-instruction.

6.10.8 OBJ, NOOBJ pseudo-instruction

- Syntax

OBJ [(*file_specification*)]

NOOBJ

- Description

These pseudo-instructions control making of an object file.

Specification of an OBJ pseudo-instruction creates an object file. An object file name is specified as file_specification.

Specification of a NOOBJ pseudo-instruction does not create an object file.

An object file is created if an OBJ pseudo-instruction and a NOOBJ pseudo-instruction are omitted. The object file name in this case is replaced to ".OBJ" of the extension of a source file name.

- Supplement

If two or more these pseudo-instructions were specified, it was specified effectively first. Moreover, priority is given to /O and a /NO option over a pseudo-instruction.

6.11 Macro definition pseudo-instruction

A macro definition pseudo-instruction registers the statements (aggregate of processing) of sequence defined as the macro body as a macro symbol. If a macro symbol is described in a program, RASi3 will expand during an assemble the statement defined as the macro body.

The syntax of a macro definition pseudo-instruction is as follows.

```
name    MACRO    [[parameter] , --]
            :
        Instruction statement (macro body)
```

:

ENDM

The *name* is the macro symbol assigned to the macro body.

In order to call a macro, a macro symbol is described to a program. If RASi3 detects the macro symbol described in the program, it will expand in the statement defined by the macro body.

The *parameter* is the formal parameter specified in the case of macro definition, and is the symbol which can be used only within the macro body.

6.11.1 MACRO pseudo-instruction

- Syntax

name MACRO [[*parameter*] , --]

- Description

A MACRO pseudo-instruction registers the statements (macro body) of sequence to an ENDM pseudo-instruction as a macro symbol specified as name.

The *parameter* is the formal parameter specified in the case of macro definition, and is the symbol which can be used only within the macro body.

In macroscopic expansion, it is replaced to the actual parameter specified at the time of a macro call of the formal parameter in the macro body.

- Supplement

If the same macro symbol is redefined by a MACRO pseudo-instruction, the macro symbol defined later becomes available.

In the macro body, you may describe another macro symbol. In this case, RASi3 performs still more macroscopic replacement in the processing which replaces the first macro. The macroscopic nest allowed by the MACRO pseudo-instruction, the REPT pseudo-instruction, and an IRP pseudo-instruction is a maximum of 24 level.

6.11.2 EXITM pseudo-instruction

- Syntax

EXITM

- Description

An EXITM pseudo-instruction can be used within the macro body. If RASi3 detects an EXITM pseudo-instruction during macroscopic expansion, even subsequent ENDM pseudo-instructions will be ignored. Moreover, if an EXITM pseudo-instruction is detected from the inside of the macro body by the include file called by the INCLUDE pseudo-instruction, macro expansion is terminated after the end of an include file.

- Supplement

An EXITM pseudo-instruction can be described only from a MACRO pseudo-instruction, a REPT pseudo-instruction, or an IRP pseudo-instruction to an ENDM pseudo-instruction.

6.11.3 LOCAL pseudo-instruction

- Syntax

LOCAL *loc_sym* [, *loc_sym* [, --]]

- Description

A LOCAL pseudo-instruction defines a local symbol only with the available inside of a macro.

The symbol specified as loc_sym can be used only within the macro body, and cannot be referred to from the outside of the macro body. It is replaced to the following formats of loc_sym after macro expansion.

?_?*number*

The *number* is the 4 digits hexadecimal from 0 to 0FFFFH, and is the serial number for every assemble.

- Supplement

A LOCAL pseudo-instruction can be used only within the macro body. If RASi3 detects the LOCAL pseudo-instruction used out of the macro body, warning will be displayed and it will be ignored.

Moreover, a LOCAL pseudo-instruction should be described at the head of the macro body. If RASi3 detects the LOCAL pseudo-instruction described by the position which is not the head of the macro body, warning will be displayed and it will be ignored.

6.11.4 REPT pseudo-instruction

- Syntax

REPT *const_expression*

- Description

A REPT pseudo-instruction expands the statements (macro body) of sequence from the description position to an ENDM pseudo-instruction repeatedly by the number of times specified by const_expression.

In the macro body, you may describe another macro symbol. In this case, RASi3 performs still more macroscopic replacement, when replacing the first macro. The macroscopic nest allowed by the MACRO pseudo-instruction, the REPT pseudo-instruction, and an IRP pseudo-instruction is a maximum of 24 level.

6.11.5 IRP pseudo-instruction

- Syntax

IRP *dummy_symbol* < [[*parameter*], --] >

- Explanation

An IRP pseudo-instruction is replaced to parameter of dummy_symbol used for the statement from the description to an ENDM pseudo-instruction, and expands. Only the number is expanded if two or more parameter(s) are specified. If parameter is omitted,

an IRP pseudo-instruction is ignored.

In the macro body, you may describe another macro symbol. In this case, RASi3 is the process which replaces the first macro, and performs still more macroscopic replacement. The macroscopic nest allowed by the MACRO pseudo-instruction, the REPT pseudo-instruction, and an IRP pseudo-instruction is a maximum of 24 level.

