To our customers,

Old Company Name in Catalogs and Other Documents

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

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

April 1st, 2010
Renesas Electronics Corporation

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

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

Notice

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

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

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

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

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

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

RENESAS

Application Note

CA850 Ver.2.50

C Compiler Package

Coding Technique

Target Devices
V850 Series™

Document No. U16076EJ1V0AN00 (1st edition)
Date Published August 2002 N CP(K)

[MEMO]

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

NEC Electronics (Europe) GmbH
Duesseldorf, Germany
Tel: 0211-65 03 01
Fax: 0211-65 03 327

- **Sucursal en España**
  Madrid, Spain
  Tel: 091-504 27 87
  Fax: 091-504 28 60

- **Succursale Française**
  Vélizy-Villacoublay, France
  Tel: 01-30-67 58 00
  Fax: 01-30-67 58 99

- **Filiale Italiana**
  Milano, Italy
  Tel: 02-66 75 41
  Fax: 02-66 75 42 99

- **Branch The Netherlands**
  Eindhoven, The Netherlands
  Tel: 040-244 58 45
  Fax: 040-244 45 80

- **Branch Sweden**
  Taeby, Sweden
  Tel: 08-63 80 820
  Fax: 08-63 80 388

- **United Kingdom Branch**
  Milton Keynes, UK
  Tel: 01908-691-133
  Fax: 01908-670-290

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 Shanghai, Ltd.
Shanghai, P.R. China
Tel: 021-6841-1138
Fax: 021-6841-1137

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

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

J02.4

INTRODUCTION

Target Readers

This manual is intended for users who wish to design and develop application systems using the V850 Series.

Purpose

This manual is intended to give users an understanding of the coding techniques for increasing execution speed or reducing code size after specifying an optimization option using the V850 Series C compiler CA850.

Organization

This manual is divided into the following parts.

- Overview
- Reducing code size
- Increasing execution speed
- Defining variables

How to Read This Manual

It is assumed that the reader of this manual has general knowledge in the fields of electrical engineering, logic circuits, microcontrollers, C language, and assembler programming.

To know the hardware functions of the V850 Series
→ Refer to the hardware user's manual of each product

To know the instruction functions of the V850 Series
→ Refer to **V850 Series Architecture User's Manual**

Conventions

Data significance:	Higher digits on the left and lower digits on the right
Note:	Footnote for item marked with **Note** in the text
Caution:	Information requiring particular attention
Remark:	Supplementary information
Numeric representation:	Binary ... XXXX or XXXXB
	Decimal ... XXXX
	Hexadecimal ... XXXXH

Prefix indicating the power of 2 (address space, memory capacity):

K (kilo):	$2^{10} = 1024$
M (mega):	$2^{20} = 1024^2$
G (giga):	$2^{30} = 1024^3$

Related Documents Refer to the following manuals when using this manual.
The related documents indicated in this publication may include preliminary versions. However, preliminary versions are not marked as such.

Documents related to the V850 Series (user's manuals)

Document Name		Document No.
IE-703002-MC (In-Circuit Emulator for V853™, V850/SA1™, V850/SB1™, V850/SB2™, V850/SF1™, V850/SV1™)		U11595E
IE-V850E-MC (In-Circuit Emulator for V850E/IA1™, V850E/IA2™) IE-V850E-MC-A (In-Circuit Emulator for V850E/MA1™, V850E/MA2™)		U14487E
IE-703003-MC-EM1 (In-Circuit Emulator Option Board for V853)		U11596E
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™)		U13875E
IE-703102-MC-EM1, IE-703102-MC-EM1-A (In-Circuit Emulator Option Board for V850E/MS1)		U13876E
IE-703107-MC-EM1 (In-Circuit Emulator Option Board for V850E/MA1)		U14481E
IE-703116-MC-EM1 (In-Circuit Emulator Option Board for V850E/IA1)		U14700E
CA850 Ver.2.50 C Compiler Package	Operation	U16053E
	C Language	U16054E
	PM plus	U16055E
	Assembly Language	U16042E
ID850 Ver.2.40 Integrated Debugger	Operation Windows™ Based	U15181E
SM850 Ver.2.40 System Emulator	Operation Windows Based	U15182E
SM850 Ver.2.00 or Later System Emulator	External Part User Open Interface Specifications	U14873E
RX850 Ver.3.13 or Later Real-time OS	Basics	U13430E
	Installation	U13410E
	Technical	U13431E
RX850 Pro Ver.3.13 or Later Real-time OS	Basics	U13773E
	Installation	U13774E
	Technical	U13772E
RD850 Ver.3.01 Task Debugger		U13737E
RD850 Pro Ver.3.01 Task Debugger		U13916E
AZ850 Ver.3.10 System Performance Analyzer		U14410E
PG-FP4 Flash Memory Programmer		U15260E
CA850 Ver.2.50 C Compiler Package (Application Note)	Coding Technique	This manual

