NEC

Application Note

Guide to Porting from CC78K4 to CA850

Target Devices

V850 Family™

Target Tools

CA850 Ver. 2.40
CC78K4 Ver. 2.20
RA78K4 Ver. 1.30

Document No. U15653EJ1V0AN00 (1st edition)
Date Published September 2001 N CP(K)

[MEMO]

- **The information in this document is current as of July, 2001. The information is subject to change without notice. For actual design-in, refer to the latest publications of NEC's data sheets or data books, etc., for the most up-to-date specifications of NEC semiconductor products. Not all products and/or types are available in every country. Please check with an NEC sales representative for availability and additional information.**
- No part of this document may be copied or reproduced in any form or by any means without prior written consent of NEC. NEC assumes no responsibility for any errors that may appear in this document.
- NEC does not assume any liability for infringement of patents, copyrights or other intellectual property rights of third parties by or arising from the use of NEC semiconductor products listed in this document or any other liability arising from the use of such products. No license, express, implied or otherwise, is granted under any patents, copyrights or other intellectual property rights of NEC or others.
- Descriptions of circuits, software and other related information in this document are provided for illustrative purposes in semiconductor product operation and application examples. The incorporation of these circuits, software and information in the design of customer's equipment shall be done under the full responsibility of customer. NEC assumes no responsibility for any losses incurred by customers or third parties arising from the use of these circuits, software and information.
- While NEC endeavours to enhance the quality, reliability and safety of NEC semiconductor products, customers agree and acknowledge that the possibility of defects thereof cannot be eliminated entirely. To minimize risks of damage to property or injury (including death) to persons arising from defects in NEC semiconductor products, customers must incorporate sufficient safety measures in their design, such as redundancy, fire-containment, and anti-failure features.
- NEC semiconductor products are classified into the following three quality grades:
  "Standard", "Special" and "Specific". The "Specific" quality grade applies only to semiconductor products developed based on a customer-designated "quality assurance program" for a specific application. The recommended applications of a semiconductor product depend on its quality grade, as indicated below. Customers must check the quality grade of each semiconductor product before using it in a particular application.

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

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

  "Specific": Aircraft, aerospace equipment, submersible repeaters, nuclear reactor control systems, life support systems and medical equipment for life support, etc.

  The quality grade of NEC semiconductor products is "Standard" unless otherwise expressly specified in NEC's data sheets or data books, etc. If customers wish to use NEC semiconductor products in applications not intended by NEC, they must contact an NEC sales representative in advance to determine NEC's willingness to support a given application.

  (Note)

  (1) "NEC" as used in this statement means NEC Corporation and also includes its majority-owned subsidiaries.

  (2) "NEC semiconductor products" means any semiconductor product developed or manufactured by or for NEC (as defined above).

M8E 00.4

Regional Information

Some information contained in this document may vary from country to country. Before using any NEC product in your application, please contact the NEC office in your country to obtain a list of authorized representatives and distributors. They will verify:

- Device availability
- Ordering information
- Product release schedule
- Availability of related technical literature
- Development environment specifications (for example, specifications for third-party tools and components, host computers, power plugs, AC supply voltages, and so forth)
- Network requirements

In addition, trademarks, registered trademarks, export restrictions, and other legal issues may also vary from country to country.

NEC Electronics Inc. (U.S.)
Santa Clara, California
Tel: 408-588-6000
800-366-9782
Fax: 408-588-6130
800-729-9288

NEC Electronics (Germany) GmbH
Duesseldorf, Germany
Tel: 0211-65 03 02
Fax: 0211-65 03 490

NEC Electronics (UK) Ltd.
Milton Keynes, UK
Tel: 01908-691-133
Fax: 01908-670-290

NEC Electronics Italiana s.r.l.
Milano, Italy
Tel: 02-66 75 41
Fax: 02-66 75 42 99

NEC Electronics (Germany) GmbH
Benelux Office
Eindhoven, The Netherlands
Tel: 040-2445845
Fax: 040-2444580

NEC Electronics (France) S.A.
Velizy-Villacoublay, France
Tel: 01-3067-5800
Fax: 01-3067-5899

NEC Electronics (France) S.A.
Madrid Office
Madrid, Spain
Tel: 091-504-2787
Fax: 091-504-2860

NEC Electronics (Germany) GmbH
Scandinavia Office
Taeby, Sweden
Tel: 08-63 80 820
Fax: 08-63 80 388

NEC Electronics Hong Kong Ltd.
Hong Kong
Tel: 2886-9318
Fax: 2886-9022/9044

NEC Electronics Hong Kong Ltd.
Seoul Branch
Seoul, Korea
Tel: 02-528-0303
Fax: 02-528-4411

NEC Electronics Singapore Pte. Ltd.
Novena Square, Singapore
Tel: 253-8311
Fax: 250-3583

NEC Electronics Taiwan Ltd.
Taipei, Taiwan
Tel: 02-2719-2377
Fax: 02-2719-5951

NEC do Brasil S.A.
Electron Devices Division
Guarulhos-SP, Brasil
Tel: 11-6462-6810
Fax: 11-6462-6829

J01.2

INTRODUCTION

Target Readers

This Application Note is intended for users who understand the functions of the 78K/IV Series microcontrollers, V800 Series™ microcontrollers, 78K/IV Series CC78K4 C compiler (CC78K4), and RA78K4 assembler package (RA78K4).

Purpose

This Application Note explains the points to be noted and basic method of description when replacing a program described for the CC78K4 by the V850 Family CA850 C compiler package (CA850).

Organization

This Application Note consists of the following chapters.

CHAPTER 1 OVERVIEW

Compares the CC78K4, RA78K4, and CA850.

CHAPTER 2 C LANGUAGE

Explains how to replace compiler-dependent description in C language from CC78K4 to CA850.

CHAPTER 3 ASSEMBLY LANGUAGE

Explains how to replace assembler control instructions and quasi-directives from RA78K4 to CA850.

CHAPTER 4 LINK DIRECTIVES

Explains how to replace link directives from CC78K4 to CA850.

CHAPTER 5 TRANSLATION LIMIT

Explains the maximum performance of the CC78K4, RA78K4, and CA850.

This Application Note does not explain instructions. For details of instructions, refer to the manual of each microcontroller.

How to Read This Manual

Read this Application Note from **CHAPTER 1 OVERVIEW**. If your program is not described in C language, you may skip **CHAPTER 2 C LANGUAGE**. If your program is not described in assembler language, you may skip **CHAPTER 3 ASSEMBLY LANGAUGE**. To change a link directive file, read **CHAPTER 4 LINK DIRECTIVES**.

Conversions

[ ]:	May be omitted.
“ ”:	Reference destination
[CC78K4]:	Function and description format in CC78K4
[RA78K4]:	Function and description format in RA78K4
[CA850]:	Function and description format in CA850
<Example>:	Description example

Related Documents When using this Application Note, also refer to the following related documents. The related documents indicated in this publication may include preliminary versions. However, preliminary versions are not marked as such.

Documents Related to 78K/IV Series (User's Manuals)

Document Name		Document No.
IE-78K4-NS		U13356E
RA78K4 Assembler Package	Operation	U11334E
	Language	U11162E
	Structured Assembler Preprocessor	U11743E
CC78K4 C Compiler	Operation	U11572E
	Language	U11571E
SM78K4 System Simulator Ver. 1.40 or Later Windows™ Based	Reference	U10093E
SM78K Series System Simulator Ver. 1.40 or Later	External Part User Open Interface Specifications	U10092E
ID78K Series Integrated Debugger Ver. 2.30 or Later Windows Based	Operation	U15185E
ID78K4 Integrated Debugger Windows Based	Reference	U10440E
RX78K4 Real-Time OS	Basics	U10603E
	Installation	U10604E

Documents Related to V850 Family (User's Manuals) (Hardware Tools)

Document Name	Document No.
IE-703002-MC (In-Circuit Emulator for V851™, V852™, V853™, V854™, V850/SA1™, V850/SB1™, V850/SB2™, V850/SV1™, V850/SF1™)	U11595E
IE-703003-MC-EM1 (In-Circuit Emulator Option Board for V853)	U11596E
IE-703008-MC-EM1 (In-Circuit Emulator Option Board for V854)	U12420E
IE-703017-MC-EM1 (In-Circuit Emulator Option Board for V850/SA1)	U12898E
IE-703037-MC-EM1 (In-Circuit Emulator Option Board for V850/SB1, V850/SB2)	U14151E
IE-703040-MC-EM1 (In-Circuit Emulator Option Board for V850/SV1)	U14337E
IE-703079-MC-EM1 (In-Circuit Emulator Option Board for V850/SF1)	U15447E
IE-703102-MC (In-Circuit Emulator for V850E/MS1™, V850E/MS2™)	U13875E
IE-703102-MC-EM1 (In-Circuit Emulator Option Board for V850E/MS1, V850E/MS2) IE-703102-MC-EM1-A (In-Circuit Emulator Option Board for V850E/MS1)	U13876E
IE-V850E-MC (In-Circuit Emulator for V850E/IA1™, V850E/IA2™) IE-V850E-MC-A (In-Circuit Emulator for V850E1 (NB85E Core), V850E/MA1™, V850E/MA2™)	U14487E
IE-V850E-MC-EM1-A (In-Circuit Emulator Option Board for V850E1 (NB85E Core))	To be prepared
IE-V850E-MC-EM1-B, IE-V850E-MC-MM2 (In-Circuit Emulator Option Board for V850E1 (NB85E Core))	U14482E
IE-703107-MC-EM1 (In-Circuit Emulator Option Board for V850E/MA1, V850E/MA2)	U14481E
IE-703116-MC-EM1 (In-Circuit Emulator Option Board for V850E/IA1)	U14700E
IE-703114-MC-EM1 (In-Circuit Emulator Option Board for V850E/IA2)	To be prepared

Documents Related to V850 Family (User's Manuals) (Software Tools)

Document Name		Document No.
CA850 C Compiler Package Ver. 2.40 or Later	Operation	U15024E
	C Language	U15025E
	Project Manager	U15026E
	Assembly Language	U15027E
ID850 Integrated Debugger Ver. 2.40 Windows Based	Operation	To be prepared
ID850NW Integrated Debugger Ver. 1.10 or Later Windows Based	Operation	U14891E
SM850 System Simulator Ver. 2.40 Windows Based	Operation	To be prepared
SM850 System Simulator Ver. 2.00 or Later	External Part User Open Interface Specifications	U14873E
RX850 Real-Time OS Ver. 3.13 or Later	Basics	U13430E
	Installation	U13410E
	Technical	U13431E
RX850 Pro Real-Time OS Ver. 3.13	Basics	U13773E
	Installation	U13774E
	Technical	U13772E
RD850 Task Debugger Ver. 3.01		U13737E
RD850 Pro Task Debugger Ver. 3.01		U13916E
AZ850 System Performance Analyzer Ver. 3.0		U14410E
PG-FP3 Flash Memory Programmer		U13502E
CA850 C Compiler Package Ver. 2.40 (Application Note)	Coding Technique	U15184E
V800 Series Development Tools (32 bits) Ver. 2.40 Windows Based (Application Note)	Tutorial Guide	U15196E
Guide to Porting from CC78K4 to CA850 (Application Note)		This manual

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1 OVERVIEW

This chapter gives an outline of the CC78K4, RA78K4, and CA850.

1.1 Product Form

The C compiler, CC78K4, and assembler and linker, RA78K4, are separated for the 78K/IV Series. In contrast, the C compiler, assembler, and linker for the V850 Family are combined into one package, the CA850.

Table 1-1. Product Name

	78K/IV Series	V850 Family
C compiler package	USxxxxCC78K4	USxxxxCA703000
Assembler package	USxxxxRA78K4	

Note, however, that the device file for both the 78K/IV Series and V850 Family must be purchased separately.

1.2 Package Software

The CC78K4, RA78K4, and CA850 include the following programs.

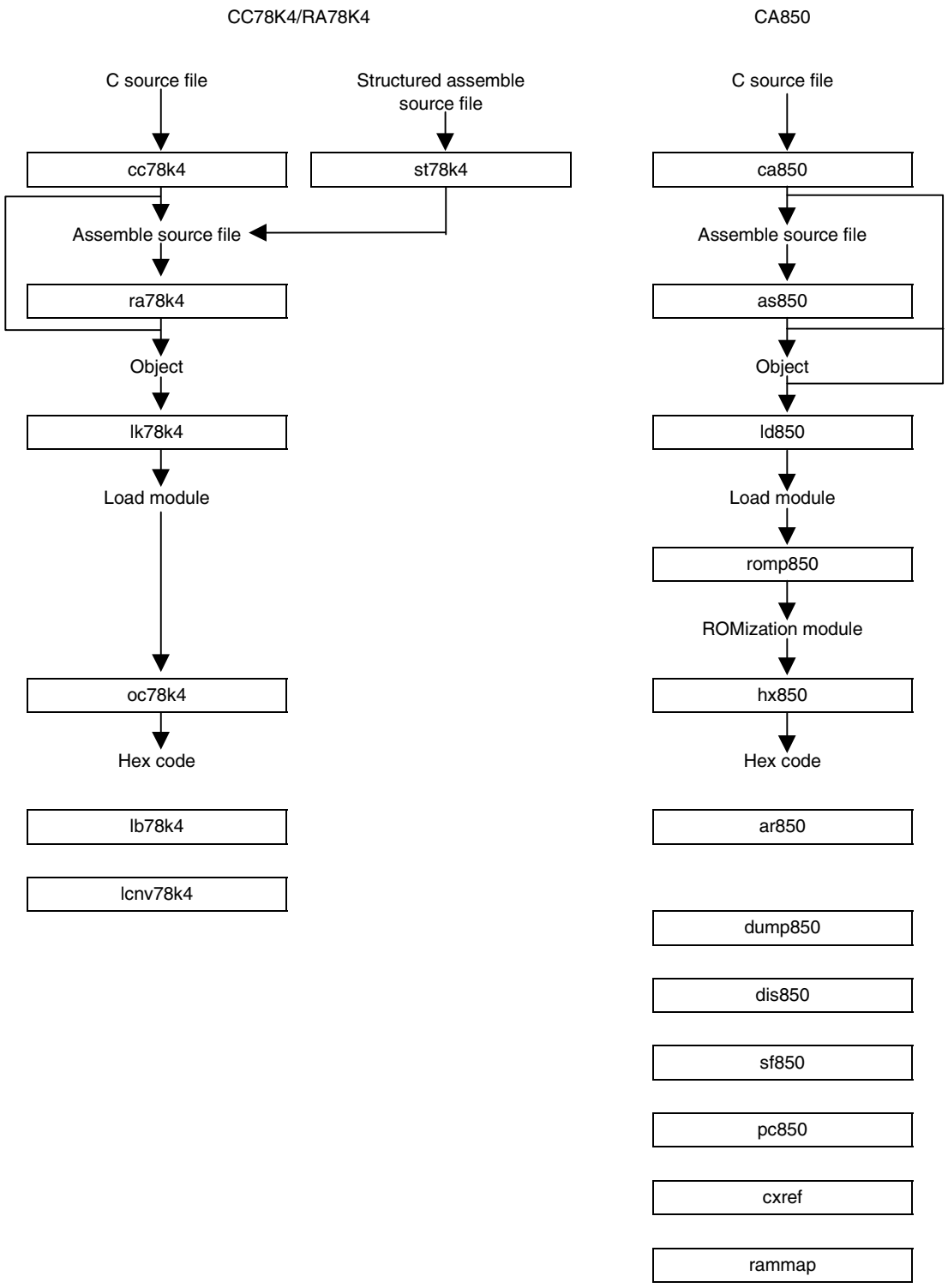
Table 1-2. Package Software

Name[Note]	CC78K4, RA78K4	CA850
C compiler	cc78k4	ca850
Assembler	ra78k4	as850
Linker	lk78k4	ld850
ROMization processor	–	romp850
Object converter Hex converter	oc78k4	hx850
Librarian Archiver	lb78k4	ar850
Structured assembler	st78k4	–
List converter	lcnv78k4	–
Dump directive	–	dump850
Disassembler	–	dis850
Section file generator	–	sf850
Performance checker	–	pc850
Cross-reference tool	–	cxref
Memory layout visualization tool	–	rammap

Note When two names are given, the name on the top is the one used in the RA78K4 and the name on the bottom is the one used in the CA850.