6.12 Scope definition pseudo-instruction

6.12.1 SCOPE pseudo-instruction

- Syntax

scope_tag SCOPE [[*global_symbol*] --]

:

ENDC

- Description

A SCOPE pseudo-instruction declares from the description to an ENDC pseudo-instruction as a scope area.

scope_tag is the symbol which shows a scope area name, and calls it a scope tag.

global_symbol is the global symbol name used in a scope area, and calls it a scope symbol.

Some scope areas with the same scope tag name are managed as the same scope.

- Supplement

The nest of a scope pseudo-instruction cannot be described.

6.13 Optimization pseudo-instruction

i-Core3 has some branch instruction. Instead of describing such branch instruction directly, RASi3 is changed into the optimal instruction according to the distance to the address value of a branch place, or a branch place by using an optimization pseudo-instruction.

6.13.1 GJMP, GJMPD, GJSR, GJSRD pseudo-instruction

- Syntax

GJMP[.cc] *label*

GJSR[.cc] *label*

- Description

GJMP pseudo-instruction optimize a branch instruction.

GJSR pseudo-instruction optimize a subroutine call.

label is a symbol showing a branch place and it is also possible to specify a temporary

symbol.

Moreover, if only a label is described to the operand of a basic instruction of JMP, JSR, RASi3 performs optimization equivalent to an optimization pseudo-instruction. However, if an expression is described to an operand, it is managed as a 2-word instruction, and if a SHORT operator is specified, it is managed as a 1-word instruction.

- Supplement

These pseudo-instructions can be described only in a CODE segment.

6.14 C debugging information pseudo-instruction

This section explains each C debugging information pseudo-instruction for information disclosure. However, if these pseudo-instructions are described in the usual assembly source or these pseudo-instructions contained in the source program which CCi3 generates are rewritten, a normal assemble result may not be obtained.

The operand of C debugging information pseudo-instruction has the following restrictions.

(1) A symbol cannot be specified.

(2) An expression cannot be specified.

(3) In a parameter, only an integer constant or a character string can be specified.

6.14.1 CFILE pseudo-instruction

- Syntax

CFILE *file_id total_line inc_id inc_line "filename"*

- Description

A CFILE pseudo-instruction defines the information about the file of C source file.

The *file_id* is a file identification number and it is used for evaluation of the value described by the *file_id* parameter of other C debugging information pseudo-instructions.

6.14.2 CFUNCTION, CFUNCTIONEND pseudo-instruction

- Syntax

CFUNCTION *file_id fn_id*

CFUNCTIONEND *fn_id*

- Description

These pseudo-instructions define the information about the function of C source program. A CFUNCTION pseudo-instruction defines the start of a function and a CFUNCTIONEND pseudo-instruction defines the end of a function.

The *fn_id* is a function identification number and it is used for evaluation of the value

described by the *fn_id* parameter of other C debugging information pseudo-instructions.

6.14.3 CARGUMENT pseudo-instruction

- Syntax

CARGUMENT *attrib size line column offset phy_seg* "*variable_name*" *hierarchy*

- Description

A CARGUMENT pseudo-instruction defines the information about the argument of the function of C source program.

6.14.4 CBLOCK, CBLOCKEND pseudo-instruction

- Syntax

CBLOCK *fn_id block_id line_no*

CBLOCKEND *fn_id block_id line_no*

- Description

These pseudo-instructions define the start and end of block description in the function in C source program

A CBLOCK pseudo-instruction defines the starting position of a block. A CBLOCKEND pseudo-instruction defines the end position of a block.

- Supplement

These pseudo-instructions must be defined as CFUNCTION between CFUNCTIONEND pseudo-instructions.

6.14.5 CLABEL pseudo-instruction

- Syntax

CLABEL *label_no* "*label_name*"

- Description

A CLABEL pseudo-instruction relates the label described on C source program, and the label in the assembly source which CCi3 compiler outputs.

- Supplement

The CLABEL pseudo-instruction must be defined as CFUNCTION between CFUNCTIONEND pseudo-instructions.

6.14.6 CLINE pseudo-instruction

- Syntax

CLINE *line_atr line_no start_column end_column*

- Description

A CLINE pseudo-instruction defines the information about the line number of C source program.

- Supplement

The CLINE pseudo-instruction must be defined as CFUNCTION between CFUNCTIONEND pseudo-instructions.

6.14.7 CGLOBAL pseudo-instruction

- Syntax

CGLOBAL *file_id usg_typ attrib size line column "variable_name" hierarchy*

- Description

A CGLOBAL pseudo-instruction defines the information about the global variable defined in C source program. The *variable_name* is the name of the global variable on C source program. The symbol by which the underscore (_) was added to the head of *variable_name* is outputted to an assembly source file as a public symbol or a share symbol.

6.14.8 CSGLOBAL pseudo-instruction

- Syntax

CSGLOBAL *file_id attrib size line column "variable_name" hierarchy*

- Description

A CSGLOBAL pseudo-instruction defines the information about the static global variable defined in C source program. The *variable_name* is the name of the static global variable on C source program. The symbol by which the underscore (_) was added to the head of *variable_name* is outputted to an assembly source file as a public symbol or a share symbol.