CONTENTS

CHAPTER 1 OVERVIEW

This application note explains the coding techniques for further decreasing the code size or increasing the effective execution speed after an optimization option was specified using the CA850, which is the C compiler for the V850 Series.

For details concerning the CA850, refer to CA850-related user's manuals.

The amount that the coding size is reduced for the examples in each section is specific to those examples. The amount that the coding size will be reduced when that technique is applied elsewhere will differ somewhat depending on the individual case.

Also, the following points should be noted when modifying source code.

Since the register usage conditions will vary depending on the source code modification, register transfers that remain without having been eliminated by the optimization up to that point may be removed or, conversely, optimization that was performed may no longer be effective and lengthy register transfers may remain.

By adding temporary variables, registers for the new register variables may be used and this may cause coding for saving/restoring those registers to be added at function entries/exits. In this case, the code size will be increased by the amount of code required for saving/restoring the registers.

The output assembly list for the examples that appear in this application note shows assembler source code that was compiled by specifying size-priority optimization (-Os). Note that the results will differ when optimization other than size priority optimization is specified.

CHAPTER 2 REDUCING CODE SIZE

This chapter introduces coding techniques for reducing the code size.

Note that reducing the code size may also increase the effective execution speed.

2.1 Using the if~else Statement Instead of the switch Statement

If the following two conditions are satisfied simultaneously, the CA850 generates table branching format code for the switch statement.

- The number of case labels is at least four.
- The difference between the upper and lower limits of the label value is at most three times the number of case labels.

In this case, if the number of case labels is roughly 16 or less (however, this number differs according to the format of the switch expression or the distribution of label values), switching to the equivalent if~else statements and arranging a sequence of compare and branch instructions will produce a smaller code size.

If the switch expression is an external variable reference or complicated expression, you must assign a value to a temporary variable and change to a reference of that temporary variable in the if expression.

Note, however, that since the switch instruction is output for the V850E, the switch statement will produce a smaller code size.

Caution Even if the source description is not change the expansion code of a switch statement can be specified per file using the -Xcase option.

A sample program is shown below.

Remark x is assumed to be an auto variable.

Before Modification	After Modification

`int x;`
 	
`switch(x) {`
`  case 0:`
`    return(0);`
`  case 1:`
`    return(1);`
`  case 2:`
`    return(2);`
`  case 3:`
`    return(3);`
`  case 4:`
`    return(4);`
`  case 5:`
`    return(5);`
`}`
	`int x;`
	
	`if (x == 0)`
	`  return(0);`
	`else if (x == 1)`
	`  return(1);`
	`else if (x == 2)`
	`  return(2);`
	`else if (x == 3)`
	`  return(3);`
	`else if (x == 4)`
	`  return(4);`
	`else if (x == 5)`
	`  return(5);`

[Output Assembly List for the V850]