The development procedure of each program is as illustrated below.

Figure 1-1. Development Procedure



CHAPTER 2 C LANGUAGE

This chapter explains the points to be noted when rewriting a C source program for the CC78K4 into a program for the CA850, and how to describe the program.

The CC78K4 and CA850 conform to the ANSI-C[Note] Standard.

Note ANSI stands for American National Standards Institute

2.1 Compiler-Defined Macros

The following macros are defined in advance for the CC78K4 and CA850.

Table 2-1. Compiler-Defined Macros

CC78K4, RA78K4	CA850	Contents
__LINE__		Line number of current source
__FILE__		Source file name
__DATE__		Date of compiling source file (“mm dd yyyy”)
__TIME__		Time of compiling source file (“hh:mm:ss”)
__STDC__		1 in decimal numbers (CC78K4: Defined when -ZA option is specified CA850: Defined when -ansi option is specified)
__K4__	__v850 __v850__	1 in decimal numbers (macro indicating series name of devices)
CPU macro		1 in decimal numbers (macro indicating CPU of target) Examples of μPD784038 and μPD703003 are shown.
__4038_	__3003__	

By using these compiler-defined macros, descriptions for the CC78K4 and those for the CA850 can exist together in a C source program.

<Example>

```
#ifdef __K4__
        #pragma sfr               /* Description for CC78K4 */
#endif
#ifdef __v850__
        # pragma ioreg            /* Description for CA850 */
#endif
```

2.2 #pragma Directive

The #pragma directives instruct the compiler to run using the method defined by the compiler. The CC78K4 and CA850 have the following #pragma directives.

Table 2-2. #pragma Directives

No.	Function	CC78K4	CA850
1	Use of special function register name	#pragma sfr	#pragma ioreg
2	Description of assembler instruction	#pragma asm	#pragma asm #pragma endasm
3	Interrupt function	#pragma vect #pragma interrupt	#pragma interrupt
4	Specification of interrupt disabled function	Interrupts are disabled before preprocessing if DI(); is described at the beginning of a function[Note].	#pragma block_interrupt
5	Control of interrupt disabling (DI/EI)	#pragma di #pragma ei	#pragma directive is not necessary.
6	CPU control instructions	#pragma halt #pragma stop #pragma nop #pragma brk	#pragma directive is not provided. Description using assembler instruction
7	Absolute address access	#pragma access	None
8	Change of section name	#pragma section	#pragma section #pragma text
9	Change of module name	#pragma name	None
10	Specification of inline expansion	None	#pragma inline
11	Use of rotate function	#pragma rot	None
12	Use of multiplication function	#pragma mul	None
13	Use of division function	#pragma div	None
14	Use of data insertion function	#pragma opc	#pragma directive is not provided. Description by assembler instruction
15	Specification of interrupt handler supporting real-time OS	#pragma rtos_interrupt	#pragma directive is not provided. Description by assembler instruction
16	Specification of real-time OS function	#pragma rtos_task	#pragma rtos_task
17	Specification of device type	#pragma pc	#pragma cpu
18	Structure packing	None	#pragma pack

Note To describe DI();, the #pragma di directive is necessary.

2.2.1 Use of special function register name (peripheral function register name)

To rewrite a description for the CC78K4 into one for the CA850, change “sfr” to “ioreg”.

[CC78K4]

Using the #pragma directive, declare that a special function register name is used in the C source.

```
#pragma sfr
```

[CA850]

Using the #pragma directive, declare that a peripheral function register name is used in the C source.

```
#pragma ioreg
```

2.2.2 Description of assembler instruction

There are two ways to describe assembler instructions in the C source program, for both the CC78K4 and CA850.

[CC78K4]

(1) To insert one line

Declare use of _asm using the #pragma directive.

```
#pragma asm
```

Describe the assembler instruction in the C source in the following format.

```
_asm (character string literal);
```

<Example>

```
#pragma asm
void main(void)
{
      __asm("\tMOV\tA,B");
}
```

(2) To insert two or more lines

Indicate the start of the assembler instruction by #asm, and the end by #endasm.

Then, describe the assembler instruction between #asm and #endasm.

```
#asm
    assembler instruction
#endasm
```

<Example>

```
#asm
         MOV      A,B
#endasm
```

[CA850]

(1) To insert one line

Describe the assembler instruction in the C source in the following format.

__asm (character string constant);

or,

_asm (character string constant);

When specifying the –ansi option, however, use the __asm format.

<Example>

```
__asm("\tmov\tr10,r11");
```

(2) To insert two or more lines

Indicate the start of the assembler instruction by #pragma asm, and the end by #pragma endasm.

Then, describe the assembler instruction between #pragma asm and #pragma endasm.

#pragma asm

assembler instruction

#pragma endasm

<Example>

```
#pragma asm
        mov         r10,r11
#pragma endasm
```

2.2.3 Interrupt function

When defining an interrupt function using the #pragma directive, the address of the interrupt function is output to an interrupt vector (handler) in the case of the CC78K4, and a branch instruction to the interrupt function is output to an interrupt vector (handler) in the case of the CA850.

When replacing a description for the CC78K4 with one for the CA850, note the differences in interrupt request names (which differ depending on the microcontroller).

[CC78K4]

Use the #pragma directive to specify an interrupt request name, function name, stack selection, register, and saving/restoring the saddr2 area to be used.

#pragma vect interrupt request name function name [, stack selection specification]
stack use specification [{specification without change}] register bank specification

or,

#pragma interrupt interrupt request name function name [, stack selection specification]
stack use specification [{specification without change}] register bank specification

Two types of interrupt function qualifiers are available.

Non-maskable/maskable interrupt function

__interrupt function definition or function declaration

Software interrupt function

__interrupt_brk function definition or function declaration

<Example>

```
#pragma interrupt INTP0 int1
#pragma vect INTP1 int2 sp=buff+10 RB1
#pragma interrupt BRK_I int_b RB2

__interrupt
void int1(void)
{
        …
}
__interrupt
void int2(void)
{
        …
}
__interrupt_brk
void int_b(void)
{
        …
}
```

[CA850]

Specify an interrupt request name, function name, and placement method using the #pragma directive.

#pragma interrupt interrupt request name function name [placement method]

The interrupt function qualifier is as follows.

__interrupt function definition or function declaration

However, the qualifier is as follows for multiple interrupts.

__multi_interrupt function definition or function declaration

<Example>

```
#pragma interrupt INTP110 int1
#pragma interrupt INTP111 int2 direct
#pragma interrupt INTP112 int3

__interrupt
void int1(void)
{
        …
}
__interrupt
void int2(void)
{
        …
}
__multi_interrupt
void int3(void)
{
        …
}
```

2.2.4 Specification of interrupt disabled function

The entire function is disabled from being interrupted.

When porting from the CC78K4 to the CA850, specify the name of the function to be disabled from being interrupted using the #pragma directive, instead of using the interrupt functions DI() and EI().

[CC78K4]

Interrupts are disabled before preprocessing if DI() is described at the beginning of a function.

<Example>

```
#pragma DI
#pragma EI

void func1(void)
{
        DI();
        …
        EI();
}
```

[CA850]

Specify a function name using the #pragma directive.

#pragma block_interrupt function name

<Example>

```
#pragma block_interrupt func1
void func1(void)
{
        …
}
```

2.2.5 Control of interrupt disabling

Interrupt functions are provided in both the CC78K4 and CA850. To use these functions in the CC78K4, however, the #pragma directive is necessary. The directive is not necessary in the CA850.

[CC78K4]

Specify that DI() and EI() are used as interrupt functions using the #pragma directive.

```
#pragma di
#pragma ei
```

Describe the functions in the C source in the same manner as calling a function.

```
DI();
EI();
```

<Example>

```
#pragma DI
#pragma EI

void func2(void)
{
        …
        DI();
        …
        EI();
        …
}
```

[CA850]

A function to control interrupt disabling is provided. The #pragma directive is not necessary.

```
__DI();
__EI();
```

<Example>

```
void func2(void)
{
        …
        __DI();
        …
        __EI();
        …
}
```

2.2.6 CPU control instructions

The CC78K4 has CPU control instructions that control the CPU on the C source program (HALT, STOP, BRK, and NOP).

The CA850 does not have such instructions. Describe assembler instructions instead.

[CC78K4]

Using the #pragma directive, specify that the HALT(), STOP(), BRK(), and NOP() instructions are used as interrupt functions.

```
#pragma halt
#pragma stop
#pragma brk
#pragma nop
```

Describe the instructions in the C source in the same manner as calling a function.

```
HALT();
STOP();
BRK();
NOP();
```

<Example>

```
#pragma HALT
#pragma STOP
#pragma BRK
#pragma NOP

HALT();
STOP();
BRK();
NOP();
```

[CA850]

No CPU control instructions are available. Describe assembler instructions.

halt, trap, nop

<Example>

```
__asm("\thalt");
__asm("\ttrap\t0x00");
__asm("\tnop");
```

2.2.7 Absolute address access

The CC78K4 has a function for accessing absolute addresses.

The CA850 does not have such a function. Access absolute addresses using a pointer.

[CC78K4]

Using the #pragma directive, specify that a function for absolute address access is used.

```
#pragma access
```

Describe the function in the C source in the same manner as calling a function.

The names of the functions for accessing absolute addresses are the following four.

- peekb: Returns 1 byte of the contents of the address of the argument.
- peekw: Returns 2 bytes of the contents of the address of the argument.
- pokeb: Writes 1 byte to the address of the argument.
- pokew: Writes 2 bytes to the address of the argument.

<Example>

```
#pragma access

int      data1,data2;
char     cdata1,cdata2;

cdata1 = peekb ( 0x1234 );
data1 = peekw ( 0x1234 );
pokeb ( 0xfe20 , 5 );
pokew ( 0xfe20 , 0xffff );
```

[CA850]

No function for accessing absolute addresses is available. Use a pointer as shown below.

Also use the sample in Example 2 for reference.

<Example 1>

```
int      data1,data2;
char     cdata1,cdata2;
char     *p;
int      *q;

cdata1 = *((char *)0x1234);
data1 = *((int *)0x1234);

p = (char *)0xffe000;
*p = 5;
q = (int *)0xffe000;
*q = 0xffff;
```

<Example 2>

```
unsigned char peekb(unsigned int);
unsigned short peekh(unsigned int);
unsigned int peekw(unsigned int);

void pokeb(unsigned int,unsigned char);
void pokeh(unsigned int,unsigned short);
void pokew(unsigned int,unsigned int);

unsigned char peekb(unsigned int adr)
{
        return *((unsigned char *)adr);
}

unsigned short peekh(unsigned int adr)
{
        return *((unsigned short *)adr);
}

unsigned int peekw(unsigned int adr)
{
        return *((unsigned int *)adr);
}

void pokeb(unsigned int adr,unsigned char val)
{
unsigned char *p;

        p = (unsigned char *)adr;
        *p = val;
}

void pokeh(unsigned int adr,unsigned short val)
{
unsigned short *p;

        p = (unsigned short *)adr;
        *p = val;
}

void pokew(unsigned int adr,unsigned int val)
{
unsigned int *p;

        p = (unsigned int *)adr;
        *p = val;
}
```

2.2.8 Change of section name

Although section of the #pragma directive is the same, the character string to be specified after that differs.

[CC78K4]

Specify the section name to be changed, a new section name, and the start address of the section using the #pragma directive.

#pragma section compiler output section name section name to be changed [AT start address]

<Example>

```
#pragma section @@CODE CODESEG
        /* Changes segment name @@CODE to CODESEG */
#pragma section @@DATA DATASEG AT 0FFE10H
        /* Changes segment name @@DATA to DATASEG and allocates it to address 0FFE10H */
```

[CA850]

Using the #pragma directive, specify a section name for data or a module.

If no section name is specified, the default section name is assumed.

Data

#pragma section section type [“section name”] begin
declaration of variable/constant
#pragma section section type [“section name”] end

Module

#pragma text [“section name”] [function name]

Allocation can be specified for the following data section types.

.tidata (.tidata section, .tibss section)
.data (.data section, .bss section)
.sdata (.sdata section, .sbss section)
.sedata (.sedata section, .sebss section)
.sidata section (.sidata section, .sibss section)
.sconst section
.const section

A section name can be specified for the following section types.

.data (.data section, .bss section)
.sdata (.sdata section, .sbss section)
.const section
.sconst section

<Example>

```
#pragma section sconst begin
const int const_data=10;
#pragma section sconst end
        /* Allocates const_data variable to .sconst section */
#pragma section data "d1sec" begin
int data_data;
#pragma section data "d1sec" end
        /* Allocates data_data variable to dlsec.bss section of data attribute */

#pragma text "f1sec" func1
        /* Allocates func1 function to flsec.text section */
```

2.2.9 Change of module name

A module name cannot be changed in the CA850.

[CC78K4]

Specify a module name using the #pragma directive.

#pragma name module name

<Example>

```
#pragma name new_name
```

[CA850]

Not provided

2.2.10 Specification of inline expansion

With the CC78K4, a user-defined function cannot be expanded inline. With the CA850, inline expansion of a user function can be specified by the #pragma directive.

[CC78K4]

Not provided

[CA850]

Using the #pragma directive, specify inline expansion of each function.

#pragma inline function name [,function name…]

<Example>

```
#pragma inline func
```

2.2.11 Use of rotate function

The V850 Family does not have an instruction that rotates data. Therefore, the CA850 does not have a rotate function. To rotate data, create an assembler program by referring to the example below.

[CC78K4]

Using the #pragma directive, specify that a code to be rotated is directly expanded inline and output, instead of calling a function.

```
#pragma rot
```

Describe the function name in the C source in the same manner as calling a function.

The rotate function names are the following four.

rorb
rolb
rorw
rolw

<Example>

```
#pragma rot

ucdata3 = rorb(ucdata1,ucdata2);
ucdata3 = rolb(ucdata1,ucdata2);
uidata3 = rorw(uidata1,ucdata2);
uidata3 = rolw(uidata1,ucdata2);
```

[CA850]

No rotate function is available. To rotate a variable, create a program by referring to the sample below. However, inline expansion is not executed.

<Example>

Prototype declaration in C source

```
unsigned char rorb(unsigned char,unsigned char);
unsigned char rolb(unsigned char,unsigned char);
unsigned short rorh(unsigned short,unsigned char);
unsigned short rolh(unsigned short,unsigned char);
unsigned int rorw(unsigned int,unsigned char);
unsigned int rolw(unsigned int,unsigned char);
```

Assembler description (1/2)

```
        .globl  _rorb
        .globl  _rolb
        .globl  _rorh
        .globl  _rolh
        .globl  _rorw
        .globl  _rolw

        .file   "rot.s"
        .text
-- 1byte variable rotation --
        .align  4
_rorb:
        shr     1,r6
        bnc     rorb1
        or      0x00000080,r6
rorb1:
        add     -1,r7
        mov     r6,r10
        bnz     _rorb
        jmp[lp]

        .align  4
_rolb:
        add     -4,sp
rolb1:
        shl     1,r6
        st.h    r6,[sp]
        tst1    0,1[sp]
        bz      rolb2
        or      0x00000001,r6
rolb2:
        add     -1,r7
        mov     r6,r10
bnz     rolb1
add     4,sp
jmp[lp]


-- 2byte variable rotation --
        .align  4
_rorh:
                shr     1,r6
        bnc     rorh1
                or      0x00008000,r6
rorh1:
                add     -1,r7
        mov     r6,r10
                bnz     _rorh
        jmp[lp]

                .align  4
_rolh:
                add     -4,sp
```

Assembler description (2/2)

```
rolh1:
                shl       1,r6
        st.w    r6,[sp]
        tst1    0,2[sp]
                bz        rolh2
        or      0x00000001,r6
rolh2:
        add     -1,r7
        mov     r6,r10
        bnz     rolh1
                add       4,sp
        jmp[lp]


-- 4byte variable rotation --
        .align  4
_rorw:
                shr       1,r6
        bnc     rorw1
        or      0x80000000,r6
rorw1:
        add     -1,r7
        mov     r6,r10
        bnz     _rorw
        jmp[lp]

        .align  4
_rolw:
        shl     1,r6
        bnc     rolw1
        or      0x00000001,r6
rolw1:
        add     -1,r7
        mov     r6,r10
        bnz     _rolw
        jmp[lp]
```

2.2.12 Use of multiplication function

The CA850 does not have a multiplication function. Describe multiplication using the value of an expression.