6.14.9 CLOCAL pseudo-instruction

- Syntax

CLOCAL *attrib size line column offset block_id* "*variable_name*" *hierarchy*

- Description

A CLOCAL pseudo-instruction defines the information about the local variable which C source program defined. The *variable_name* is the name of the local variable on C source program.

- Supplement

The CLOCAL pseudo-instruction must be defined as CBLOCK between CBLOCKEND pseudo-instructions.

6.14.10 CSLOCAL pseudo-instruction

- Syntax

CSLOCAL *attrib size line column alias_no block_id* "variable_name" *hierarchy*

- Description

A CSLOCAL pseudo-instruction defines the information about the static local

variable which C source program defined. The *variable_name* is the name of the static local variable on C source program. The decimal number shown by alias_no is added behind "_$ST" at the symbol of the assembly source.

- Supplement

The CSLOCAL pseudo-instruction must be defined as CBLOCK between CBLOCKEND pseudo-instructions.

6.14.11 CSTRUCTTAG, CSTRUCTMEM pseudo-instruction

- Syntax

CSTRUCTTAG *file_id fn_id block_id su_id total_mem total_size line column* "*tag_name*"

CSTRUCTMEM *attrib size line column offset* "*member_name*" *hierarchy*

- Description

These pseudo-instructions define the information about the structure described on C source program.

A CSTRUCTTAG pseudo-instruction defines the tag of a structure , and a CSTRUCTMEM pseudo-instruction defines the structure member defined by the CSTRUCTTAG pseudo-instruction described just before.

- Supplement

These pseudo-instructions must be defined as CBLOCK between CBLOCKEND pseudo-instructions.

6.14.12 CUNIONTAG, CUNIONMEM pseudo-instruction

- Syntax

CUNIONTAG *file_id fn_id block_id su_id total_mem total_size line column* "*tag_name*"

CUNIONMEM *attrib size line column* "*member_name*" *hierarchy*

- Description

These pseudo-instructions define the information about the union described on C source program.

A CUNIONTAG pseudo-instruction defines the tag of a common object, and a CUNIONMEM pseudo-instruction defines the union member defined by the CUNIONTAG pseudo-instruction described just before.

- Supplement

These pseudo-instructions must be defined as CBLOCK between CBLOCKEND pseudo-instructions.

6.14.13 CENUMTAG, CENUMMEM pseudo-instruction

- Syntax

CENUMTAG *file_id fn_id block_id enum_id total_mem line column* "*tag_name*"

CENUMMEM *value line column* "*member_name*"

- Description

These pseudo-instructions define the information about the enumerator type described on C source program.

A CENUMTAG pseudo-instruction defines an enumerator tag, and a CENUMMEM pseudo-instruction defines the enumrator list of the CENUMTAG pseudo-instruction described just before.

- Supplement

These pseudo-instructions must be defined as CBLOCK between CBLOCKEND pseudo-instructions.

6.14.14 CTYPEDEF pseudo-instruction

- Syntax

CTYPEDEF *file_id fn_id block_id attrib line column* "*type_name*" *hierarchy*

- Description

A CTYPEDEF pseudo-instruction defines the information about the type defined by typedef in C source program.

- Supplement

The CTYPEDEF pseudo-instruction must be defined as CBLOCK between CBLOCKEND pseudo-instructions.

6.14.15 CENVINFO pseudo-instruction

- Syntax

CENVINFO *language* "*compiler_info*" "*work_directory*"

- Description

A CENVINFO pseudo-instruction defines the compile environment which generated the assembly source file.

language is a value showing the language of a source program, and, in the case of CCi3 compiler, is set to 1.

6.14.16 CMAINFO pseudo-instruction

- Syntax

CMACINFO *file_id line def_flag "macro_string"*

- Description

A CMAINFO pseudo-instruction defines the macroscopic information defined at the time of compile.

6.14.17 CINCPATH pseudo-instruction

- Syntax

CINCPATH *inc_id "include_path"*

- Description

A CINCPATH pseudo-instruction defines the include path information defined at the time of compile.

7 List file

In this chapter, the form and reading of a list file which RASi3 creates are explained. The list file consists of following lists.

1. Assembly list
   Assembly listing is a list of the object code corresponding to a program.
2. Cross reference list
   A cross reference list shows the line information by which each user symbol was defined, and the line information for which each user symbol was referred to.
3. Symbol list
   A symbol list is a list including the information about the user symbol used for the program.
4. End message
   An end message is the list which is outputted when an assemble is completed, and displays the number of an error or warning.

The following options or pseudo-instructions are used and the output of a list file can be controlled.