Before Modification		After Modification	
Program	Size [bytes]	Program	Size [bytes]
`ld.w -8+0x8[sp], r10`	4	`ld.w -8+0x8[sp], r10`	4
`cmp 5, r10`	2	`cmp r0, r10`	2
`jh .L1`	2	`jne .L2`	2
`shl 1, r10`	2	`st.w r0, -4+0x8[sp]`	4
`add tp, r10`	2	`jbr .L1`	2
`ld.h .L10[r10], r11`	8	`.L2:`	
`add .L10, r11`	10	`cmp 1, r10`	2
`add tp, r11`	2	`jne .L4`	2
`jmp [r11]`	2	`mov 1, r11`	2
`.L10:`		`st.w r11, -4+0x8[sp]`	4
`.hword .L4-.L10`	2	`jbr .L1`	2
`.hword .L5-.L10`	2	`.L4:`	
`.hword .L6-.L10`	2	`cmp 2, r10`	2
`.hword .L7-.L10`	2	`jne .L6`	2
`.hword .L8-.L10`	2	`mov 2, r12`	2
`.hword .L9-.L10`	2	`st.w r12, -4+0x8[sp]`	4
`.L4:`		`jbr .L1`	2
`st.w r0, -4+0x8[sp]`	4	`.L6:`	
`jbr .L1`	2	`cmp 3, r10`	2
`.L5:`		`jne .L8`	2
`mov 1, r12`	2	`mov 3, r13`	2
`st.w r12, -4+0x8[sp]`	4	`st.w r13, -4+0x8[sp]`	4
`jbr .L1`	2	`jbr .L1`	2
`.L6:`		`.L8:`	
`mov 2, r13`	2	`cmp 4, r10`	2
`st.w r13, -4+0x8[sp]`	4	`jne .L10`	2
`jbr .L1`	2	`mov 4, r14`	2
`.L7:`		`st.w r14, -4+0x8[sp]`	4
`mov 3, r14`	2	`jbr .L1`	2
`st.w r14, -4+0x8[sp]`	4	`.L10:`	
`jbr .L1`	2	`cmp 5, r10`	2
`.L8:`		`jne .L1`	2
`mov 4, r15`	2	`mov 5, r15`	2
`st.w r15, -4+0x8[sp]`	4	`st.w r15, -4+0x8[sp]`	4
`jbr .L1`	2	`.L1:`	
`.L9:`		`ld.w -4+0x8[sp], r10`	4
`mov 5, r16`	2		
`st.w r16, -4+0x8[sp]`	4		
`.L1:`			
`ld.w -4+0x8[sp], r10`	4		
Total code size	94 bytes	Total code size	76 bytes

[Output Assembly List for the V850E]

Before Modification		After Modification	
Program	Size [bytes]	Program	Size [bytes]
`ld.w -8+0x8[sp], r10`	4	`ld.w -8+0x8[sp], r10`	4
`cmp 5, r10`	2	`cmp r0, r10`	2
`jh .L1`	2	`jne .L2`	2
`switch r10`	2	`st.w r0, -4+0x8[sp]`	4
`.L10:`		`jbr .L1`	2
`.shword .L4-.L10`	2	`.L2:`	
`.shword .L5-.L10`	2	`cmp 1, r10`	2
`.shword .L6-.L10`	2	`jne .L4`	2
`.shword .L7-.L10`	2	`mov 1, r11`	2
`.shword .L8-.L10`	2	`st.w r11, -4+0x8[sp]`	4
`.shword .L9-.L10`	2	`jbr .L1`	2
`.L4:`		`.L4:`	
`st.w r0, -4+0x8[sp]`	4	`cmp 2, r10`	2
`jbr .L1`	2	`jne .L6`	2
`.L5:`		`mov 2, r12`	2
`mov 1, r12`	2	`st.w r12, -4+0x8[sp]`	4
`st.w r12, -4+0x8[sp]`	4	`jbr .L1`	2
`jbr .L1`	2	`.L6:`	
`.L6:`		`cmp 3, r10`	2
`mov 2, r13`	2	`jne .L8`	2
`st.w r13, -4+0x8[sp]`	4	`mov 3, r13`	2
`jbr .L1`	2	`st.w r13, -4+0x8[sp]`	4
`.L7:`		`jbr .L1`	2
`mov 3, r14`	2	`.L8:`	
`st.w r14, -4+0x8[sp]`	4	`cmp 4, r10`	2
`jbr .L1`	2	`jne .L10`	2
`.L8:`		`mov 4, r14`	2
`mov 4, r15`	2	`st.w r14, -4+0x8[sp]`	4
`st.w r15, -4+0x8[sp]`	4	`jbr .L1`	2
`jbr .L1`	2	`.L10:`	
`.L9:`		`cmp 5, r10`	2
`mov 5, r16`	2	`jne .L1`	2
`st.w r16, -4+0x8[sp]`	4	`mov 5, r15`	2
`.L1:`		`st.w r15, -4+0x8[sp]`	4
`ld.w -4+0x8[sp], r10`	4	`.L1:`	
		`ld.w -4+0x8[sp], r10`	4
Total code size	70 bytes	Total code size	76 bytes