[CC78K4]

Using the #pragma directive, specify that a code to be multiplied is directly expanded inline and output, instead of calling a function.

```
#pragma mul
```

Describe the function name in the C source in the same manner as calling a function.

The multiplication function names are the following three.

mulu
muluw
mulw

<Description>

```
#pragma mul
uidata3 = mulu(ucdata1,ucdata2);
uldata3 = muluw(uidata1,uidata2);
ldata3 = mulw(idata1,idata2);
```

[CA850]

No function for inline expansion is available.

Describe multiplication in the form of an expression (* operator), not in the form of a function.

Be careful of the size of the argument assigned as the argument of muluw or mulw because int type and long type are 4 bytes (int type must be short type).

2.2.13 Use of division function

The CA850 does not have a division function. Describe division using the value of an expression.

[CC78K4]

Using the #pragma directive, specify that a code to be divided is directly expanded inline and output, instead of calling a function.

```
#pragma div
```

Describe the function name in the C source in the same manner as calling a function.

The division function names are the following two.

```
divuw
moduw
```

<Example>

```
#pragma div
uidata3 = divuw(uidata1,ucdata2);
ucdata3 = moduw(uidata1,ucdata2);
```

[CA850]

No function for inline expansion is available.

Describe division in the form of an expression (/operator or %operator), not in the form of a function.

2.2.14 Use of data insertion function

The CA850 does not have a data insertion function. Describe data insertion using the assembler.

[CC78K4]

Using the #pragma directive, specify use of a function for data insertion.

```
#pragma opc
```

Describe the function in the C source in the same manner as calling a function.

```
__OPC (data value [, data value …]
```

<Example>

```
#pragma opc

__OPC(0xBF,0x12);
```

[CA850]

No function for data insertion is available. Describe data insertion using an assembler quasi-directives.

<Example>

```
__asm(".byte 0xBF,0x12");
```

2.2.15 Specification of interrupt handler supporting real-time OS

With the CA850, a directly activated interrupt handler cannot be described in C language. It must be described using the assembler.

[CC78K4]

Using the #pragma directive, specify an interrupt request name, function name, and stack selection.

#pragma rtos_interrupt interrupt function name function name [,stack selection specification]

The interrupt function qualifier is as follows.

Non-maskable/maskable interrupt function

__rtos_interrupt function definition or function declaration

<Example>

```
#pragma rtos_interrupt INTP3 int4 sp=buff+10
__rtos_interrupt
void int4(void)
{
        …
        ret_int();
}
```

[CA850]

No #pragma directive is available. The handler must be described using the assembler.

The RX850 and RX850 Pro have macros for directly activated interrupt handlers. Describe a directly activated interrupt handler as illustrated below.

The RX850 and RX850 Pro have more than one method of description, depending on the return processing. For details, refer to the user's manual of the real-time OS.

<Example>

Return by reti in RX850

```
#include "stdrx.inc"
    .section     " INTP123"
    jr  _inthdr
    .text
    .align 4
    .globl       _inthdr
_inthdr:
    RTOS_IntEntry
        …
    RTOS_IntExit
```

2.2.16 Specification of real-time OS function

The description format is the same in the CC78K4 and CA850.

[CC78K4]

Using the #pragma directive, the function name specified is interpreted as a task for the real-time OS.

```
#pragma rtos_task  task function name
```

[CA850]

Using the #pragma directive, the function name specified is interpreted as a task for the real-time OS.

```
#pragma rtos_task  task function name
```

There is no difference between the CC78K4 and CA850.

<Example>

```
#pragma rtos_task func4
void func4(void)
{
        …
        ext_tsk();
}
```

2.2.17 Specification of device type

To replace a description for the CC78K4 with one for the CA850, change “pc” to “cpu”.

When specifying a device, use a blank instead of parentheses.

[CC78K4]

Specify a device type using the #pragma directive.

```
#pragma pc (device type)
```

<Example>

```
#pragma pc(4038)
```

[CA850]

Specify a device type using the #pragma directive.

```
#pragma cpu  device type
```

<Example>

```
#pragma cpu 3003
```

2.2.18 Structure packing

The CC78K4 does not have a structure packing function. Structures are justified and an area is reserved.

[CC78K4]

Not provided

[CA850]

The value specified by the #pragma directive is used as the current packing value. The specified value is valid until the next #pragma pack directive appears.

#pragma pack (packing value)

<Example>

```
#pragma pack(1)
struct{
     char data1;
     short data2;
     char data3;
}
```

2.3 Extended Descriptions

Extended descriptions are used to realize functions peculiar to a device. The extended descriptions of the CC78K4 and CA850 are as follows.

Table 2-3. Extended Descriptions

No.	Function	CC78K4	CA850
1	callt function	callt / __callt	None
2	Register variable	register	register
3	Use of saddr area	sreg / __sreg / __sreg1	None
4	noauto function	noauto	None
5	norec function	norec	None
6	Bit type variable	bit/boolean / __boolean / __boolean1	Bit field
7	Bit access	Variable name. bit position	Union and bit field
8	callf function	callf / __callf	None
9	Binary constant	0bxxxxxxxx	0bxxxxxxxx
10	Interrupt level	None	__set_il
11	Pascal function	__pascal	None

2.3.1 callt function

The CA850 has a callt instruction only for the V850E. However, use an ordinary function in the CA850.

[CC78K4]

callt/__callt stores the address of a function to be called in an area called callt (0x40 to 0x7f) to make it possible to call a function with a code shorter than that used to directly call the function.

The callt instruction is used to call a function.

The description format is as follows.

callt type name function name

__callt type name function name

<Example>

```
callt void func(void)
{
        …
}
```

[CA850]

The V850E has a callt instruction, but the compiler uses the callt instruction only during the runtime of the prologue and epilogue of a function.

Therefore, use an ordinary function.

<Example>

```
void func(void)
{
        …
}
```

2.3.2 Register variables

A register variable is described in the same manner in both the CC78K4 and CA850, but a variable is allocated to a register differently. If optimization is specified in the CA850, even a variable declared by register may not be allocated to a register.

[CC78K4]

Variables declared by register are allocated to a register (RP3 or VP) or the saddr2 area.

When the -ZO option is specified, the variables are allocated in the sequence in which they were declared. If the -ZO option is not specified, they are allocated in the order of the number of times they were referenced.

The description format is as follows.

register type name variable name

[CA850]

Variables declared by register are allocated to registers for register variables in the sequence in which they were declared. If optimization is specified, however, a variable that has been referenced fewer times may not be allocated.

The description format is as follows.

register type name variable name

<Example>

```
int func(void)
{
register int reg_a,reg_b;
        …
}
```

2.3.3 Using saddr area

It is recommended to locate the saddr2 area of the 78K/IV Series to the .tidata/.tibss section of the CA850. The .tidata section can be accessed by a 2-byte instruction (sld/sst).

[CC78K4]

External variables declared by sreg or __sreg and static variables in a function are relocatable and automatically allocated to the saddr2 area.

Variables declared by __sreg1 are relocatable and automatically allocated to the saddr1 area.

The description format is as follows.

sreg type name variable name
__sreg type name variable name
__sreg1 type name variable name

<Example>

```
sreg intsreg_data0,sreg_data1,sreg_data2;
```

[CA850]

Not provided.

Using the #pragma section directive, allocate external variables to the .tidata/.tibss section of the internal RAM.

```
#pragma section tidata      begin
                    type name variable name
#pragma section tidata      end
```

<Example>

```
#pragma section tidata begin
int     sreg_data0,sreg_data1,sreg_data2;
#pragma section tidata end
```

2.3.4 noauto function

The CA850 does not have a noauto function. Use an ordinary function.

[CC78K4]

The noauto function restricts an automatic variable from outputting a code for pre- and post-processing (creation of stack frame).

The description format is as follows.

noauto type name function name

<Example>

```
noauto void func(void)
{
        …
}
```

[CA850]

No noauto function is available.

Use an ordinary function.

<Example>

```
void func(void)
{
        …
}
```

2.3.5 norec function

The CA850 does not have a norec function. Use an ordinary function.

[CC78K4]

A function that does not call an other function from itself can be used as a norec function.

The norec function does not output a code for pre- and post-processing of a function (creation of stack frame).

All arguments are allocated to the saddr2 area for register and norec function arguments.

The description format is as follows.

norec type name function name

<Example>

```
norec void func(void)
{
        …
}
```

[CA850]

No norec function is available.

Use an ordinary function.

<Example>

```
void func(void)
{
        …
}
```

2.3.6 Bit type variable

The CA850 cannot define bit type variables. Describe them by using a bit field.

[CC78K4]

bit, boolean, and __boolean type variables are treated as 1-bit data and allocated to the saddr2 area.

The __boolean1 type variable is treated as 1-bit data and allocated to the saddr1 area.

bit, boolean, __boolean, and __boolean1 type variables are treated in the same manner as an external variable without a default value (undefined).

The description format is as follows.

bit variable name

boolean variable name

__boolean variable name

__boolean1 variable name

<Example>

```
boolean bit1,bit2,bit3;
void func(void)
{
        bit1 = 1;
        bit2 = bit3;
}
```

[CA850]

Not provided.

Use a bit field.

```
struct{
        unsigned char   f0:1;
        unsigned char   f1:1;
        unsigned char   f2:1;
        unsigned char   f3:1;
        unsigned char   f4:1;
        unsigned char   f5:1;
        unsigned char   f6:1;
        unsigned char   f7:1;
};
```

<Example>

```
struct   bitf{
         unsigned int bit1:1;
         unsigned int bit2:1;
         unsigned int bit3:1;
}bitfield;

void func(void)
{
         bitfield.bit1 = 1;
         bitfield.bit2 = bitfeeld.bit3;
}
```

2.3.7 Bit access

The CA850 cannot define a bit access using a type variable. Use a bit field to describe a bit access.

[CC78K4]

The description format is as follows.

variable name.bit position

<Example>

```
unsigned char data;
void func(void)
{
      data = 0xff;
      data.1 = 0;
}
```

[CA850]

Not provided.

Use a union and a bit field.

<Example>

```
union{
    unsigned char cdata;
    struct{
           unsigned char bit0:1;
           unsigned char bit1:1;
           unsigned char bit2:1;
           unsigned char bit3:1;
           unsigned char bit4:1;
           unsigned char bit5:1;
           unsigned char bit6:1;
           unsigned char bit7:1;
    }bitfield;
}data;

void func(void)
{
    data.cdata = 0xff;
    data.bitfield.bit1 = 0;
}
```

2.3.8 callf function

The V850 Family does not have an instruction equivalent to the callt instruction. Use an ordinary function.

[CC78K4]

callf/__callf can call a function using the callf instruction with a code shorter than when using the call instruction.

The description format is as follows.

callf type name function name

__callf type name function name

<Example>

```
callf void func(void)
{
        …
}
```

[CA850]

No callf function is available.

Use an ordinary function.

<Example>

```
void func(void)
{
        …
}
```

2.3.9 Binary constant

The description format is the same in both the CC78K4 and CA850.

[CC78K4] [CA850]

Binary constants can be described where integer constants can be described.

The description format is as follows.

0b binary constant

0B binary constant

<Example>

```
ucdata1 = 0b00010001;
ucdata2 = 0B11110000;
```

2.3.10 Specification of interrupt level

The CA850 allows the description of an interrupt level specification.

[CC78K4]

Not provided

[CA850]

A function for controlling the interrupt level (INT level) is available.

An integer value of –1 to 8 can be specified as the interrupt priority level.

Specify an interrupt request name from the maskable interrupts defined in the device file under “interrupt request name”.

If –1 is specified as the interrupt priority level, acknowledgement of maskable interrupts is disabled. If 0 is specified, it is enabled.

The description format is as follows.

__set_il (interrupt priority level, “interrupt request name”)

The meaning of a value of the interrupt priority level is as follows.

–1: Disables acknowledgement of maskable interrupts.

0: Enables acknowledgement of maskable interrupts.

1 to 8: Sets interrupt priority level 0 to 7.

Note that the specified value is the value of the level plus 1.

<Example>

When the priority level of the interrupt request name INTP110 is 1

```
__set_il(2,"INTP110");
```

2.3.11 Pascal function

The CA850 does not have a Pascal function. Use an ordinary function.

[CC78K4]

The Pascal function generates a code to correct the stack used by piling up arguments when a function is called on the called function side, instead of on the function calling side.

When a function is declared, the __pascal attribute is prefixed.

<Example>

```
__pascal void func(void);

void func(void)
{
        …
}
```

[CA850]

No Pascal function is available.

Use an ordinary function.

<Example>

```
void func(void);

void func(void)
{
        …
}
```

2.4 Size and Alignment Conditions of Variables

The CC78K4 and CA850 differ in the size of the area reserved by RAM and the alignment conditions of variables, even when the type is the same.

2.4.1 Size of variable type

The differences in the size of the variable types in the CC78K4 and CA850 are as follows.

Table 2-4. Differences in Size of Type

Type	CC78K4	CA850
char	1 byte	1 byte
short	2 bytes	2 bytes
int[Note]	2 bytes	4 bytes
long	4 bytes	4 bytes
float	4 bytes	4 bytes
double	4 bytes	4 bytes

Note Note that the size of the int type differs.

2.4.2 Alignment conditions

The alignment conditions of variables differ in the CC78K4 and CA850 as follows.

[CC78K4]

Table 2-5. Alignment Conditions (CC78K4)

Option	Alignment Condition
Default	Byte boundary alignment
-RA option	2-byte boundary alignment for external variables of 2 bytes or more (except variables allocated to saddr area)

[CA850]

Table 2-6. Alignment Conditions (CA850)

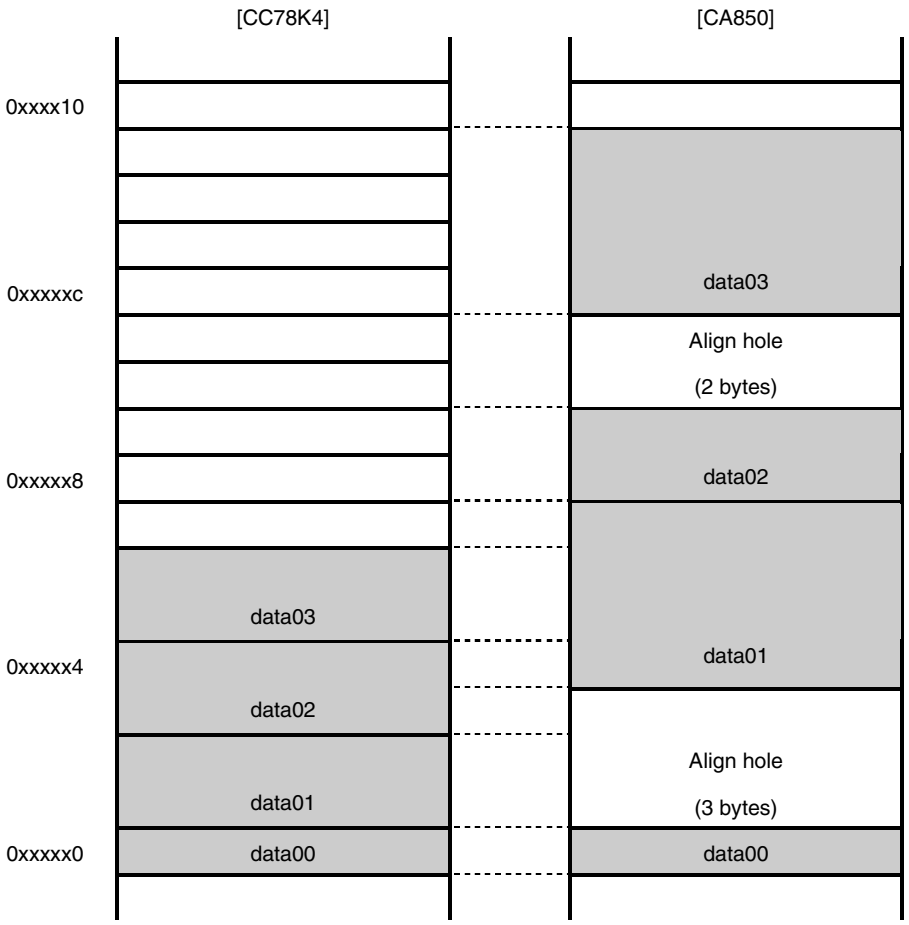
Type	Size	Alignment Condition
Basic type	(unsigned) chart and its array	Byte boundary
	(unsigned) short and its array	2-byte boundary
	Other basic types (including pointer)	4-byte boundary
Union	2 bytes < size	4-byte boundary
	Size ≤ 2 bytes	Maximum member size boundary
Structure	2 bytes < size	4-byte boundary
	Size ≤ 2 bytes If member of type greater than int type exists	4-byte boundary
	Size ≤ 2 bytes If member greater than int type does not exist and if 1 byte < size of type ≤ 2 bytes	2-byte boundary
	Size ≤ 2 bytes If member greater than int type does not exist and if size of type ≤ 1 byte	Byte boundary

<Example 1>

```
struct ST0 {
        char    data00;
        int     data01;
        short   data02;
        int     data03;
}ST0data;
```

The image on RAM is as shown below.