	Pseudo-instruction	Option
Whole of list file	PRN, NOPRN	/PR, /NPR
Assembly list	LIST, NOLIST	/L, /NL
Cross reference list	REF, NOREF	/R, /NR
Symbol list	SYM, NOSYM	/S, /NS

7.1 Reading of an assembly list

The example of an assembly list is shown below.

```
*****************************************************************************
RASi3 (Target) Relocatable Assembler Ver.x.xx Assembly list.Page    : #      <-- (1)
  Source File         :   filename       <-- (2)
  Object File         :   filename       <-- (3)
  Date                :   date day time  <-- (4)
  Title               :   title          <-- (5)

  Loc.   Object   Line    Source Statements
   (6)     (7)     (8)            (9)

 (10)

  Target              :Core Type / Machine Type <-- (11)
  Errors              :# <-- (12)
  Warnings            :# <-- (13)
  Lines               :# <-- (14)
*****************************************************************************
```

(1) This line is displayed at the head of each page of an assembling list. The target model name specified by the TYPE pseudo-instruction goes into *Target*, and the number of pages goes into #.

(2) An input source file name is displayed.

(3) An object file name is displayed.

(4) The date and time which started assembling are displayed.

(5) The character string specified by the TITLE pseudo-instruction is displayed. It becomes blank when there is no description of a TITLE pseudo-instruction.

(6) The location of the instruction statement of each segment is displayed.

(7) The 32-bit machine code to each instruction statement is displayed.

(8) The line number in a source file is displayed.

(9) The instruction statement described by the source file is displayed.

(10) When there is an error, the occurred error message is displayed under a sauce line.

(11) The target core specified by the CORE pseudo-instruction and the target model specified by the TYPE pseudo-instruction are displayed.

(12) The total number of error is displayed.

(13) The total number of warning is displayed.

(14) The total number of lines of an input source file is displayed.

(2),(3),(4) and (5) are displayed only on the page of the beginning of an assembling list.
(11),(12),(13) and (14) are displayed only on the page of the last of an assembling list.

7.2 Reading of a cross reference list

A cross reference list shows the list of the line numbers of the symbol which appeared in the program.

The example of a cross reference list is shown below.

```
*******************************************************************************
RASi3 (Target) Relocatable Assembler Ver.x.xx.xx C-Ref list.Page     : # <-- (1)

  symbol              lines ( #:definition line)
---------- ----------------------------
     (2)                 (3)
-------SCOPE Symbol----------
  symbol           scope_tag           lines ( #:definition line)
---------- --------------- ----------------------------
                          (4)
------Temporary Symbol-------
  symbol           def_line            lines ( #:definition line)
---------- --------------- ----------------------------
                          (5)
*******************************************************************************
```

(1) This line is displayed at the head of the page of a cross reference list. *Target* is the target model name specified by the TYPE pseudo-instruction, and page number goes into #.

(2) A symbol name is displayed on alphabetic order.

(3) The line number by which the symbol was referred to is displayed. # is attached to the defined line number.

(4) A scope tag name is displayed.

(5) The line number by which the temporary symbol was defined is displayed.

7.3 Reading of a symbol list

A symbol list shows the detailed information on the symbol which appeared in the program. The symbol list consists of a symbol information and a segment information.

The detailed information on all the symbols defined by the program and the SFR symbol referred to once or more is displayed on a symbol information.

The detailed information on the relocatable segment defined by the program is displayed on a segment information.

The example of a symbol list is shown below.

```
*******************************************************************************
RASi3 (Target) Relocatable Assembler Ver.x.xx.xx Symbol list.Page     : # <-- (1)

----- symbol information -----

  symbol            type      usgtyp       value       ID
------------- ------ --------- -------- ---
     (2)              (3)        (4)           (5)         (6)

----------SCOPE Symbol-----------
  symbol            type      usgtyp       value       scope_tag
------------- ------ --------- --------- -----------
                                                          (7)
---------Temporary Symbol--------
  symbol            type      usgtyp       value        def_line
------------- ------ --------- --------- -----------------
                                                           (8)
---------segment information--------
  S-ID   symbol          segtyp   size     bound     link_attribute
---- ----------- ------ ------ ----- --------------
    (9)     (10)            (11)    (12)       (13)          (14)
*******************************************************************************
```

(1) This line is displayed at the head of each page of a symbol list. *Target* is the target model name specified by the TYPE pseudo-instruction, and page number goes into #.

(2) A symbol name is displayed on alphabetic order.

(3) The type of a symbol is displayed.
The kind of type is shown below.
seg : segment symbol

mcr : macroscopic symbol
tmp : temporary symbol
scp : scope symbol
loc : other user symbols
def : DEFINE symbol
ext : external symbol
com : communal symbol
pub : public symbol
sfr : SFR symbol
*** : undefined symbol

(4) The usage type of a symbol is displayed.

(5) The value of symbol is displayed.
In the case of a macro symbol, the number of lines of the macro body is displayed on value.

(6) ID of a segment symbol, a communal symbol, and a external symbol is displayed.
In the case of a simple relocatable symbol, ID of the segment to which the symbol belongs is displayed.

(7) The tag name of a scope is displayed.

(8) The line number by which the temporary symbol was defined is displayed.

(9) The order of a definition of a segment symbol is displayed.

(10) The name of a segment symbol is displayed.

(11) A segment type is displayed.

(12) Segment size is displayed by a hexadecimal number.