2.2 Making Assignments to the Same External Variable Via a Temporary Variable at the Branch Destinations of the switch Statement or if~else Statement

To assign a different value to the same external variable at each branch destination of the switch statement or if~else statement, the code size can be reduced by assigning a temporary variable at each location, and then assigning the value to the original external variable from the temporary variable after the control logic has merged together again.

The code size is reduced for the following reason. Since external variables are rarely allocated to registers, an assignment to an external variable becomes a store instruction to memory (4 bytes). However, an assignment to a temporary variable often becomes a register transfer (2 bytes).

A sample program is shown below.

Remark s is assumed to be an external variable.

Before Modification	After Modification

`int x;`
 	
`switch (x) {`
`  case 0:`
`    s = 0;`
`    break;`
`  case 1000:`
`    s = 0x5555;`
`    break;`
`  case 2000:`
`    s = 0xAAAA;`
`    break;`
`  case 3000:`
`    s = 0xFFFF;`
`}`
	`int x;`
	`int tmp;`
	
	`if (x == 0) {`
	`  tmp = 0;`
	`}`
	`else if (x == 1000) {`
	`  tmp = 0x5555;`
	`}`
	`else if (x == 2000) {`
	`  tmp = 0xAAAA;`
	`}`
	`else if (x == 3000) {`
	`  tmp = 0xFFFF;`
	`}`
	`else {`
	`  goto label;`
	`}`
	`s = tmp;`
	`label:`
	`;`

[Output Assembly List for the V850]

Before Modification		After Modification	
Program	Size [bytes]	Program	Size [bytes]
`ld.w -4+0x4[sp], r10`	4	`ld.w -4+0x4[sp], r10`	4
`cmp r0, r10`	2	`cmp r0, r10`	2
`je .L4`	2	`jne .L2`	2
`addi -1000, r10, r0`	4	`mov r0, r10`	2
`je .L5`	2	`jbr .L3`	2
`addi -2000, r10, r0`	4	`.L2:`	
`je .L6`	2	`addi -1000, r10, r0`	4
`addi -3000, r10, r0`	4	`jne .L4`	2
`je .L7`	2	`mov 21845, r10`	4
`jbr .L3`	2	`jbr .L3`	2
`.L4:`		`.L4:`	
`st.w r0, $_s`	4	`addi -2000, r10, r0`	4
`jbr .L3`	2	`jne .L6`	2
`.L5:`		`mov 43690, r10`	8
`mov 21845, r11`	4	`jbr .L3`	2
`st.w r11, $_s`	4	`.L6:`	
`jbr .L3`	2	`addi -3000, r10, r0`	4
`.L6:`		`jne .L10`	2
`mov 43690, r12`	8	`mov 65535, r10`	8
`st.w r12, $_s`	4	`.L3:`	
`jbr .L3`	2	`st.w r10, $_s`	4
`.L7:`		`.L10:`	
`mov 65535, r13`	8		
`st.w r13, $_s`	4		
`.L3:`			
Total code size	70 bytes	Total code size	58 bytes

[Output Assembly List for the V850E]