Figure 2-1. RAM Image of Example 1



Because the size of this structure includes an align hole, it is 7 bytes in the CC78K4 and 16 bytes in the CA850.

The alignment condition of the structure is 1-byte alignment in the CC78K4 and 4-byte alignment in the CA850.

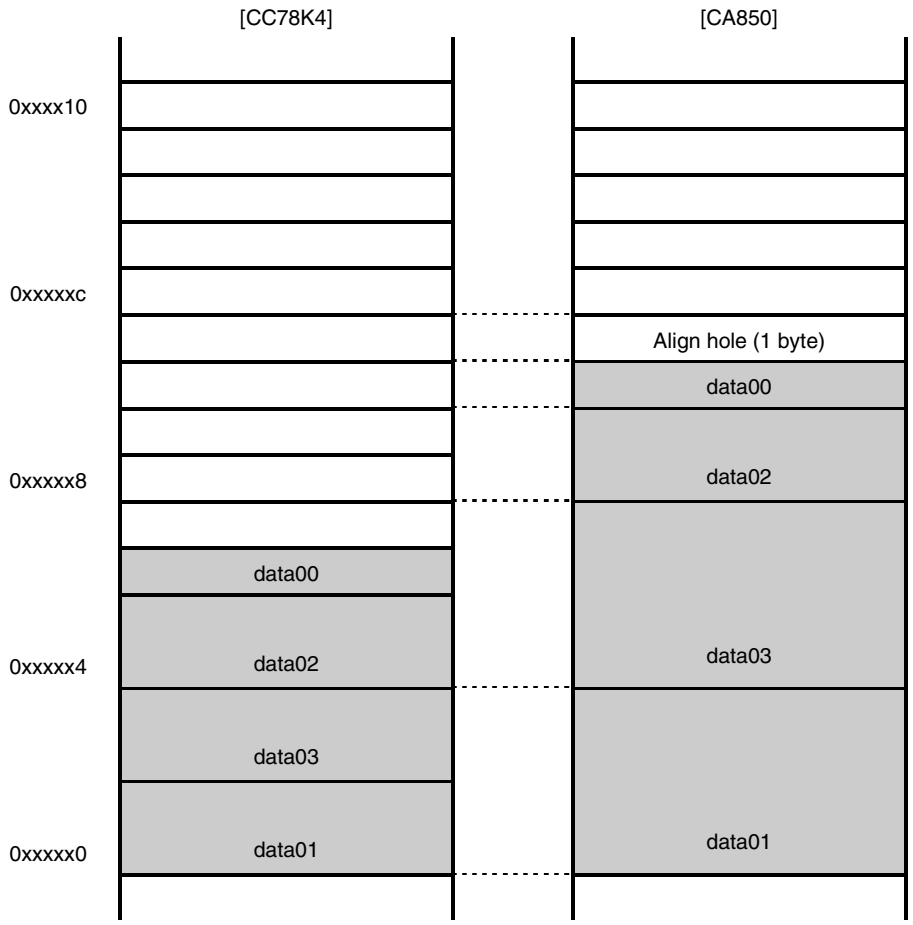
The number of align holes can be decreased by defining the size of variables, starting from the greatest, as follows.

<Example 2>

```
struct ST0 {
        int     data01;
        int     data03;
        short   data02;
        char    data00;
}ST0data;
```

The image on RAM is as shown below.

Figure 2-2. RAM Image of Example 2



The size of this structure is 7 bytes in the CC78K4 and 12 bytes in the CA850.

2.5 Startup Routine (Startup Module)

This section compares the large model of the CC78K4 with the contents of the default startup routine (startup module) of the CA850 by function. Use this section for reference when customizing a startup routine.

For the flow of the program (whole program), refer to the manual of each compiler.

2.5.1 Setting module name and loading include file

The CC78K4 sets a module name and loads an include file. The CA850 does not have such processing.

[CC78K4]

```
        NAME    @cstart
$INCLUDE (mod.inc)
```

[CA850]

Not provided

2.5.2 Setting library switch

The CC78K4 allows the startup routine to be changed to accord with the library used. Therefore, symbols corresponding to the library used are defined.

The CA850 does not have such processing.

[CC78K4]

```
BRKSW     EQU     1       ;brk,sbrk,calloc,free,malloc,realloc function use
EXITSW    EQU     1       ;exit,atexit function use
RANDSW    EQU     1       ;rand,srand  function use
DIVSW     EQU     1       ;div         function use
LDIVSW    EQU     1       ;ldiv        function use
STRTOKSW  EQU     1       ;strtok      function use
```

[CA850]

Not provided

2.5.3 Defining symbols

The CC78K4 has symbols that are used when the standard library is used, but the CA850 does not have such symbols.

If the V850E is specified in the CA850, a symbol must be defined to specify the runtime library code of the prologue/epilogue of a function.

[CC78K4]

```
Start/end symbols of startup routine
        PUBLIC  _@cstart,_@cend
Symbols used when standard library is used
        PUBLIC  _errno,_@BRKADR,_@MEMTOP,_@MEMBTM
        PUBLIC  _@FNCTBL,_@FNCENT,_@SEED,_@DIVR,_@LDIVR,_@TOKPTR
Symbols for stack resolution
        EXTRN   _@STBEG
Symbols of main module
        EXTRN   _main,_hdwinit,_exit
Symbols for ROMization processing
        EXTRN   _?R_INIT,_?R_INIS,_?R_INS1,_?DATS,_?DATS1,_?DATA
```

[CA850]

```
Start/end symbols of startup routine
        .globl   __start
        .globl  __exit
        .globl   __startend
Symbols automatically generated by link editor
        .extern __tp_TEXT, 4
        .extern __gp_DATA, 4
        .extern __ep_DATA, 4
Start/end symbols of section
        .extern  __ssbss, 4
        .extern __esbss, 4
        .extern __sbss, 4
        .extern __ebss, 4
Symbol of main module
        .extern  _main
Symbols used to generate runtime library code of prologue/epilogue of function (V850E only)
        .globl  ___PROLOG_TABLE
```

2.5.4 Reserving area for library

With the CA850, a heap area must be reserved only when the storage area management library of the standard library is used.

[CC78K4]

```
@@DATA   DSEG
_@FNCTBL:         DS       3*32
_@FNCENT:         DS       2
_@SEED:           DS       4
_@DIVR:           DS       4
_@LDIVR:          DS       8
_@TOKPTR:         DS       3
_errno:           DS       2
_@BRKADR:         DS       3
_@MEMTOP:         DS       48
_@MEMBTM:
```

[CA850]

Set a heap area as follows to use the storage area management library of the standard library. This description can be made in the C source, instead of the startup routine.

```
        .sbss
        .comm    ___sysheap, HEAPSIZE, 4
        .sdata
        .globl   ___sizeof_sysheap
___sizeof_sysheap:
        .word    HEAPSIZE
```

2.5.5 Reserving stack area

The CC78K4 can reserve a stack area using the linker but an area must be explicitly reserved for the CA850.

[CC78K4]

No description of a stack area is available.

By specifying the -S option using the linker, the maximum address area of the memory area is found and the _@STBG and _@STEND symbols are generated.

[CA850]

```
        .set     STACKSIZE, 0x200
        .bss
        .lcomm   __stack, STACKSIZE, 4
```

Caution Be sure to reserve a stack in 4-byte units.

2.5.6 Setting reset vector

Setting a reset vector cannot be described using the #pragma directive in either the CC78K4 or the CA850. The startup routine therefore includes this description.

[CC78K4]

```
@@VECT00 CSEG     AT       0H
         DW       _@cstart
```

[CA850]

```
        .section        "RESET", text
        jr      __start
```

2.5.7 Setting location

The location instruction of the 78K/IV Series is not provided in the V850 Family.

[CC78K4]

```
        LOCATION          0FH
```

[CA850]

Not provided

2.5.8 Setting register bank

The register bank of the 78K/IV Series is not provided in the V850 Family.

[CC78K4]

```
        SEL     RB0
```

[CA850]

Not provided

2.5.9 Setting stack pointer

The CC78K4 sets a symbol generated by the linker to the stack pointer.

The CA850 adds the size of the stack to the symbol at the beginning of the area explicitly reserved and sets the result to the stack pointer.

[CC78K4]

```
        MOVG    SP,#_@STBEG
```

[CA850]

```
        mov     #__stack+STACKSIZE, sp
```

2.5.10 Setting general-purpose registers

The CA850 accesses a program and variables by using a relative address calculated from a register value. Therefore, general-purpose registers must be set.

These registers must also be set when the mask register function is used.

[CC78K4]

The registers do not have to be set in the case of the large model. For the medium and small models, refer to the sample below.

[CA850]

```
        mov     #__tp_TEXT, tp:         -- Setting of text pointer
        mov     #__gp_DATA, gp:         --
        add     tp, gp:                 -- Setting of global pointer
        mov     #__ep_DATA, ep:         -- Setting of element pointer
        mov     0xff,   r20:            -- Setting of mask register (byte)
        mov     0xffff, r21:            -- Setting of mask register (halfword)
```

It is assumed that “use of the offset from the text pointer as the global pointer” is specified in the link directive file. Therefore, tp and gp are added and substituted for gp.

The mask registers (r20 and r21) are necessary only when use of the mask registers is specified by an option.

2.5.11 Setting special registers

The CA850 requires special registers to be set only when the V850E generates the runtime library code of the prologue/epilogue of a function.

[CC78K4]

Not provided

[CA850]

Set the CALLT table pointer as follows to generate the runtime library code of the prologue/epilogue of a function (V850E only).

```
        mov     #___PROLOG_TABLE, r12
        ldsr    r12, 20
```

2.5.12 Calling hardware initialization function

The CC78K4 allows a hardware initialization function to be called so that customization can be made by the user, but the CA850 does not. To initialize the hardware, call a function in the same manner as in the CC78K4, or describe a function call in the startup routine.

[CC78K4]

```
        CALL    !!_hdwinit
```

[CA850]

Not provided

2.5.13 Setting default value of standard library

The CC78K4 requires the default values of symbols to be set when the standard library is used, but the CA850 does not.

When using the standard library in the CA850, however, ROMization processing (romp850) is necessary because variables with a default value exist.

[CC78K4]

```
        SUBW    AX,AX
        MOVW    !!_errno,AX         ;errno <- 0
        MOVW    !!_@FNCENT,AX       ;FNCENT <- 0
        MOVW    !!_@SEED+2,AX
        MOVW    !!_@SEED,#1         ;SEED <- 1
        MOVG    WHL,#_@MEMTOP
        MOVG    !!_@BRKADR,WHL      ;BRKADR <- #MEMTOP
```

[CA850]

Not provided

2.5.14 ROMization processing

With the CC78K4, ROMization processing is described in the startup module. With the CA850, the ROMization library must be called at the part of the C source that is executed first (beginning of the main function).

To expand the memory, set the necessary peripheral function registers before calling the ROMization library.

[CC78K4]

(1/2)

```
; copy external variables having initial value
        MOVG    TDE,#_@INIT
        MOVG    WHL,#_@R_INIT
LINIT1:
        SUBG    WHL,#_?R_INIT
        BE      $LINIT2
        ADDG    WHL,#_?R_INIT
        MOV     A,[HL+]
        MOV     [DE+],A
        BR      $LINIT1
LINIT2:
; copy external variables which doesn't have initial value
        MOVG    TDE,#_@DATA
        MOVG    WHL,#_?DATA
        MOV     A,#0
LDATA1:
        SUBG    WHL,TDE
        BE      $LDATA2
        ADDG    WHL,TDE
```

(2/2)

```
          MOV       [DE+],A
          BR        $LDATA1
LDATA2:
; copy sreg variables having initial value
          MOVG      TDE,#_@INIS
          MOVG      WHL,#_@R_INIS
LINIS1:
          SUBG      WHL,#_?R_INIS
          BE        $LINIS2
          ADDG      WHL,#_?R_INIS
          MOV       A,[HL+]
          MOV       [DE+],A
          BR        $LINIS1
LINIS2:
; copy sreg variables which doesn't have initial value
          MOVG      TDE,#_@DATS
          MOVG      WHL,#_?DATS
          MOV       A,#0
LDATS1:
          SUBG      WHL,TDE
          BE        $LDATS2
          ADDG      WHL,TDE
          MOV       [DE+],A
          BR        $LDATS1
LDATS2:
; copy sreg1 variables having initial value
          MOVG      TDE,#_@INIS1
          MOVG      WHL,#_@R_INS1
LINIS11:
          SUBG      WHL,#_?R_INS1
          BE        $LINIS12
          ADDG      WHL,#_?R_INS1
          MOV       A,[HL+]
          MOV       [DE+],A
          BR        $LINIS11
LINIS12:
; copy sreg1 variables which doesn't have initial value
          MOVG      TDE,#_@DATS1
          MOVG      WHL,#_?DATS1
          MOV       A,#0
LDATS11:
          SUBG      WHL,TDE
          BE        $LDATS12
          ADDG      WHL,TDE
          MOV       [DE+],A
          BR        $LDATS11
LDATS12:
```

[CA850]

Call a copy routine (_rcopy) at the beginning of the main function.

```
extern   unsigned long     _S_romp;
main()
{
         int      ret;
         ret = _rcopy(&_S_romp,-1);
         …
}
```

2.5.15 Initializing variable area without default value

The CA850 executes initialization by using a symbol that starts/ends a section without default value.

[CC78K4]

Not provided

[CA850]

```
         mov      #__ssbss, r13          -- clear sbss section
         mov      #__esbss, r12
         cmp      r12, r13
         jnl      .L11
.L12:
         st.w     r0, [r13]
         add      4, r13
         cmp      r12, r13
         jl       .L12
.L11:
         mov      #__sbss, r13           -- clear bss section
         mov      #__ebss, r12
         cmp      r12, r13
         jnl      .L14
.L15:
         st.w     r0, [r13]
         add      4, r13
         cmp      r12, r13
         jl       .L15
.L14:
```

The above symbols are reserved words in the link editor.

__ssbss: Symbol starting .sbss section

__esbss: Symbol ending .sbss section

__sbss: Symbol starting .bss section

__ebss: Symbol ending .bss section

2.5.16 Calling main function

The main function is called.

[CC78K4]

```
        CALL    !!_main
```

[CA850]

```
        jarl    _main, lp
```

2.5.17 Calling exit function

The CC78K4 calls the exit function but the CA850 calls the halt instruction to set the HALT mode.