(13) An address boundary value is displayed.

(14) A linkage attribute is displayed. A linkage attribute is the segment name assigned to the same address different segment type by the SEGMENT pseudo-instruction.

A usage type is expressed in the usgtyp field as the following keywords.

Display	Explanation
NUMBER	Usage type NUMBER
CODE	Usage type CODE
PRAM	Usage type PRAM
XRAM	Usage type XRAM
XROM	Usage type XROM
YRAM	Usage type YRAM
YROM	Usage type YROM
FLOAT	Usage type FLOAT

8 A message and end code

RASi3 reports the error about assembling processing. There is the following kind of the reports of an error.

1. An error message is outputted to a screen or an error file.
2. The number of an error is outputted to a print file.

There is the following kind of the errors of RASi3.

1. Command line error
2. Grammar error
3. Warning
4. Fatal error
5. Internal processing error

A command line error is an error occurred when an error is in description of a starting command line. RASi3 is terminated after outputting a command line error, without performing assembling processing.

A grammar error is an error occurred on the analysis of a source file. RASi3 continues assembling processing, even if a grammar error occurs, but an object file does not generate it. Moreover, this file is deleted, when a grammar error occurs and the object file of a same name already exists.

Warning shows that a problem may be in a program. Even if warning occurs, assembling processing is continued, and a print file and an object file are created.

A fatal error is a fatal error to which RASi3 cannot continue assembling. If a fatal error occurs, RASi3 will suspend assembling processing.

An internal processing error is an error occurred when a certain fault is detected by the internal processing of RASi3. If an internal processing error occurs, RASi3 will suspend assembling processing.

Usually, these error messages are displayed on a screen. Please use the redirection feature of DOS to output an error message to a file. Moreover, please use a /E option or an ERR pseudo-instruction to output only the message of an assembling error and warning to a file.

8.1 Form of an error message

The form of the error message outputted to a screen or an error file is as follows.

- Syntax

filename(*line1*) : *line2* : *type*　*error code* : *error message*

Here, filename indicates a source file, line1 indicates the line number on a source file, line2 indicates the line number on an assemble list, error code indicates the kind of error, and error message indicates the contents of the error.

The kind of the following errors is displayed on type.

type	Explanation of an error
Error	It is shown that it is a grammar error.
Warning	It is shown that it is warning.

8.2 Error message list

The list of the error messages which RASi3 displays is shown below. The number of an error message is indicated on the left of an error message, and the details are described under the error message.

8.2.1 Fatal error message

Error number	An error message and explanation
000	insufficient memory The memory for continuing processing is lacking. The cause of this error can consider the shortage of an area of the virtual memory of Windows. When other applications which increase the maximum of virtual memory which increases the availability of a hard disk have started, please operate ending etc. and increase the availability of virtual memory. Moreover, when the /R option (or REF pseudo-instruction) is specified, please remove the specification. When this error still occurs in addition, please perform the measure which splits a program or reduces the number of symbols.
001	file not found : file_name The source file shown by file_name, an include file, or a DCL file is not found.
002	cannot open file : file_name

The file shown by file_name cannot be created.
file_name is an object file, a print file, or an error file. Please confirm whether the invalid character is used for specification of a name, or the directory not existing is not specified.

003 cannot close file : file_name
The file shown by file_name cannot be closed.
The cause considered most is shortage of disk storage capacity.

004 error(s) found in DCL file
In the DCL file, a certain one or more grammar errors were found.
In this case, since an assembling result is not secured, an assembler outputs this error message and ends processing. As long as the original DCL file which our company provides is used, this error does not occur.

005 file seek error
Seeking of a file cannot be performed.

006 too many INCLUDE nesting levels
The nesting level of an include file is over 8.

007 line number overflow 9,999,999
The number of lines of one source program (total including an include file) is over 9,999,999.

008 I/O error writing file
The writing to an object file is not made.

009 TYPE directive missing
Specification of TYPE (target device name) is not in a source program.
Or another instruction is described before a TYPE pseudo-instruction.

010 unclosed block comment
A block comment /* ... */has not closed.

011 illegal reading binary file:*message*
The contents of the ABL file are not right.
The contents of the error are displayed on message. When this error occurs, please confirm the following thing first.
- Has not the error (except for warning) occurred in the first assembling?
- Isn't the source file edited after the first assembling?
- Are various options and an include path in agreement by the first assembling and re-assembling?
- Has not the fatal error of those other than addressing relation occurred at the time of a link?

	- Is the /A option specified when linked? After confirming the above-mentioned contents, when an error occurs in addition, please contact me to our company.
012	reading file check sum error The checksum of the record of an ABL file is not right.
013	I/O error reading file An ABL file cannot be read.
016	source filename is not specified There is no specification of a source file at the time of RASi3 starting.
017	too many MACRO nesting levels There are too many nesting levels of a MACRO pseudo-instruction.
018	too less ENDM The ENDM pseudo-instruction to a MACRO pseudo-instruction is lacking.
020	same macro is expanding now The same broad view is developed in the broad view.
021	too many REPT nesting level There are too many nesting levels of a REPT pseudo-instruction.
022	too many IRP nesting level There are too many nesting levels of an IRP pseudo-instruction.
024	valid core name is I-CORE3 or ICORE3 Invalid CORE name is specified as #CORE of DCL file.
025	TYPE derective is redefined Two or more TYPE direcitves are specified.
026	SCOPE is not closed ENDC directive is not specified.
027	cannot specify before TYPE directive TYPE directive must be specified first.
028	bad memory range Invalid memory range is specified in DCL file.