Before Modification		After Modification	
Program	Size [bytes]	Program	Size [bytes]
`ld.w -4+0x4[sp], r10`	4	`ld.w -4+0x4[sp], r10`	4
`cmp r0, r10`	2	`cmp r0, r10`	2
`je .L4`	2	`jne .L2`	2
`addi -1000, r10, r0`	4	`mov r0, r10`	2
`je .L5`	2	`jbr .L3`	2
`addi -2000, r10, r0`	4	`.L2:`	
`je .L6`	2	`addi -1000, r10, r0`	4
`addi -3000, r10, r0`	4	`jne .L4`	2
`je .L7`	2	`mov 21845, r10`	4
`jbr .L3`	2	`jbr .L3`	2
`.L4:`		`.L4:`	
`st.w r0, $_s`	4	`addi -2000, r10, r0`	4
`jbr .L3`	2	`jne .L6`	2
`.L5:`		`mov 43690, r10`	6
`mov 21845, r11`	4	`jbr .L3`	2
`st.w r11, $_s`	4	`.L6:`	
`jbr .L3`	2	`addi -3000, r10, r0`	4
`.L6:`		`jne .L10`	2
`mov 43690, r12`	6	`mov 65535, r10`	6
`st.w r12, $_s`	4	`.L3:`	
`jbr .L3`	2	`st.w r10, $_s`	4
`.L7:`		`.L10:`	
`mov 65535, r13`	6		
`st.w r13, $_s`	4		
`.L3:`			
Total code size	66 bytes	Total code size	54 bytes

2.3 Moving One Assignment Statement in Front of the if Statement When the Branch Destinations of the if~else Statement Are Only Assignment Statements to the Same Variable

When the branch destinations of the if~else statement are only statements that assign different values to the same variable, the code size can be reduced by moving one of the assignment statements in front of the if statement and removing the else block, which eliminates the jump statement to the location following the else block from the if block.

A sample program is shown below.

Remark s is assumed to be an external variable.

Before Modification	After Modification

`int x;`
 	
`if (x == 10) {`
`  s = 1;`
`}`
`else {`
`  s = 0;`
`}`
	`int x;`
	
	`s = 0;`
	`if (x == 10) {`
	`  s = 1;`
	`}`

[Output Assembly List for the V850]

Before Modification		After Modification	
Program	Size [bytes]	Program	Size [bytes]
`ld.w -4+0x4[sp], r10`	4	`st.w r0, $_s`	4
`cmp 10, r10`	2	`ld.w -4+0x4[sp], r10`	4
`jne .L2`	2	`cmp 10, r10`	2
`mov 1, r11`	2	`jne .L2`	2
`st.w r11, $_s`	4	`mov 1, r11`	2
`jbr .L3`	2	`st.w r11, $_s`	4
`.L2:`		`.L2:`	
`st.w r0, $_s`	4		
`.L3:`			
Total code size	20 bytes	Total code size	18 bytes

[Output Assembly List for the V850E]

Before Modification		After Modification	
Program	Size [bytes]	Program	Size [bytes]
`ld.w -4+0x4[sp], r10`	4	`st.w r0, $_s`	4
`cmp 10, r10`	2	`ld.w -4+0x4[sp], r10`	4
`jne .L2`	2	`cmp 10, r10`	2
`mov 1, r11`	2	`jne .L2`	2
`st.w r11, $_s`	4	`mov 1, r11`	2
`jbr .L3`	2	`st.w r11, $_s`	4
`.L2:`		`.L2:`	
`st.w r0, $_s`	4		
`.L3:`			
Total code size	20 bytes	Total code size	18 bytes

2.4 Replacing an Access to an External Variable with an Access to a Temporary Variable

Since an external variable access requires 4 bytes for both loading and storing, if the value of an external variable is assigned to a temporary variable and that temporary variable is used even in cases other than assignments like those described in Section 2.2, the code size can be reduced because the memory access is changed to a register access.

A sample program is shown below.

Remark s is assumed to be an external variable.

Before Modification	After Modification

int x;
 	
if (x != 0) {
 if ((s & 0x00F00F00) != 0x00E00E00) {
 return;
 }
 s >>= 12;
 s &= 0xFF;
}
else {
 if ((s & 0x00FF0000) != 0x00EE0000) {
 return;
 }
 s >>= 24;
}
	int x;
	unsigned int tmp = s;
	
	if (x != 0) {
	 if ((tmp & 0x00F00F00) != 0x00E00E00) {
	 return;
	 }
	 tmp >>= 12;
	 tmp &= 0xFF;
	}
	else {
	 if ((tmp & 0x00FF0000) != 0x00EE0000) {
	 return;
	 }
	 tmp >>= 24;
	}
	s = tmp;

[Output Assembly List for the V850]