[CC78K4]

```
        SUBW    AX,AX
        CALL    !!_exit
        BR      $$
```

[CA850]

Not provided. The HALT mode is set as is.

```
        halt
```

2.5.18 Defining segment (section)

The CC78K4 defines segments and labels used for ROMization processing.

The CA850 defines a section without default value to initialize the section without default value.

[CC78K4]

```
@@R_INIT  CSEG
_@R_INIT:
@@R_INIS  CSEG
_@R_INIS:
@@R_INS1  CSEG
_@R_INS1:
@@INIT    DSEG
_@INIT:
@@DATA    DSEG
_@DATA:
@@INIS    DSEG     SADDR2
_@INIS:
@@DATS    DSEG     SADDR2
_@DATS:
@@INIS1   DSEG     SADDR
_@INIS1:
@@DATS1   DSEG     SADDR
_@DATS1:
@@CODE    CSEG
@@CALF    CSEG     FIXED
@@CNST    CSEG
@@CALT    CSEG     CALLT0
@@BITS    BSEG     SADDR2
@@BITS1   BSEG     SADDR
```

[CA850]

```
        .sbss
        .lcomm   __sbss_dummy, 0, 0
```

2.6 Segment (Section) Output by Compiler

This section explains the segment output by the CC78K4 and section output by the CA850.
For how to allocate the segment and section, refer to **CHAPTER 4 LINK DIRECTIVES**.

2.6.1 Segment Output by CC78K4

The CC78K4 outputs the following segments by default.

Table 2-7. Segments Output by CC78K4

Section Name	Segment Type	Usage
@@BASE	CSEG	callt function, interrupt function segment
@@VECTnn	CSEG	Segment for interrupt vector table
@@CODES @@CODE	CSEG	Segment for ordinary function codes
@@CNSTS @@CNSTM @@CNST	CSEG	Segment for const variable
@@CALFS @@CALF	CSEG	Segment for callf function
@@CALT	CSEG	Segment for callt function table
@@RSINIT @@R_INIT	CSEG	Segment for variable with default value
@@RSINS @@R_INS	CSEG	Segment for sreg variable with default value
@@RSINS1 @@R_INS1	CSEG	Segment for sreg1 variable with default value
@@INITM @@INIT	DSEG	Segment for temporary variable with default value
@@DATAM @@DATA	DSEG	Segment for variable without default value
@@INIS	DSEG	Segment for temporary sreg variable with default value
@@DATS	DSEG	Segment for temporary sreg variable without default value
@@INIS1	DSEG	Segment for temporary sreg1 variable with default value
@@DATS1	DSEG	Segment for temporary sreg1 variable without default value
@@BITS	BSEG	Segment for boolean/bit type variable
@@BITS1	BSEG	Segment for __boolean1 type variable

2.6.2 Section output by CA850

The CA850 outputs the following sections by default.

Table 2-8. Sections Output by CA850

Section Name	Type of Section	Usage
.text	text	Section for function code
.pro_epi_runtime	text	Section for runtime library code of prologue/epilogue
.const	const	Section for const variable Section that can be accessed by ld/st instructions (two instructions) of r0 relative
.sconst	const	Section for const variable Section that can be accessed by ld/st instruction of r0 relative (addresses 0x0 to 0x8000)
.data	data	Section for variable with default value Section that can be accessed by ld/st instructions (two instructions) of gp relative
.sdata	sdata	Section for variable with default value Section that can be accessed by ld/st instruction of gp relative
.sedata	sedata	Section for variable with default value Section that can be accessed by ld/st instruction of ep relative (first address of internal RAM – 0x8000 to first address of internal RAM are recommended)
.sidata	sidata	Section for variable with default value Section that can be accessed by ld/st instruction of ep relative (internal RAM is recommended)
.tidata	tidata	Section for variable with default value Section that can be accessed by sld/sst instruction of ep relative (256 bytes starting from first address of internal RAM is recommended)
.bss	bss	Section for variable without default value Section that can be accessed by ld/st instructions (two instructions) of gp relative
.sbss	sbss	Section for variable without default value Section that can be accessed by ld/st instruction of gp relative
.sebss	sebss	Section for variable without default value Section that can be accessed by ld/st instruction of ep relative (first address of internal RAM – 0x8000 to first address of internal RAM are recommended)
.sibss	sibss	Section for variable without default value Section that can be accessed by ld/st instruction of ep relative (internal RAM is recommended)
.tibss	tibss	Section for variable without default value Section that can be accessed by sld/sst instruction of ep relative (256 bytes starting from first address of internal RAM is recommended)
rompsec	text	Sections including data section with default value packed for copying data with default value and section including address information of these sections

2.7 Library and Header File

This section explains the library and header file supported by the CC78K4 and CA850.

If a library that is supported by the CC78K4 but not by the CA850 is used, replace the library with one supported by the CA850.

Table 2-9. Library and Header File (1/2)

Function	CC78K4	CA850
Character, character string function	isalnum, isalpha, iscntrl, isdigit, isgragh, islower, isprint, ispunct, isspace, isupper, isxdigit, tolower, toupper, isascii, toascii, _toupper, _tolower, tolow*, toup*	_tolower, _toupper, toascii, tolower, toupper, isalnum, isalpha, isascii, iscntrl, isdigit, isgraph, islower, isprint, ispunct, isspace, isupper, isxdigit
Program control function	setjmp, longjmp	setjmp, longjmp
Special function	va_start, va_arg, va_end	va_start, va_arg, va_end
I/O function	sprintf, sscanf, printf, scanf, vprintf, vsprintf, getchar, gets, putchar, puts	sprintf, sscanf, fprintf*, fscanf*, printf, scanf, vfprintf*, vprintf, vsprintf, fgetc*, fgets*, fputc*, fputs*, getc*, getchar, gets, putc*, putchar, puts, ungetc*, fread*, fwrite*, rewind*, perror*
Utility function	atoi, atol, strtol, strtoul, calloc, free, malloc, realloc, abort*, atexit*, exit*, abs, labs, div, ldiv, brk*, sbrk*, atof*, strtod*, itoa, ltoa, ultoa, rand, srand, bsearch, qsort, strbrk*, strsbrk*, stritoa*, strltoa*, strultoa*	abs, labs, bsearch, div, ldiv, ecvtf*, gcvtf*, atoff*, strtodf*, atoi, atol, strtol, strtoul, rand, srand, calloc, malloc, free, realloc, qsort, itoa, ltoa, ultoa
Character string/ memory function	memcpy, memmove, strcpy, strncpy, strcat, strncat, memcmp, strcmp, strncmp, memchr, strchr, strrchr, strspn, strcspn, strpbrk, strstr, strtok, memset, strerror, strlen, strcoll, strxfrm	bcmp*, bcopy*, memchr, memcmp, memcpy, memmove, memset, index*, rindex*, strcat, strchr, strcmp, strcpy, strcspn, strerror, strlen, strncat, strncmp, strncpy, strpbrk, strrchr, strspn, strstr, strtok
Mathematical function	acos*, asin*, atan*, atan2*, cos*, sin*, tan*, cosh*, sinh*, tanh*, exp*, frexp*, ldexp*, log*, log10*, modf*, pow*, sqrt*, ceil*, fabs*, floor*, fmod*, matherr, acosf, asinf, atanf, atan2f, cosf, sinf, tanf, coshf, sinhf, tanhf, expf, frexpf, ldexpf, logf, log10f, modff, powf, sqrtf, ceilf, fabsf, floorf, fmodf, __assertfail*	j0f*, j1f*, jnf*, y0f*, y1f*, ynf*, erff*, erfcf*, expf, logf, log2f*, log10f, powf, sqrtf, ceilf, fabsf, floorf, fmodf, frexpf, ldexpf, modff, gammaf*, hypotf*, matherr, acoshf*, asinhf*, atanhf*, coshf, sinhf, tanhf, acosf, asinf, atanf, atan2f, cosf, sinf, tanf, cbrtf*

Table 2-9. Library and Header File (2/2)

Function	CC78K4	CA850
Integer operation[Note]	lsinc*, luinc*, lsdec*, ludec*, lsrev*, lurev*, lscom*, lucom*, lsnot*, lunot*, lsmul*, lumul*, csdiv*, isdiv*, lsdiv*, ludiv*, csrem*, isrem*, lsrem*, lurem*, lsadd*, luadd*, lssub*, lusub*, lslsh*, lulsh*, lsrsh*, lursh*, lscmp*, lucmp*, lsband*, luband*, lsbor*, lubor*, lsbxor*, lubxor*	__mul*, __mulu*, __div*, __divu*, __mod*, __modu*
Floating-point operation[Note]	finc*, fdec*, frev*, fnot*, fmul*, fdiv*, fadd*, fsub*, fcmp*, fand*, for*, ftols*, ftoul*, lstof*, lutof*, btol*,	__addf.s*, __subf.s*, __mulf.s*, __divf.s*, __cmpf.s*, __cvt.ws*, __trnc.sw*
Copying ROMization default value data	None	_rcopy*
Header file	ctype.h, setjmp.h, stdarg.h, stdio.h, stdlib.h, string.h, error.h*, errno.h, limits.h, stddef.h, math.h, float.h, assert.h*	ctype.h, setjmp.h, stdarg.h, stdio.h, stdlib.h, string.h, errno.h, limits.h, stddef.h, math.h, float.h

Note Because these functions are those of a runtime library, they are not described in the C source program.

The functions marked “*” in the above table are supported only by either the CC78K4 or CA850. For the details of the library, refer to the manual of each compiler.

CHAPTER 3 ASSEMBLY LANGUAGE

This chapter explains the points to be noted when rewriting the quasi-directives and control instructions of the assembler for the RA78K4 into those for the CA850 (as850), and how to describe a program.

The quasi-directives and control instructions of the RA78K4 and CA850 are as follows.

Table 3-1. Quasi-Directives and Control Instructions (1/3)

No.	RA78K4		CA850	
	Function	Instruction	Function	Instruction
1	Segment quasi-directive	CSEG	Section definition quasi-directive	.text .const .sconst
		DSEG		.bss .data .sbss .sdata .sebss .sedata .sibss .sidata .tibss .tidata .tibss.byte .tidata.byte .tibss.word .tidata.word
		BSEG		None
		None		.previous .section .vdbstrtab .vdebug .vline
		ORG	Location counter control quasi-directive	.org
		None		.align
2	Symbol definition quasi-directive	EQU	None	
		SET	Symbol control quasi-directive	.set
		None		.size .frame .file
3	Object module name declaration quasi-directive	NAME	None	

Table 3-1. Quasi-Directives and Control Instructions (2/3)

No.	RA78K4		CA850	
	Function	Instruction	Function	Instruction
4	Memory initialization, area reservation quasi-directive	DB	Area reservation quasi-directive	.byte
		DW		.hword
		DG		None
		None		.word
		DS		.lcomm
		None		.space
		DBIT		None
		None		.float
5	Linkage quasi-directive	PUBLIC	Program linkage quasi-directive	.globl
		EXTRN		.extern
		EXTBIT		None
		None		.comm
6	Automatic selection quasi-directive	BR CALL	None	
7	General-purpose register selection quasi-directive	RSS	None	
8	Macro quasi-directive	MACRO	Macro quasi-directive	.macro
		LOCAL		.local
		ENDM		.endm
		EXITM	Skip quasi-directive	.exitm
		None		.exitma
		REPT	Repeat assemble quasi-directive	.repeat
		IRP		.irepeat
9	Assemble end quasi-directive	END	None	
10	Assemble target model specification control instruction	$PROCESSOR	Assembler control quasi-directive	.option cpu
11	Debug information output control instruction	$DEBUG/ $NODEBUG $DEBUGA/ $NODEBUGA	None	
12	Cross-reference list output specification control instruction	$XREF/$NOXREF $SYMLIST/ $NOSYMLIST	None	
13	Include control instruction	$INCLUDE	File input control quasi-directive	.include
		None		.binclude

Table 3-1. Quasi-Directives and Control Instructions (3/3)

No.	RA78K4		CA850	
	Function	Instruction	Function	Instruction
14	Assemble list control instruction	$EJECT $TITLE $SUBTITLE $LIST/$NOLIST $GEN/$NOGEN $COND/$NOCOND $FORMFEED/ $NOFORMFEED $WIDTH $LENGTH $TAB	None	
15	Conditional assembly control instruction	$SET $RESET	None	
		$IF	Conditional assembly quasi-directive	.ifdef
		None		.ifndef
		$_IF		.if
		None		.ifn
		$ELSEIF $_ELSEIF		.elseif
		None		.elseifn
		$ELSE		.else
		$ENDIF		.endif
16	SFR area change control instruction	$CHGSFR $CHGSFRA	None	

3.1 Segment Quasi-Directive (Section Definition Quasi-Directive)

This section explains quasi-directives related to segments (sections).

3.1.1 CSEG, .text, .const, .sconst

These are quasi-directives that define the segments (sections) to be allocated to ROM.

The RA78K4 uses the CSEG quasi-directive to define the segment of a program and a variable that is only referenced. The CA850 uses the .text quasi-directive to define a program section, and the .sconst or .const quasi-directive to define the section of a variable that is only referenced.

[RA78K4]

CSEG: Indicates the start of a code segment.
The description format is as follows.

[segment name] CSEG [relocation attribute]

<Example>

```
         CSEG     UNIT
         MOV      A,B
         CSEG     BASE
DATAC0:  DB       12H
DATAC1:  DB       34H
         CSEG
DATAC2:  DW       5678H
```

[CA850]

.text: Allocates generated codes to the .text section.
The description format is as follows.

.text

.const: Allocates generated codes to the .const section.
The .const section is a section for constant data (read-only) and is located in memory that is referenced by two instructions, using r0 and 32-bit displacement.
The description format is as follows.

.const

.sconst: Allocates generated codes to the .sconst section.
The .sconst section is a section for constant data (read-only) and is located in a memory range (up to 32 KB in the plus direction from r0) that is referenced by one instruction, using r0 and 16-bit displacement.
The description format is as follows.

.sconst

<Example>

```
        .text
        .align 4
        mov     r10,r11
        .const
DATAC0:
        .byte   0x12
DATAC1:
        .byte   0x34
        .sconst
DATAC2:
        .hword  0x5678
```

3.1.2 DSEG, .bss, .data, .sbss, .sdata, .sebss, .sedata, .sibss, .sidata, .tibss, .tidata, .tibss.byte, .tidata.byte, .tibss.word, .tidata.word

These quasi-directives define segments (sections) to be allocated to RAM.

The RA78K4 uses the DESG quasi-directive to define the segments of variables other than bit variables. In the CA850, bit variables do not have sections, and the section of a variable is defined by using the .bss and .data quasi-directives, which are not dependent on the address to be located, or the .sbss and .sdata quasi-directives, which are dependent on the GP register value. In addition, defining to the external RAM immediately before the internal RAM is performed by the .sebss or .sedata quasi-directive. Defining to the internal RAM is performed by the .sibss, .sidata, .tibss, .tidata, tibss.byte, .tidata.byte, tibss.word, or .tidata.word quasi-directive.

[RA78K4]

DSEG: Indicates the start of a data segment.
The description format is as follows.

[segment name] DSEG [relocation attribute]

<Example>

```
         DSEG     UNIT
DATAD0:  DS       1
         DSEG     UNITP
DATAD2:  DS       2
         DSEG
DATAD4:  DS       2
         DSEG     SADDR
DATAD6:  DS       2
         DSEG     SADDR2
DATAD8:  DS       2
         DSEG     SADDR2
DATAD10: DS       1
         DSEG     SADDRP2
DATAD12: DS       2
```

[CA850]

.bss: Allocates generated codes to the .bss section.
The .bss section does not have a default value and is located in a memory range that is referenced by two instructions, using gp and 32-bit displacement.
The description format is as follows.

.bss

.data: Allocates generated codes to the .data section.
The .data section has a default value and is located in a memory range that is referenced by two instructions, using gp and 32-bit displacement.
The description format is as follows.

.data

.sbss: Allocates generated codes to the .sbss section.
The .sbss section does not have a default value and is located in a memory range (up to 64 KB including the .sdata section) that is referenced by one instruction, using gp and 16-bit displacement.
The description format is as follows.