8.2.2 Assembling error message

Error number	An error message and explanation
000	bad operand Description of an operand is wrong. Specification of addressing has mistaken or it is possible whether there to be or many operands are few. In a pseudo-instruction, it is possible that a format and

	description of each pseudo-instruction are not in agreement.
001	bad syntax It is the basic syntax mistake before recognizing an instruction.
004	bad character:c(XX) The character c (ASCII code XX) cannot be used by a program.
005	illegal escape sequence format
006	illegal integer constant Description of the number of settings or an address constant is wrong.
007	unexpected EOL The character constant ('c') or the character string constant ("--") has not closed.
008	unexpected EOF The character constant ('c') or the character string constant ("--") has not closed.
009	illegal string constant A character string constant has an invalid description.
010	string constant too long The number of characters of a character string constant is over 256 characters.
011	illegal option:option option is not accepted as an option. This specification is ignored.
012	constant required The number of settings needs to be specified for the operand or option of an instruction.
013	declaration duplicated The same pseudo-instruction or the same option is specified 2 times or more.
014	location out of range The location has exceeded the stipulated range. This error takes place, when AT address of segment start specification (CSEG pseudo-instruction etc.) or the start address of an ORG pseudo-instruction is over the maximum or minimum of segment restrictions, or when the location updated by an instruction and pseudo-instruction exceeds the maximum of a segment.
018	segment type / usage type mismatch The segment type or you sage type which an instruction requires, and the specified type are not in agreement. This error is occurred when as follows. - The you sage type of the address of segment start specification

is not in agreement with a current segment type.
- The you sage type of the operand of a symbol definition pseudo-instruction is not in agreement with the type of an instruction.
- DS pseudo-instruction is described in a bit system segment.
- The DBIT pseudo-instruction is described in a byte system segment.
- DB pseudo-instruction and DW pseudo-instruction are described to segments other than CODE, XRAM, XROM, YRAM, and YROM.

019 undefined symbol:symbol
symbol is not defined.

020 segment symbol required
A segment symbol is required for the operand of a RELSEG pseudo-instruction.

021 forward reference not allowed
It is performing "refer to the front."
Many pseudo-instructions have not permitted "refer to the front" to an operand.
Moreover, you have to define the segment name specified as a RELSEG pseudo-instruction in addition to it.

023 symbol redefinition:symbol
symbol is already defined.

025 segment ID mismatch
When specifying a relocatable address as the operand of an ORG pseudo-instruction by a relocatable segment, the expression must indicate the address of a current segment.

026 address not allowed
When a current segment is an ANY type relocatable segment, if the address of an ORG pseudo-instruction is not an integer type (NUMBER), it will not become.

028 local symbol required:symbol
symbol which makes a public declaration must be defined as a local symbol.

029 out of range:message
The value of an operand is over the range of regular.
The name of a concrete area is displayed on message.

031 illegal relocation type
Boundary value specification of a SEGMENT pseudo-instruction or a COMM pseudo-instruction or specification of a special area attribute is wrong.

033 entry overflow
The number of a segment symbol, a share symbol, or external symbols is over 65535. Or there are too many areas added by the /B option.

034 string constant required
A character string constant is required for the operand of a TITLE pseudo-instruction. Or the operand form of C debugging pseudo-instruction is wrong.

035 absolute expression required
If an operand is not constant expression, it will not become.
This error is occurred when neither the operand of many pseudo-instructions nor the shift width of a rotate shift instruction is constant values.

036 simple relocatable expression required
If the operand of symbol definition pseudo-instructions (EQU pseudo-instruction etc.) or an ORG pseudo-instruction is not constant expression or a simple relocatable type, it will not become.

037 expression is unresolved
It is calculating further to unsolved operation.
Or the expression which includes unsolved operation in the operand of symbol definition pseudo-instructions (EQU pseudo-instruction etc.) or an ORG pseudo-instruction is specified.

038 illegal expression format
It is the basic syntax mistake of an expression.
For example, the case where the balance of a parenthesis is not correct etc. corresponds to this.

039 invalid relocatable expression
Operation which is not allowed is performed to the relocatable symbol.

040 divide by zero
The division or modulo 算 by 0 is performed.

044 illegal core name
The core-based-CPU name of #CORE sentence of a DCL file is wrong.

048 #ENDCASE does not have a matching #CASE
There is no #CASE sentence which hangs with #ENDCASE sentence and suits in a DCL file.

051 CODE segment only
It is the instruction or pseudo-instruction which can be described

only to a CODE segment.

055 LABEL or NAME format error

The syntax relation of an instruction, a symbol, or a label is wrong.