Before Modification		After Modification	
Program	Size [bytes]	Program	Size [bytes]
`ld.w -4+0x4[sp], r11`	4	`ld.w $_s, r10`	4
`cmp r0, r11`	2	`ld.w -4+0x4[sp], r11`	4
`je .L2`	2	`cmp r0, r11`	2
`ld.w $_s, r10`	4	`je .L2`	2
`andi 0xf00f00, r10, r12`	10	`andi 0xf00f00, r10, r12`	10
`cmp 14683648, r12`	10	`cmp 14683648, r12`	10
`jne .L1`	2	`jne .L1`	2
`shr 12, r10`	2	`shr 12, r10`	2
`andi 0xff, r10, r13`	4	`and 0xff, r10`	4
`st.w r13, $_s`	4	`jbr .L4`	2
`jbr .L1`	2	`.L2:`	
`.L2:`		`andi 0xff0000, r10, r13`	6
`ld.w $_s, r10`	4	`cmp 15597568, r13`	6
`andi 0xff0000, r10, r14`	6	`jne .L1`	2
`cmp 15597568, r14`	6	`shr 24, r10`	2
`jne .L1`	2	`.L4:`	
`sar 24, r10`	2	`st.w r10, $_s`	4
`st.w r10, $_s`	4	`.L1:`	
`.L1:`			
Total code size	70 bytes	Total code size	62 bytes

[Output Assembly List for the V850E]

Before Modification		After Modification	
Program	Size [bytes]	Program	Size [bytes]
`ld.w -4+0x4[sp], r11`	4	`ld.w $_s, r10`	4
`cmp r0, r11`	2	`ld.w -4+0x4[sp], r11`	4
`je .L2`	2	`cmp r0, r11`	2
`ld.w $_s, r10`	4	`je .L2`	2
`andi 0xf00f00, r10, r12`	8	`andi 0xf00f00, r10, r12`	8
`cmp 14683648, r12`	8	`cmp 14683648, r12`	8
`jne .L1`	2	`jne .L1`	2
`shr 12, r10`	2	`shr 12, r10`	2
`zxb r10`	2	`zxb r10`	2
`st.w r10, $_s`	4	`jbr .L4`	2
`jbr .L1`	2	`.L2:`	
`.L2:`		`andi 0xff0000, r10, r13`	6
`ld.w $_s, r10`	4	`cmp 15597568, r13`	6
`andi 0xff0000, r10, r14`	6	`jne .L1`	2
`cmp 15597568, r14`	6	`shr 24, r10`	2
`jne .L1`	2	`.L4:`	
`sar 24, r10`	2	`st.w r10, $_s`	4
`st.w r10, $_s`	4	`.L1:`	
`.L1:`			
Total code size	64 bytes	Total code size	56 bytes

2.5 Moving the Same Statement From After the Branch Clauses to Before the Branching Begins

If the same assignment statement or function call exists after each branch clause, move it in front of where the branching begins if it can be moved there.

If the evaluation result of that statement is referenced, assign it to a temporary variable and reference that temporary variable.

A sample program is shown below.

Remark s is assumed to be an external variable.

Before Modification	After Modification

`int x;`
 	
`if (x >= 0) {`
`  if (x > func(0, 1, 2)) {`
`  s++;`
`  }`
`}`
`else {`
`  if (x < -func(0, 1, 2)) {`
`  s--;`
`  }`
`}`
	`int x;`
	`int tmp;`
	
	`tmp = func(0, 1, 2);`
	`if (x >= 0) {`
	`  if (x > tmp) {`
	`  s++;`
	`  }`
	`}`
	`else {`
	`  if (x < -tmp) {`
	`  s--;`
	`  }`
	`}`

[Output Assembly List for the V850]