.sbss

.sdata: Allocates generated codes to the .sdata section.
The .sdata section has a default value and is located in a memory range (up to 64 KB including .sbss section) that is referenced by one instruction, using gp and 16-bit displacement.
The description format is as follows.

.sdata

.sebss: Allocates generated codes to the .sebss section.
The .sebss section does not have a default value and is located at the higher addresses equaling the size of the .sedata section in the memory range (32 KB in the minus direction from ep) that is referenced by one instruction, using ep and 16-bit displacement.
The description format is as follows.

.sebss

.sedata: Allocates generated codes to the .sedata section.
The .sedata section has a default value and is located at the lower addresses equaling the size of the .sebss section in the memory range (32 KB in the minus direction from ep) that is referenced by one instruction, using ep and 16-bit displacement.
The description format is as follows.

.sedata

.sibss: Allocates generated codes to the .sibss section.
The .sibss section does not have a default value and is located in the memory range (32 KB in the plus direction from ep, i.e., internal RAM) that is referenced by one instruction, using ep and 16-bit displacement.
The description format is as follows.

.sibss

.sidata: Allocates generated codes to the .sidata section.
The .sidata section has a default value and is located in the memory range (32 KB in the plus direction from ep, i.e., internal RAM) that is referenced by one instruction, using ep and 16-bit displacement.
The description format is as follows.

.sidata

.tibss: Allocates generated codes to the .tibss section.
The .tibss section is a section of data without a default value and is assumed to be accessed by ep relative, using the sld/sst instruction[Note]. If both the .tibss.byte and .tibss.word sections are used, .tibss is located at the address indicated by ep to which the sum of the size of the both the sections is added.
The description format is as follows.

.tibss

.tidata: Allocates generated codes to the .tidata section.
The .tidata section is a section of data with a default value and is assumed to be accessed by ep relative, using the sld/sst instruction[Note]. If both the .tidata.byte and .tidata.word sections are used, .tidata is located at the address indicated by ep to which the sum of the size of both the sections is added.
The description format is as follows.

.tidata

.tibss.byte: Allocates generated codes to the .tibss.byte section.
The .tibss.byte section is a section of data without a default value and is assumed to be accessed by ep relative, using the sld/sst instruction[Note]. To effectively use the area that can be accessed by the sld/sst instruction[Note], the section is divided into a .tibss.byte section and a .tibss.word section, depending on the size of the data, and the .tibss.byte section is allocated at the address indicated by ep.
The description format is as follows.

.tibss.byte

.tidata.byte: Allocates generated codes to the .tidata.byte section.
The .tidata.byte section is a section of data with a default value and is assumed to be accessed by ep relative, using the sld/sst instruction[Note]. To effectively use the area that can be accessed by the sld/sst instruction[Note], the section is divided into a .tidata.byte section and a .tidata.word section, depending on the size of the data, and the .tidata.byte section is allocated at the address indicated by ep.
The description format is as follows.

.tidata.byte

Note The sld/sst instruction is a 2-byte instruction that can access an area of up to 128 bytes if the data to be accessed is byte data, or an area of up to 256 bytes if the data to be accessed is a halfword or larger data.

.tibss.word: Allocates generated codes to the .tibss.word section.
The .tibss.word section is a section of data without a default value and is assumed to be accessed by ep relative, using the sld/sst instruction[Note]. To effectively use the area that can be accessed by the sld/sst instruction[Note], the section is divided into a .tibss.byte section and a .tibss.word section, depending on the size of the data, and is located at the address indicated by ep to which the size of the .tibss.byte section is added.
The description format is as follows.

.tibss.word

.tidata.word: Allocates generated codes to the .tidata.word section.
The .tidata.word section is a section of data with a default value and is assumed to be accessed by ep relative, using the sld/sst instruction[Note]. To effectively use the area that can be accessed by the sld/sst instruction[Note], the section is divided into a .tidata.byte section and a .tidata.word section, depending on the size of the data, and is located at the address indicated by ep to which the size of the .tidata.byte section is added.
The description format is as follows.

.tidata.word

Note The sld/sst instruction is a 2-byte instruction that can access an area of up to 128 bytes if the data to be accessed is byte data, or an area of up to 256 bytes if the data to be accessed is a halfword or larger data.

<Example>

```
        .bss
        .lcomm   DATAD0,0x1,1
        .data
DATAD1:
        .byte    0xff
        .sbss
        .lcomm   DATAD2,0x2,2
        .sdata
DATAD3:
        .hword   0xffff
        .sebss
        .lcomm   DATAD4,0x2,2
        .sedata
DATAD5:
        .hword   0xabcd
        .sibss
        .lcomm   DATAD6,0x2,2
        .sidata
DATAD7:
        .hword   0xfedc
        .tibss
        .lcomm   DATAD8,0x2,2
        .tidata
DATAD9:
        .hword   0x4321
        .tibss.byte
        .lcomm   DATAD10,0x1,1
        .tidata.byte
DATAD11:
        .byte    0x21
        .tibss.word
        .lcomm   DATAD12,0x2,2
        .tidata.word
DATAD13:
        .hword   0x5678
```

3.1.3 BSEG

This quasi-directive defines a segment (section) to be allocated to RAM.

The RA78K4 uses the BSEG quasi-directive to define the segment of a bit variable, but in the CA850, a bit variable does not have a section.

[RA78K4]

BSEG: Indicates the start of a bit segment.
The description format is as follows.

[segment name] BSEG [relocation attribute]

<Example>

```
        BSEG
DATAB0  DBIT
DATAB1  DBIT
        CSEG
        SET1    DATAB0
        CLR1    DATAB1
```

[CA850]

Not provided.
Reserve an area in a size such as bytes, and access the area by specifying a bit position using the set1, clr1, or tst1 instruction.

<Example>

```
        .bss
        .lcomm  DATAB,0x1,1
        .text
        set1    0,$DATAB[gp]
        clr1    1,$DATAB[gp]
```

3.1.4 .previous, .section

These quasi-directives define a segment (section).

The RA78K4 can specify a segment name using the CSEG, DSEG, or BSEG quasi-directive. The CA850 specifies a section name using the .section quasi-directive. The CA850 can specify the section specified by the previous section quasi-directive, by using .previous.

[RA78K4]

A quasi-directive equivalent to .previous is not available.

A segment name can be specified by CSEG, DSEG, or BSEG.

<Example>

```
SEG1     CSEG
         MOV      A,B
SEG2     DSEG
SEGL0:   DS       2
```

[CA850]

.previous: Specifies (again) the section definition quasi-directive preceding the section definition quasi-directive that specified the current section quasi-directive.
The description format is as follows.

.previous

.section: Allocates a code generated for the program as a section name specified by the first operand to the type of the section specified by the second operand.
The description format is as follows.

.section “section name” [,type of section]

<Example>

```
.section "SEG1",text
mov      r10,r11
.section "SEG2",bss
.lcomm   SEGL0,2,2
```

3.1.5 ORG, .org

These quasi-directives define a segment (section).

The RA78K4 uses the ORG quasi-directive to specify the absolute address of a segment. The CA850 cannot specify the absolute address of a section but only increments the location counter that indicates the current section.

To specify an absolute value, specify the location of the section using the link directive file.

[RA78K4]

ORG: Sets the value of an expression specified by the operand to the location counter.
After this quasi-directive, the segment is located starting from the address belonging to an absolute segment and specified by the operand.
The description format is as follows.

[segment name] ORG absolute expression

<Example>

```
"MOV A,C" is allocated to the segment that is located at address 0x100.
        CSEG
        MOV     A,B
        ORG     0100H
        MOV     A,C
```

[CA850]

.org: Increments the value of the location counter for the current section specified by the previously specified section definition quasi-directive to the value specified by the operand.
This quasi-directive does not specify the absolute address of a section. If a hole is generated as a result of incrementing the value of the location counter, the hole is filled with 0.
The description format is as follows.

.org value

<Example>

```
"mov r12,r11" is allocated to the address indicated by the location counter of the
.text section incremented by 0x100.

        .text
        mov     r10,r11
        .org    0x100
        mov     r12,r11
```

3.1.6 .align

This quasi-directive specifies the alignment condition of a segment (section).

It is not provided in the RA78K4. The CA850 can specify the alignment condition of a section using the .align quasi-directive.

[RA78K4]

Not provided

[CA850]

.align: Aligns the value of the location counter for the current section specified by the previously specified section definition quasi-directive under the alignment condition specified by the first operand.
If a hole is generated as a result of aligning the location counter value, it is filled with the filling value specified by the second operand or the default value 0.
The description format is as follows.

.align alignment condition [,filling value]

<Example>

```
        .text
        .align 4
        .globl   func
func:
        jmp      [lp]
```

3.2 Symbol Definition Quasi-Directives (Symbol Control Quasi-Directives)

This section explains the quasi-directives related to symbols.

3.2.1 EQU

The RA78K4 has the EQU and SET quasi-directives, which define a name. The CA850 supports only the .set quasi-directive, which sets a value to a name.

[RA78K4]

EQU: Defines a name having the value of the expression specified by the operand.
The description format is as follows.

name EQU expression

<Example>

```
DATAE     EQU     0FE00H
```

[CA850]

Not provided.
Use the .set quasi-directive to specify a numeric value.

3.2.2 SET, .set

The RA78K4 uses the SET quasi-directive to define a name, but the CA850 uses the .set quasi-directive to set a value to a name.

[RA78K4]

SET: Defines a name having the value of the expression specified by the operand.
This quasi-directive can be re-defined in the same module.
The description format is as follows.

name SET absolute expression

<Example>

```
DATAS     SET     100
```

[CA850]

.set: Defines a symbol having the symbol name specified by the first operand and the value (integer value) specified by the second operand.
The description format is as follows.

.set symbol name, value

<Example>

```
.set      DATAE,0xfe00
```

3.2.3 .size, .frame, .file

The CA850 has the .size quasi-directive, which gives a size to a label, the .frame quasi-directive for debugging at the C language level, and the .file quasi-directive, which defines a file name.

[RA78K4]

Not provided

[CA850]

.size: Specifies the size specified by the second operand as the size of the data indicated by the label specified by the first operand.
The description format is as follows.

.size label name, size

.frame: Generates a symbol table entry having the size specified by the second operand and the type FUNC when an object file is generated and when a symbol table entry for the label specified by the first operand is generated.
This quasi-directive is used for debugging at the C language level.
The description format is as follows.

.frame label name, size

.file: Generates a symbol table entry having the file name specified by the operand and type FUNC when an object file is generated.

.file “file name”

<Example>

```
    .file      "main.s"
```

3.3 Object Module Name Declaration Quasi-Directives

This section explains the quasi-directives related to object module names.

3.3.1 NAME

The RA78K4 has the NAME quasi-directive, which defines the name of an object module, but the CA850 does not have such a quasi-directive.

[RA78K4]

NAME: Assigns the object module name described as the operand to the object module output by the assembler.
The description format is as follows.

NAME object module name

[CA850]

Not provided

3.4 Memory Initialization and Area Reservation Quasi-Directives (Area Reservation Quasi-Directives)

This section explains the quasi-directives that initialize the memory or reserve a memory area.

3.4.1 DB, .byte

The RA78K4 uses the DB quasi-directive and the CA850 uses the .byte quasi-directive to reserve a 1-byte area of memory

[RA78K4]

DB: Initializes a byte area (1 byte).
If the size is specified by the operand, an area of the specified size is initialized by 0.
If the default value is specified by the operand, the area is initialized by the specified default value.
The following two description formats are available.

DB (size)
DB default value [, …]

<Example>

```
        CSEG
DBDATA0: DB       (1)
DBDATA1: DB       0AAH,0BBH
```

[CA850]

.byte: The first format of this quasi-directive reserves an area of 1 byte for each operand, and stores the lower byte of the specified value in the reserved area.
The second format reserves an area of the specified bit width, and stores the specified value in that area.
The following two description formats are available.

.byte value [, value…]
.byte bit width: value [, bit width: value …]

<Example>

```
        .sdata
DBDATA0:
        .space   1
DBDATA1:
        .byte    0xaa,0xbb
```

3.4.2 DW, .hword

The RA78K4 uses the DW quasi-directive and the CA850 uses the .hword quasi-directive to reserve a 2-byte area of memory.

[RA78K4]

DW: Initializes a word area (2 bytes).
If the size is specified by the operand, an area of the specified size × 2 bytes is initialized by 0.
If the default value is specified by the operand, the area is initialized by the specified default value.
The following two description formats are available.

DW (size)
DW default value [, …]

<Example>

```
        CSEG
DWDATA0: DW       (1)
DWDATA1: DW       0CCCCH,0DDDDH
```

[CA850]

.hword: The first format of this quasi-directive reserves an area of 1 halfword for each operand and stores the lower halfword of the specified value in the reserved area.
The second format reserves an area of the specified bit width and stores the specified value in that area.
The following two description formats are available.

.hword value [, value …]
.hword bit width: value [, bit width: value …]

<Example>

```
        .sdata
DWDATA0:
        .space   2
DWDATA1:
        .hword   0xcccc,0xdddd
```

3.4.3 DG

The RA78K4 uses the DG quasi-directive to reserve a 3-byte area of memory. The CA850 does not have a quasi-directive that reserves a 3-byte area and supports only the .word quasi-directive that reserves a 4-byte area. For details of the .word quasi-directive, refer to **3.4.4 .word**.

[RA78K4]

DG: Initializes a 3-byte area.
If the size is specified by the operand, an area of the specified size × 3 bytes is initialized by 0.
If the default value is specified by the operand, the area is initialized by the specified default value.
The following two description formats are available.

DG (size)
DG default value [, …]

<Example>

```
        CSEG
DGDATA0: DG       (1)
DGDATA1: DG       0123456H,0789ABCH
```

[CA850]

Not provided

3.4.4 .word

The CA850 uses the .word quasi-directive to reserve a 4-byte area of memory.

[RA78K4]

Not provided

[CA850]

.word: The first format of this quasi-directive reserves an area of 1 word for each operand and stores the lower word of the specified area in that area.
The second format reserves an area of the specified bit width and stores the specified value in that area.
The following two description formats are available.

.word value [, value …]
.word bit width: value [, bit width: value …]

<Example>

```
        .sdata
DGDATA0:
        .space   4
DGDATA1:
        .word    0x123456,0x789abc
```

3.4.5 .space

The CA850 uses the .space quasi-directive to reserve memory of specified size and fill that area with the specified value. Filling the area with 0 is equivalent to specifying the size for DB, DW, or DG with the RA78K4.

[RA78K4]

Filling the specified area with 0 is equivalent to specifying the size for DB, DW, or DG.
For the description format and example, refer to the explanation of each quasi-directive.

[CA850]

.space: Reserves an area of the size specified by the first operand and fills the area with the filling value specified by the second operand (the default value is 0).
The description format is as follows. Refer to the explanation of DB, DW, and DG for an example.

.space size [, filling value]

<Example>

```
        .sdata
SPDATA:
        .space   4
```

3.4.6 .shword

The CA850 uses the .shword quasi-directive to reserve a 2-byte area and store a specified value shifted 1 bit to the right in that area.

This instruction is supported only by the V850E and is used when the switch instruction is used.

[RA78K4]

Not provided

[CA850] (V850E only)

.shword: The first format of this quasi-directive reserves an area of 1 halfword for each operand and stores the specified value shifted 1 bit to the right in that area.
The second format reserves an area of the specified bit width and stores the specified value shifted 1 bit to the right in that area.
The following two description formats are available.

.shword value [, value …]
.shword bit width: value [, bit width; value …]

<Example>

The value 0x2222, which is obtained as a result of shifting 0x4444 1 bit to the right, is stored in the area.

```
        .sdata
SHDATA0:
        .shword  0x4444
```

3.4.7 DS, .lcomm

The RA78K4 uses the DS quasi-directive and the CA850 uses the .lcomm quasi-directive to reserve memory of the specified number of bytes.

[RA78K4]

DS: Reserves a memory area of the number of bytes specified by the operand.
The description format is as follows.