For example, when as follows, this error occurs.

```
LABEL:  EQU  100H
```

057 invalid initialization directive

The description position of an assembler initial-setting pseudo-instruction is not right.

This error is occurred when the instruction which cannot be specified before an assembler initial-setting pseudo-instruction is described.

058 illegal SFR word/byte attribute

The format of the WORD / byte access attribute field in the SFR access attribute definition sentence of a DCL file is wrong.

059 illegal SFR bit attribute

The format of the bit access attribute field in the SFR access attribute definition sentence of a DCL file is wrong.

060 out of SFR address range

The SFR address in the SFR access attribute definition sentence of a DCL file is not contained in the range of the SFR area defined by the SFR keyword.

061 misplaced ENDIF directive

There is no condition assembling start pseudo-instruction (IF, IFDEF, IFNDEF) corresponding to an ENDIF pseudo-instruction.

062 misplaced ELSE directive

There is no condition assembling start pseudo-instruction (IF, ENDIF, IFNDEF) corresponding to an ELSE pseudo-instruction.

063 unexpected end of file in conditional directive

There is no ENDIF pseudo-instruction corresponding to a condition assembling start pseudo-instruction (IF, IFDEF, IFNDEF). This error is always occurred to the line in the end of a program.

064 too many conditional directive nesting levels

The nesting level of a condition assembling instruction is over 15.

065 too many macro nesting level

The level of nesting of a DEFINE broad view is over 8.

066 illegal relocation type combine

071 label or '$' is not allowed

At the time of branch optimization option specification, the label

and current location symbol which were defined within the CODE segment become the same treatment as a front reference symbol. Therefore, operands which do not allow "refer to the front", such as an operand of a CODESEG pseudo-instruction or an EQU pseudo-instruction, cannot describe.

072 invalid SHORT/LONG

Description of SHORT and a LONG addressing child is wrong.

073 usage type NUMBER expected

The numerical value type expression is demanded. When it describes as follows, this error occurs.

074 cannot write to ROM

It is going to execute the write-in instruction to a ROM area.

075 invalid fn_id

In C debugging information pseudo-instruction, the value of a fn_id operand is not right.

076 invalid block_id

In C debugging information pseudo-instruction, the value of a block_id operand is not right.

077 cfunction cannot nest

It is trying to nest a CFUNCTION pseudo-instruction. For example, when the case where a CFUNCTIONEND pseudo-instruction is deleted by mistake differs from fn_id, this error occurs.

078 invalid position

The description position of C debugging information pseudo-instruction is wrong. It occurs, when C debugging information instruction which should correspond before this pseudo-instruction does not exist.

079 overlay location out of range

The real arrangement address of the CODE segment which uses an overlay feature has pointed out the address in which ROM does not exist.

080 illegal range

The specified area is wrong. This error is occurred when an area which is different from the existing area when the range which cannot be specified as a /B option is specified and a start address is a larger address than an end address is specified.

082 missing member directives for previous CxxxTAG directive

To a CSTRUCTTAG pseudo-instruction, when a CSTRUCTMEM pseudo-instruction is insufficient, when a CENUMMEM pseudo-instruction is insufficient, this error occurs to a

CENUMTAG pseudo-instruction.

083 unclosed CBLOCK directive exist

Correspondence of a CBLOCK pseudo-instruction and a CBLOCKEND pseudo-instruction cannot be taken. When the end of the case where a CFUNCTIONEND pseudo-instruction appears with no CBLOCKEND pseudo-instruction corresponding to a CBLOCK pseudo-instruction, or sauce is reached, this error occurs.

084 unclosed CFUNCTION directive exist

085 segment address mismatch

The segment is changing before the CFUNCTIONEND pseudo-instruction which corresponds from a CFUNCTION pseudo-instruction. RASi3 cannot output the right C debugging information.

086 It is the instruction which cannot be described immediately after the instruction in front of bad mnemonic for foward mnemonic.

087 cna't put near DO end address

It is the instruction which cannot be described before and after DO instruction and an address.

088 DO end address is specified at wrong place

The description position of DO instruction and an address is inaccurate.

089 same DO end address exist

It is the same and the address is specified.

090 missing operator LONG

There is no LONG operator.

091 bad expression

Description of an expression is not right.

092 can not scope nesting

The nest of a SCOPE pseudo-instruction cannot be performed.

093 not exist SCOPE directive

A SCOPE pseudo-instruction does not exist.

094 not exist MACRO directive

A MACRO pseudo-instruction does not exist.

095 MACRO symbol or SCOPE tag is used in operand

Neither a MACRO symbol nor a SCOPE tag can be used for the operand of an instruction.

096 bad core name

A core name is inaccurate.

097 only foward reference symbol is allowed

Only a front reference symbol is available.

098 CODE segment and ROM area only
It is available only to a CODE segment and a ROM area.

099 not exist SEGMENT directive
A SEGMENT pseudo-instruction does not exist.

100 Bad parameter

101 Only can use head part of Macro body
It can describe only into a macroscopic head part.

102 too many define nesting levels
There are too many nesting levels of DEFINE.