DS absolute expression

<Example>

```
        DSEG
DSDATA0: DS       1
DSDATA1: DS       2
```

[CA850]

.lcomm: Aligns the location counter for the current section under the alignment condition specified by the third operand, reserves an area of the size specified by the second operand, and defines a local label having the label name specified by the first operand for the start address of that area.
The description format is as follows.

.lcomm label name, size, alignment condition

<Example>

```
        .sbss
        .lcomm   DSDATA0,1,1
        .lcomm   DSDATA1,2,2
```

3.4.8 DBIT

The RA78K4 defines reserving a 1-bit memory area using the DBIT quasi-directive.

The CA850 does not have a quasi-directive that reserves a 1-bit memory area. Reserve an area in a size such as bytes, and access it by specifying a bit position using the set1, clr1, or tst1 instruction.

[RA78K4]

DBIT: Reserves a 1-bit memory area in a bit segment.
The description format is as follows.

name DBIT

<Example>

```
        BSEG
DATAB0  DBIT
DATAB1  DBIT

        CSEG
        SET1     DATAB0
        CLR1     DATAB1
```

[CA850]

Not provided.
Reserve an area in a size such as bytes, and access it by specifying a bit position using the set1, clr1, or tst1 instruction.

<Example>

```
        .bss
        .lcomm   DATAB,0x1,1

        .text
        set1     0,$DATAB[gp]
        clr1     1,$DATAB[gp]
```

3.4.9 .float, .str

The CA850 uses the .float quasi-directive to reserve a floating-point value and the .str quasi-directive to reserve a character string. The RA78K4 does not have quasi-directives equivalent to these.

[RA78K4]

Not provided

[CA850]

.float: Reserves an area of 1 word for each operand and stores a floating-point value in that area.
The description format is as follows.

.float value [, value …]

.str: Reserves an area of the specified character string for each operand and stores the specified character string in that area.
The description format is as follows.

.str character string constant [, character string constant …]

<Example>

```
        .sdata
FDATA:
        .float   1.234
STRDATA:
        .str     "NEC"
```

3.5 Linkage Quasi-Directives (Program Linkage Quasi-Directives)

This section explains the quasi-directives related to linkage.

3.5.1 PUBLIC, .globl

The RA78K4 uses the PUBLIC quasi-directive and the CA850 uses the .globl quasi-directive to define that a symbol is a global symbol.

[RA78K4]

PUBLIC: Declares that the symbol described as the operand can be referenced by other modules.
The description format is as follows.

PUBLIC symbol name [, symbol name …]

<Example>

```
PUBLIC   PUBSYM
```

[CA850]

.globl: Declares an external label of the same name as the label name specified by the first operand. If the second operand is specified, the specified value is specified as the size of the data indicated by that label.
The description format is as follows.

.globl label name [, size]

<Example>

```
        .globl   PUBSYM
```

3.5.2 EXTRN, .extern

The RA78K4 uses the EXTRN quasi-directive and the CA850 uses the .extern quasi-directive to declare that a symbol is a global symbol defined by an other module.

[RA78K4]

EXTRN: Declares the symbol of an other module (except a bit symbol) referenced by this module.
The following three description formats are available.

EXTRN symbol name [, symbol name …]
EXTRN SADDR2 (symbol name [, symbol name …])
EXTRN BASE (symbol name [, symbol name …])

<Example>

```
EXTRN    EXSYM0
EXTRN    SADDR2(EXSYM1)
```

[CA850]

.extern: Declares a label name the same as the label name specified by the first operand as an external label name. If the second operand is specified, the specified value is specified as the size of the data indicated by that label.
The description format is as follows.

.extern label name [, size]

<Example>

```
        .extern  EXSYM0
        .extern  EXSYM1
```

3.5.3 EXTBIT

The RA78K4 uses the EXTBIT quasi-directive to declare that a symbol is a global bit symbol defined by an other module. The CA850 does not have an equivalent quasi-directive because it does not have bit symbols.

[RA78K4]

EXTBIT: Declares the bit symbol of an other module referenced by this module.
The following two description formats are available.

EXTBIT symbol name [, symbol name …]
EXTBIT SADDR2 (symbol name [, symbol name …])

<Example>

```
EXTBIT   EXBSYM0
EXTBIT   SADDR2(EXBSYM1)
```

[CA850]

Not provided.
Specify and reference an area of 1 byte or more using the .extern quasi-directive.

3.5.4 .comm

The CA850 uses the .comm quasi-directive to define an undefined external label.
The RA78K4 does not have a quasi-directive equivalent to .comm.

[RA78K4]

Not provided

[CA850]

.comm: Declares an undefined external label having the label name specified by the first operand, size specified by the second operand, and alignment condition specified by the third operand.
The description format is as follows.

.comm label name, size, alignment condition

<Example>

```
        .comm    EXSYM1,1,1
```

3.6 Automatic Selection Quasi-Directives

This section explains the automatic selection quasi-directives.

3.6.1 BR, CALL

In the RA78K4, the assembler can automatically select a branch instruction or a subroutine call instruction. The CA850 does not have quasi-directives equivalent to BR and CALL.

[RA78K4]

BR: The assembler automatically selects a BR branch instruction of 2 to 4 bytes according to the value of the expression specified by the operand and outputs a corresponding object code.
The description format is as follows.

BR expression

CALL: The assembler automatically selects a CALL instruction of 3 or 4 bytes according to the value of the expression specified by the operand and outputs a corresponding object code.
The description format is as follows.

CALL expression

<Example>

```
BR      JLABEL0
CALL    JLABEL1
```

[CA850]

Not provided.
Describe a branch instruction by using the jr or jarl instruction. However, use the jmp instruction for a branch that exceeds the displacement.

3.7 General-Purpose Register Selection Quasi-Directive

This section explains the general-purpose register selection quasi-directive.

3.7.1 RSS

The CA850 does not have a quasi-directive equivalent to the RSS quasi-directive.

[RA78K4]

RSS: Generates an object code based on the value of the register set selection flag specified by the operand and by replacing the general-purpose register of the function name described in the program with a general-purpose register of an absolute name.
The description format is as follows.

RSS 0 or 1

<Example>

```
RSS             0
```

[CA850]

Not provided.
The device does not have this function.

3.8 Macro Quasi-Directives (Macro, Skip, Repeat Assemble Quasi-Directives)

This section explains the macro quasi-directives.

3.8.1 MACRO, .macro

The RA78K4 uses the MACRO quasi-directive and the CA850 uses the .macro quasi-directive to define a macro.

[RA78K4]

MACRO: Defines a macro by giving a macro name to a series of statements described between the MACRO and ENDM quasi-directives.
The description format is as follows.

macro name MACRO [formal parameter [[, …]]

<Example>

```
ADDMAC   MACRO              PARA1,PARA2
         MOV       A,#PARA1
         ADD       A,#PARA2
         ENDM
```

[CA850]

.macro: Defines statements enclosed between the .macro and .endm quasi-directives as the macro body of the name specified by the first operand.
The description format is as follows.

.macro macro name [formal parameter,] …

<Example>

```
         .macro   ADDMAC   PARA1,PARA2
         mov      PARA1,r10
         add      PARA2,r10
         .endm
```

3.8.2 LOCAL, .local

The RA78K4 uses the LOCAL quasi-directive and the CA850 uses the .local quasi-directive to declare a local symbol in a macro.

[RA78K4]

LOCAL: Declares that a symbol name specified in the operand field is a local symbol that is valid only in the macro body.

The description format is as follows.

LOCAL symbol name [, …]

<Example>

```
JMAC    MACRO
        LOCAL   LAB
LAB:
        BR      $LAB
        ENDM
```

[CA850]

.local: Declares the specified character string as a local symbol that can be replaced by a specific identifier.

The description format is as follows.

.local Local symbol [, local symbol] …

<Example>

```
        .macro  JMAC
        .local  LAB
LAB:
        jr      LAB
        .endm
```

3.8.3 REPT, .repeat

The RA78K4 uses the REPT quasi-directive and the CA850 uses the .repeat quasi-directive to repeatedly expand a macro.

[RA78K4]

REPT: The assembler repeatedly expands the series of statements described between the REPT and ENDM quasi-directives the number of times specified by the value of the expression specified by the operand.
The description format is as follows.

REPT absolute expression

<Example>

```
        REPT    3
        NOP
        ENDM
```

[CA850]

.repeat: Repeatedly assembles the series of statements enclosed by the .repeat and .endm quasi-directives the number of times assigned by the absolute value expression specified by the first operand.
The description format is as follows.

.repeat absolute value expression

<Example>

```
        .repeat  3
        nop
        .endm
```

3.8.4 IRP, .irepeat

The RA78K4 uses the IRP quasi-directive and the CA850 uses the .irepeat quasi-directive to expand a macro repeatedly with a formal parameter.

[RA78K4]

IRP: Repeatedly expands the series of statements between the IRP and ENDM quasi-directives as many times as the number of actual parameters, replacing the formal parameters with the actual parameters specified by the first operand.
The description format is as follows.

IRP formal parameter, <actual parameter [, …]>

<Example>

```
IRP     PARA,<0AH,0BH,0CH>
ADD     A,#PARA
MOV     [HL],A
INCW    HL
ENDM
```

[CA850]

.irepeat: Repeatedly assembles statements enclosed by the .irepeat and .endm quasi-directives, replacing the formal parameters specified by the first operand with the actual parameters specified by the second operand.
The description format is as follows.

.irepeat formal parameter actual parameter [, actual parameter] …

<Example>

```
.irepeat PARA    0xa,0xb,0xc
add      PARA,r10
st.b     r10,[r11]
add      1,r11
.endm
```

3.8.5 EXITM, .exitm, .exitma

The RA78K4 uses the EXITM quasi-directive to forcibly stop expansion of a macro body or repeated expansion of a macro. The CA850 does not have a quasi-directive that forcibly stops expansion of a macro body. To forcibly stop repeated expansion of a macro, however, the .exitm or.exitma quasi-directive is used.

[RA78K4]

EXITM: Forcibly returns the nesting level of macro body expansion, REPT quasi-directive, or IRP quasi-directive to the nesting level when the macro body expansion, REPT quasi-directive, or IRP quasi-directive was executed.

The description format is as follows.

EXITM

<Example>

```
REPT     10
$_IF(SW1 < 5)
         DB       0FH
         EXITM
$ELSE
         DB       00H
$ENDIF
$_IF(SW1 > 10)
         DB       0AH
$ELSE
         DB       05H
$ENDIF
ENDM
```

[CA850]

.exitm: Skips repeated assembly of the innermost quasi-directive of those that enclose this quasi-directive. The description format is as follows.

.exitm

.exitma: Skips repeated assembly of the outermost quasi-directive of those that encloses this quasi-directive. The description format is as follows.

.extima

<Example>

```
        .sdata
        .repeat 10
        .if     SW1 < 5
                .byte   0xf
                .exitm
        .else
                .byte   0x0
        .endif
        .if     SW1 > 10
                .byte   0xa
        .else
                .byte   0x05
        .endif
        .endm

        .repeat 5
        .if     SW2 > 5
                .byte   0xf
                .if     SW2 < 10
                        .byte   0xa
                        .exitma
                .else
                        .byte   0x05
                .endif
        .else
                .byte   0x0
        .endif
        .endm
```

3.8.6 ENDM, .endm

The RA78K4 uses the ENDM quasi-directive and the CA850 uses the .endm quasi-directive to define the end of a macro.

[RA78K4]

ENDM: Informs the assembler of the end of a series of statements defined as a macro function.
The description format is as follows.

```
ENDM
```

<Example>

```
ADDMAC  MACRO             PARA1,PARA2
        MOV       A,#PARA1
        ADD       A,#PARA2
        ENDM
```

[CA850]

.endm: Indicates the end of a repeated zone or a macro body.
The description format is as follows.

```
.endm
```

<Example>

```
        .macro   ADDMAC   PARA1,PARA2
        mov      PARA1,r10
        add      PARA2,r10
        .endm
```

3.9 Assemble End Quasi-Directive

This section explains the assemble end quasi-directive.

3.9.1 END

The RA78K4 uses the END quasi-directive to declare the end of an assembler source module.

The CA850 does not have such a quasi-directive because it is not necessary to declare the end of the assembler module using a quasi-directive.

[RA78K4]

END: Declares the end of the source module to the assembler.
The description format is as follows.

```
END
```

[CA850]

Not provided.

It is not necessary to describe a keyword indicating the end at the end of the source file.

3.10 Assembler Target Model Specification Control Instructions (Assembler Control Quasi-Directives)

This section explains the assembler target model specification control instructions (assembler control quasi-directives).

3.10.1 $PROCESSOR, .option

The RA78K4 uses the $PROCESSOR control instruction and the CA850 uses the .option quasi-directive to describe the model to be assembled in the assembler source file. However, the .option quasi-directive has functions other than to specify the target model. For details, refer to **CA850 Assembly Language (U15027E)**.

[RA78K4]

$PROCESSOR: Specifies the model to be assembled.
The description format is as follows.

$PROCESSOR (model name)
$PC (model name)

<Example>

```
$PROCESSOR (4038)
```

[CA850]

.option: Controls the assembler in accordance with the option specified as the operand.
Specify cpu to specify the model to be assembled.
The description format is as follows.

.option cpu model name

<Example>

```
.option cpu 3003
```

3.11 Debug Information Output Control Instructions

This section explains the debug information output control instructions.

3.11.1 $DEBUG, $NODEBUG, $DEBUGA, $NODEBUGA

The RA78K4 can specify control of debug information on the source file by using a control instruction.

The CA850 does not have a quasi-directive corresponding to such a control instruction. Specify debug information using an option.

[RA78K4]

$DEBUG:	Outputs local symbol information.
$NODEBUG:	Does not output local symbol information.
$DEBUGA:	Outputs assembler source debug information.
$NODEBUGA:	Does not output assembler source debug information.

[CA850]

Not provided.

To output debug information, specify it using an option.

3.12 Cross-Reference List Output Specification Control Instructions

This section explains the cross-reference list output specification control instructions.

3.12.1 $XREF, $NOXREF, $SYMLIST, $NOSYMLIST

The RA78K4 has control instructions that control output of a cross-reference list and a symbol list.

The CA850 cannot output a cross-reference list. Reference cross-reference information by using cxref. Also, the CA850 cannot output a symbol list. Reference symbol information by using dump850 or rammap.

[RA78K4]

$XREF:	Outputs a cross-reference list.
$NOXREF:	Does not output a cross-reference list.
$SYMLIST:	Outputs a symbol list.
$NOSYMLIST:	Does not output a symbol list.

[CA850]

Not provided.

Reference a cross-reference list by using a cross-reference tool (cxref).

Reference symbols by using a dump tool (dump850) or memory visualization tool (rammap).

3.13 Include Control Instructions (File Input Control Quasi-Directives)

This section explains the include control instructions (file input control quasi-directives).

3.13.1 $INCLUDE, .include

The RA78K4 uses the $INCLUDE control instruction and the CA850 uses the .include quasi-directive to specify an include file.

[RA78K4]

$INCLUDE: Inserts, expands, and assembles the contents of a specified file.
The description format is as follows.

$INCLUDE (file name)
$IC (file name)

<Example>

```
$INCLUDE(MAIN.H)
```

[CA850]

.include: Treats the contents of the file specified by the operand as if it were placed at the position of this quasi-directive.
The description format is as follows.

.include “file name”

<Example>

```
.include        "main.h"
```

3.13.2 .binclude

The CA850 can include a binary file.

[RA78K4]

Not provided

[CA850]

.binclude: Places the contents of the file specified by the operand at the position of this quasi-directive as binary data unchanged.
The description format is as follows.

.binclude “file name”

<Example>

```
.binclude       "sub.o"
```

3.14 Assemble List Control Instructions

This section explains the assemble list control instructions.

3.14.1 $EJECT, $TITLE, $SUBTITLE, $LIST, $NOLIST, $GEN, $NOGEN, $COND, $NOCOND, $FORMFEED, $NOFORMFEED, $WIDTH, $LENGTH, $TAB

The RA78K4 has control instructions that control the number of characters on one line and the number of lines on one page of an assemble list.

The CA850 does not have an instruction that controls the assemble list.