103 illegal float constant
It is an inaccurate floating decimal constant.

104 out of memory range
It is outside the memory range.

105 stack access continuing
Access to a stack is continuing.

106 label in expression or '$' is not allowed
The expression or $ containing a label cannot be described.

107 can't use imm_9 when .cc is specified
When a condition specification child is specified, a 9-bit immediate value cannot be specified.

108 CODE label forward reference not allowed
The CODE label of referring to the front cannot be described.

109 Bad Linkage Attribute
A linkage attribute is inaccurate.

110 This linkage attribute is already used
It is the already specified linkage attribute.

111 Segment name for linkage doesn't match
The segment specified with the linkage attribute is not in agreement.

112 CDB directive parameter error
The parameter of C debugging information pseudo-instruction is inaccurate.

113 can't use imm_16 when .cc is specified
When a condition specification child is specified, a 16-bit immediate value cannot be specified.

114 FLOAT value is not allowed
A FLOAT type value cannot be specified.

115 illegal initialization
It is unjust initialization.

116 smaller than start address

	The value smaller than the start address of a segment is specified.
117	bad operand separator The separator of operand is illegal.
118	Same destination register is used in parallel instruction The same destination register is used by the parallel instruction.
119	Same address register is used in MOVXY The same address register is used by the MOVXY instruction.
120	Same destination register is used in MOVXY The same destination register is used by the MOVXY instruction.
121	page of end address is different It is specified as the page address where the end addresses of DO instruction differ.
122	loop end address cannot be specified before current location The end address is specified in front of DO instruction.
123	forward mnemonic can't put near DO end address The forward instruction cannot be specified near end address of DO instruction.
124	RAM area only It is a description effective only in RAM area.
125	Base register is same as destination register in parallel instruction Cannot specify same Destination register as Base register in parallel instruction.

8.2.3 Warning message

Warning number	An error message and explanation
002	option directive duplicated The pseudo-instruction and option which were already specified are specified again. This specification is ignored.
003	CPU type mismatch
005	address expression required An address type is required. This warning is occurred when a numerical type is specified as the right-hand side of an OFFSET operator.
006	NUMBER expression required A numerical type is required. This warning is occurred when an address is specified as the right-hand side of BYTE1, BYTE2, BYTE3, BYTE4, WORD1, and WORD2 operator, and addressing which allows only a numerical

value type.

007 FLOAT expression required
Decimal point form is required.

008 segment address mismatch
In the operation of addresses, the segment address of the left side and the right-hand side is not in agreement.

009 address attribute not inherited
An expression is managed as a numerical value. The attribute as an address is lost.

011 right expression of operator must be NUMBER
If the right-hand side of a operator is not a numerical value type, it will not become.

012 usage type mismatch

013 either right or left expression of operator must be NUMBER
If either the left side of a operator or the right-hand side is not a numerical value type, it will not become.

025 illegal access to SFR
Access to a SFR area is invalid.
This warning is occurred when it writes in to write-protected SFR.

026 cannot access to high byte of SFR word

028 cannot access to high byte

031 reference before first definition

038 current location was aligned

039 address out of range
The target memory does not exist in the address specified by the operand of an instruction. This warning is occurred, when the address in which an object memory does not exist by symbol definition pseudo-instruction was specified, or when an area has forgotten to be secured as a /B option.

041 error of address register restrict
It is warning to use restrictions of an address register.

042 address expressoin not allowed
An address type cannot be described.

043 this operator has mean for NUMBER
It is the operator which is meaningful only to NUMBER.

044 this operator has mean for FLOAT
It is the operator which is meaningful only to FLOAT.

045 minus number is not allowed
A negative value cannot be described.

046 float value over flow

	It is overflow of a floating decimal.
047	segment type mismatch
	A segment type is inharmonious.
048	extension is ignored
049	.CC is ignored
	A condition specification child is ignored.
050	CODE label with forward reference in expression
	The CODE label of referring to the front is contained.
051	SHORT operator can be specified
	A SHORT operator can be specified.
052	Addressing size is not SHORT
	Addressing size is not short size.
053	/T option is prioritized
	Priority is given to a /T option.
054	out of range
	The operand of DW directive is out of range.
	Valid range is 0H - 0FFFFH.
055	address range attribute is ignored
	The address range attribute is ignored when usage type is NUMBER or FLOAT.
056	same parameter specified
	Same symbol is specified as parameter of MACRO directive.
057	Only can use head part of Macro body
	LOCAL directive only can use head part of Macro body.
058	SHORT operator is ignored
	SHORT operater has no mean for this instruction.
061	illegal number. decimal constant required
	The argument of /W option must be specified as decimal constant.

8.3 End code

RASi3 returns the value according to an end state at the time of an assembling end. This value is called an end code. An end code can be inspected using a batch file etc. There is the following in an end code.

End code	Explanation
0	There is no error.
1	Warning occurred.
2	The assembling error occurred.

3	The fatal error and internal processing error or the DCL error occurred and forced to terminate.

RASi3 user's manual

August, 2004 preliminary edition