[RA78K4]

$EJECT:	Instructs the assembler to execute a page break of the assemble list.
$TITLE:	Specifies a character string to be printed in the title field of the header of each page of the assemble list.
$SUBTITLE:	Specifies a character string to be printed as the subtitle on each page header of the assemble list.
$LIST:	Informs the assembler of the output start position of the assemble list.
$NOLIST:	Informs the assembler of the output stop position of the assemble list.
$GEN:	Instructs to output macro expansion to the assemble list.
$NOGEN:	Instructs not to output macro expansion to the assemble list.
$COND:	Instructs to output the conditions not satisfied in conditional assembly to the assemble list.
$NOCOND:	Instructs not to output the conditions not satisfied in conditional assembly to the assemble list.
$FORMFEED:	Instructs to output form feed at the end of the list file.
$NOFORMFEED:	Instructs not to output form feed at the end of the list file.
$WIDTH:	Indicates the maximum number of characters on one line of a list file.
$LENGTH:	Indicates the number of lines on one page of a list file.
$TAB:	Indicates the number of characters to be expanded of the tab of a list file.

[CA850]

Not provided.

The assemble list file cannot be manipulated.

3.15 Conditional Assembly Control Instructions (Conditional Assembly Quasi-Directives)

This section explains the conditional assembly control instructions (conditional assembly quasi-directives).

3.15.1 $SET, $RESET

The RA78K4 uses the $SET control instruction and $RESET quasi-directive to define a value for a switch name of conditional assembly.

The CA850 does not have such a control instruction. Define a switch name by using the .set quasi-directive.

[RA78K4]

$SET: Gives a true value (0FFH) to a switch name specified by the IF/ELSEIF control instruction. The description format is as follows.

$SET (switch name [: switch name …])

$RESET: Gives a false value (0H) to a switch name specified by the IF/ELSEIF control instruction. The description format is as follows.

$RESET (switch name [: switch name …])

<Example>

```
$SET     (SWSYM0)
$RESET   (SWSYM1)
```

[CA850]

Not provided.

Use the .set quasi-directive.

<Example>

```
        .set     SWSYM0,0xff
        .set     SWSYM1,0
```

3.15.2 $IF, .ifdef

The RA78K4 performs assembly if the switch name specified by the $IF control instruction is true.
The CA850 performs assembly if the switch name specified by the .ifdef quasi-directive is defined.

[RA78K4]

$IF: Assembles until the next conditional assembly quasi-directive appears if the specified switch name is true (≠ 0). If it is false (= 0), does not assemble until the next conditional assembly quasi-directive appears.
The description format is as follows.

$IF (switch name [: switch name …])

<Example>

```
$IF(SWSYM0)
```

[CA850]

.ifdef: Assembles the blocks up to the corresponding .else, .elseif, .elseifn, or .endif quasi-directive if the name specified by the operand is defined. If not, does not assemble the blocks up to the corresponding .else, .elseif, .elseifn, or .endif quasi-directive.
The description format is as follows.

.ifdef name

<Example>

```
        .ifdef   SWSYM0
```

3.15.3 .ifndef

The CA850 performs assembly if the switch name specified by the .ifndef quasi-directive is not defined.

[RA78K4]

Not provided

[CA850]

.ifndef: Assembles the blocks up to the corresponding .else, elseif, elseifn, or .endif quasi-directive if the name specified by the operand is not defined. If it is defined, does not assemble the blocks up to the corresponding .else, .elseif, .elseifn, or .endif quasi-directive.
The description format is as follows.

.ifndef name

<Example>

```
        .ifndef  SWSYM3
```

3.15.4 $_IF, .if

The RA78K4 performs assembly if the conditional expression of the $_IF control instruction is true.
The CA850 performs assembly if the expression of the .if quasi-directive is true.

[RA78K4]

$_IF: Assembles until the next conditional assembly quasi-directive appears if the specified conditional expression is true (≠ 0). If it is false (= 0), does not assemble until the next conditional assembly quasi-directive appears.
The description format is as follows.

$_IF conditional expression

<Example>

```
$_IF SWSYM2 = 0
```

[CA850]

.if: Assembles the blocks up to the corresponding .else, .elseif, .elseifn, or .endif quasi-directive if the absolute expression specified by the operand is true (≠ 0). If it is false (= 0), does not assemble the blocks up to the corresponding .else, .elseif, .elseifn, or .endif quasi-directive.
The description format is as follows.

.if absolute expression

<Example>

```
        .if     SWSYM2==0
```

3.15.5 .ifn

The CA850 performs assembly if the expression of the .ifn quasi-directive is false.

[RA78K4]

Not provided.

[CA850]

.ifn: Assembles the blocks up to the corresponding .else, .elseif, .elseifn, or .endif quasi-directive if the absolute expression specified by the operand is false (= 0). If it is true (≠ 0), does not assemble the blocks up to the corresponding .else, .elseif, .elseifn, or .endif quasi-directive.
The description format is as follows.

.ifn absolute expression

<Example>

```
        .ifn    SWSYM2==5
```

3.15.6 $ELSEIF, $_ELSEIF, .elseif

The RA78K4 uses the $ELSEIF control instruction for symbols and the $_ELSEIF control instruction for expressions and the CA850 uses the .elseif quasi-directive to assemble if a condition is true when the conditions of the previous assembly control instructions are not satisfied.

[RA78K4]

$ELSEIF: Judges a condition only if the conditions of all the previously described conditional assembly control instructions are not satisfied.
If the specified switch name is true (≠ 0), assembles until the next conditional assembly quasi-directive appears. If it is false (= 0), does not assemble until the next conditional assembly quasi-directive appears.
The description format is as follows.

$ELSEIF (switch name [: switch name …])

$_ELSEIF: Judges a condition only if the conditions of all the previously described conditional assembly control instructions are not satisfied.
If the specified condition expression is true (≠ 0), assembles until the next conditional assembly quasi-directive appears. If it is false (= 0), does not assemble until the next conditional assembly quasi-directive appears.
The description format is as follows.

$_ELSEIF conditional expression

<Example>

```
$ELSEIF(SWSYM1)
$_ELSEIF         SWSYM2 = 0FFH
```

[CA850]

.elseif: Judges a condition only if the conditions of all the previously described conditional assembly quasi-directives are not satisfied.
If the absolute expression specified by the operand is true (≠ 0), assembles the blocks up to the corresponding .else, .elseif, .elseifn, or .endif quasi-directive. If it is false (= 0), does not assemble the blocks up to the corresponding .else, .elseif, .elseifn, or .endif quasi-directive.
The description format is as follows.

.elseif absolute expression

<Example>

```
        .elseif  SWSYM2==0xff
```

3.15.7 .elseifn

The CA850 uses the .elseifn quasi-directive to judge a condition if the conditions of all the previous conditional assembly control instructions are not satisfied, and assembles if the expression specified by the operand is false.

[RA78K4]

Not provided.

[CA850]

.elseifn: Judges a condition only if the condition of all the previously described conditional assembly quasi-directives are not satisfied. If the absolute expression specified by the operand is false (= 0), assembles the blocks up to the corresponding .else, .elseif, .elseifn, or .endif quasi-directive. If it is true (≠ 0), does not assemble the blocks up to the corresponding .else, .elseif, .elseifn, or .endif quasi-directive.
The description format is as follows.

.elseifn absolute expression

<Example>

```
        .elseifn SWSYM2==0xff
```

3.15.8 $ELSE, .else

The RA78K4 uses the $ELSE control instruction and the CA850 uses the .else quasi-directive to specify assembling if all previous conditional assembly control instructions are false.

[RA78K4]

$ELSE: If the conditions of all previously described conditional assembly control instructions are not satisfied, assembles until the ENDIF control instruction appears after the ELSE control instruction.
The description format is as follows.

$ELSE

[CA850]

.else: If the conditions of all the previously described conditional assembly quasi-directives are not satisfied, assembles statements enclosed by the .else quasi-directive and .endif quasi-directive corresponding to this quasi-directive.
The description format is as follows.

.else

3.15.9 $ENDIF, .endif

The RA78K4 uses the $ENDIF control instruction and the CA850 uses the .endif quasi-directive to define the end of conditional assembly.

[RA78K4]

$ENDIF: Informs the assembler of the end of the source statement subject to conditional assembly.
The description format is as follows.

```
$ENDIF
```

[CA850]

.endif: Indicates the end of the range of control by a conditional assembly quasi-directive.
The description format is as follows.

```
.endif
```

3.16 SFR Area Change Control Instructions

This section explains the SFR area change control instructions.

3.16.1 $CHGSFR, $CHGSFRA

The 78K/IV Series can change the address of the SFR area by using the LOCATION instruction. At this time, the SFR area whose address is to be changed is specified by the SFR area change control instruction.

The V850 Family does not have such a function and therefore does not have a quasi-directive equivalent to the SFR area change control instruction.

[RA78K4]

$CHGSFR: Specifies an address of the SFR area by the absolute expression of the operand.
The description format is as follows.

```
$CHGSFR  (absolute expression)
```

$CHGSFRA: Instructs to create an object that can be linked regardless of the SFR area.
The description format is as follows.

```
$CHGSFRA
```

[CA850]

Not provided

CHAPTER 4 LINK DIRECTIVES

This chapter explains the points to be noted about the link directives of the linker (link editor) and the basic method of describing the link directives.

4.1 Contents of Link Directive

The contents described in the link directive file of the lk78k4 of the RA78K4 and the ld850 of the CA850 are as follows.

Table 4-1. Link Directives

RA78K4	CA850
Memory directive Segment allocation directive	Segment directive Mapping directive Symbol directive

In the RA78K4, the directives do not have to be described in the order of addresses, whereas they do in the CA850.

The RA78K4 does not have a symbol directive that generates a symbol. The CA850 requires description of a symbol directive to generate the TP, GP, and EP symbols.

4.2 Description of Link Directive

The description format of the link directives of the RA78K4 and CA850 is as follows.

[RA78K4]

Memory directive

MEMORY memory area name: (start address value, size) [/memory space name]

Segment allocation directive

MERGE segment name: [AT (start address)]
[= memory area name specification] [/memory space name]

<Example>

```
memory TBL        : ( 000000h , 00080h )
memory ROM        : ( 000080h , 0BF80h )
memory RAM1       : ( 0FF700h , 00600h )
memory STK        : ( 0FFD00H, 00020h )
memory SDR        : ( 0FFD20h , 000E0h )
memory SDR1       : ( 0FFE00h , 00080h )
memory RAM        : ( 0FFE80h , 00180h )

merge @@INIT      : = RAM1
merge @@DATA      : = RAM1
merge @@INIS      : = SDR
merge @@DATS      : = SDR
merge @@BITS      : = SDR
merge @@INIS1     : = SDR1
merge @@DATS1     : = SDR1
merge @@BITS1     : = SDR1

memory EXMEM : (050000H, 1000H)

merge  DAT1 := EXMEM
```

[CA850]

Segment directive

Segment name: !segment type ?segment attribute [V address]
[L maximum memory size] [H hole size] [F filling value]
[A alignment condition] {mapping directive};

Mapping directive

section name = $section type ?section attribute [section name]
[V address] [H hole size] [A alignment condition]
[{file name [file name]…}];

Symbol directive

symbol name @%symbol type [&base symbol name] [V address]
[A alignment condition] [{segment name[segment name]…}];

<Example>

```
SCONST   : !LOAD ?R {
  .sconst          = $PROGBITS       ?A .sconst;
};

TEXT     : !LOAD ?RX {
  .pro_epi_runtime = $PROGBITS       ?AX;
  .text             = $PROGBITS      ?AX;
};

DATA     : !LOAD ?RW V0x100000 {
  .data            = $PROGBITS       ?AW.data;
  .sdata           = $PROGBITS       ?AWG.sdata;
  .sbss            = $NOBITS         ?AWG.sbss;
  .bss             = $NOBITS         ?AW.bss;
};

CONST    : !LOAD ?R {
  .const           = $PROGBITS       ?A .const;
};

SEDATA   : !LOAD ?RW V0xff6000 {
  .sedata          = $PROGBITS       ?AW .sedata;
  .sebss           = $NOBITS         ?AW .sebss;
};

SIDATA   : !LOAD ?RW V0xffe000 {
  .tidata.byte     = $PROGBITS       ?AW .tidata.byte;
  .tibss.byte      = $NOBITS         ?AW .tibss.byte;
  .tidata.word     = $PROGBITS       ?AW .tidata.word;
  .tibss.word      = $NOBITS         ?AW .tibss.word;
  .tidata          = $PROGBITS       ?AW .tidata;
  .tibss           = $NOBITS         ?AW .tibss;
  .sidata          = $PROGBITS       ?AW .sidata;
  .sibss           = $NOBITS         ?AW .sibss;
};

__tp_TEXT @ %TP_SYMBOL;
__gp_DATA @ %GP_SYMBOL &__tp_TEXT{DATA};
__ep_DATA @ %EP_SYMBOL;
```

CHAPTER 5 TRANSLATION LIMIT

This chapter compares the translation limit values of the CC78K4 and CA850 during compilation. The results of the comparison are as follows.

Table 5-1. Translation Limit Value

No.	Item	CC78K4	CA850
1	Nesting of compound, repeat control, and selection control statements	45	127
2	Nesting of conditional compiling	255	255
3	Number of pointers, arrays, and function declarators (or any combination of these) qualifying one arithmetic type, structure type, union type, or incomplete type in one declaration,	12	16
4	Number of levels of nesting declarators enclosed in parentheses in a complete declarator	591	255
5	Number of levels of nesting expressions enclosed in parentheses in a complete expression	32	255
6	Number of valid first characters in a macro name	31	1023
7	Number of valid first characters in an external symbol name	30	1022
8	Number of valid first characters in an internal symbol name	30	1023
9	Number of symbols in one source module file	1024	4095
10	Number of symbols having block scope in one block	255	
11	Number of macros in one source module file	10000	2047[Note]
12	Parameter of one function definition or one function call	39	255
13	Parameter of one macro definition or one macro call	31	127
14	Number of characters on one logic source line	509	32766
15	Number of characters in a character string literal after linking	509	32766
16	Nesting of include files	8	50
17	Number of case labels of a switch statement	257	1025
18	Number of members of one structure or union	127	1023
19	Number of enumeration constants of one enumeration	255	1023
20	Nesting of structures or unions in one structure or union	15	63

Note This can be changed by a C compiler option (-Xm) (up to 32767).

APPENDIX INDEX

[M]

[N]

[O]

[P]

[Q]

[R]

[S]

[T]

[U]

[V]

[W]

[X]

NEC

Facsimile Message

From:

Name

Company

Tel. FAX

Address

Although NEC has taken all possible steps to ensure that the documentation supplied to our customers is complete, bug free and up-to-date, we readily accept that errors may occur. Despite all the care and precautions we've taken, you may encounter problems in the documentation. Please complete this form whenever you'd like to report errors or suggest improvements to us.

Thank you for your kind support.

North America
NEC Electronics Inc.
Corporate Communications Dept.
Fax: +1-800-729-9288
+1-408-588-6130

Europe
NEC Electronics (Europe) GmbH
Technical Documentation Dept.
Fax: +49-211-6503-274

South America
NEC do Brasil S.A.
Fax: +55-11-6462-6829

Hong Kong, Philippines, Oceania
NEC Electronics Hong Kong Ltd.
Fax: +852-2886-9022/9044

Korea
NEC Electronics Hong Kong Ltd.
Seoul Branch
Fax: +82-2-528-4411

Taiwan
NEC Electronics Taiwan Ltd.
Fax: +886-2-2719-5951

Asian Nations except Philippines
NEC Electronics Singapore Pte. Ltd.
Fax: +65-250-3583

Japan
NEC Semiconductor Technical Hotline
Fax: +81- 44-435-9608

I would like to report the following error/make the following suggestion:

Document title: _____

Document number: _____ Page number: _____

If possible, please fax the referenced page or drawing.

Document Rating	Excellent	Good	Acceptable	Poor
Clarity	❑	❑	❑	❑
Technical Accuracy	❑	❑	❑	❑
Organization	❑	❑	❑	❑