Before Modification		After Modification	
Program	Size [bytes]	Program	Size [bytes]
`ld.w -4+0x4[sp], r29`	4	`mov 1, r7`	2
`cmp r0, r29`	2	`mov 2, r8`	2
`jlt .L12`	2	`mov r0, r6`	2
`mov 1, r7`	2	`jarl _func, lp`	4
`mov 2, r8`	2	`ld.w -4+0x4[sp], r11`	4
`mov r0, r6`	2	`cmp r0, r11`	2
`jarl _func, lp`	4	`mov r10, r12`	2
`cmp r10, r29`	2	`jlt .L2`	2
`jle .L4`	2	`cmp r12, r11`	2
`ld.w $_s, r10`	4	`jle .L4`	2
`add 1, r10`	2	`ld.w $_s, r10`	4
`st.w r10, $_s`	4	`add 1, r10`	2
`jbr .L4`	2	`st.w r10, $_s`	4
`.L12:`		`jbr .L4`	2
`mov 1, r7`	2	`.L2:`	
`mov 2, r8`	2	`not r12, r13`	2
`mov r0, r6`	2	`add 1, r13`	2
`jarl _func, lp`	4	`cmp r11, r13`	2
`not r10, r12`	2	`jle .L4`	2
`add 1, r12`	2	`ld.w $_s, r14`	4
`cmp r29, r12`	2	`add 4294967295, r14`	2
`jle .L4`	2	`st.w r14, $_s`	4
`ld.w $_s, r13`	4	`.L4:`	
`add 4294967295, r13`	2		
`st.w r13, $_s`	4		
`.L4:`			
Total code size	62 bytes	Total code size	54 bytes

[Output Assembly List for the V850E]

Before Modification		After Modification	
Program	Size [bytes]	Program	Size [bytes]
`ld.w -4+0x4[sp], r29`	4	`mov 1, r7`	2
`cmp r0, r29`	2	`mov 2, r8`	2
`jlt .L12`	2	`mov r0, r6`	2
`mov 1, r7`	2	`jarl _func, lp`	4
`mov 2, r8`	2	`ld.w -4+0x4[sp], r11`	4
`mov r0, r6`	2	`cmp r0, r11`	2
`jarl _func, lp`	4	`mov r10, r12`	2
`cmp r10, r29`	2	`jlt .L2`	2
`jle .L4`	2	`cmp r12, r11`	2
`ld.w $_s, r10`	4	`jle .L4`	2
`add 1, r10`	2	`ld.w $_s, r10`	4
`st.w r10, $_s`	4	`add 1, r10`	2
`jbr .L4`	2	`st.w r10, $_s`	4
`.L12:`		`jbr .L4`	2
`mov 1, r7`	2	`.L2:`	
`mov 2, r8`	2	`not r12, r13`	2
`mov r0, r6`	2	`add 1, r13`	2
`jarl _func, lp`	4	`cmp r11, r13`	2
`not r10, r12`	2	`jle .L4`	2
`add 1, r12`	2	`ld.w $_s, r14`	4
`cmp r29, r12`	2	`add 4294967295, r14`	2
`jle .L4`	2	`st.w r14, $_s`	4
`ld.w $_s, r13`	4	`.L4:`	
`add 4294967295, r13`	2		
`st.w r13, $_s`	4		
`.L4:`			
Total code size	62 bytes	Total code size	54 bytes

2.6 Moving the Same Statement From Before the Control Logic Merges Together to After It Merges Together

If the same assignment statement or function call exists after each branch clause and it cannot be moved in front of where the branching begins as described in Section 2.5, then move it after the control logic merges together if it can be moved there.

A sample program is shown below.

Remark s and t are assumed to be external variables.

Before Modification	After Modification

`int tmp;`
 	
`if (tmp & 0xff00ff00) {`
`  t++;`
`  s++;`
`}`
`else {`
`  t--;`
`  s++;`
`}`
	`int tmp;`
	
	`if (tmp & 0xff00ff00) {`
	`  t++;`
	`}`
	`else {`
	`  t--;`
	`}`
	`s++;`

[Output Assembly List for the V850]

Before Modification		After Modification	
Program	Size [bytes]	Program	Size [bytes]
`ld.w  $_s, r10`	4	`ld.w  -4+0x4[sp], r10`	4
`add  1, r10`	2	`and  0xff00ff00, r10`	10
`ld.w  -4+0x4[sp], r12`	4	`je  .L2`	2
`and  0xff00ff00, r12`	10	`ld.w  $_t, r11`	4
`je  .L2`	2	`add  1, r11`	2
`ld.w  $_t, r13`	4	`st.w  r11, $_t`	4
`add  1, r13`	2	`jbr  .L3`	2
`st.w  r13, $_t`	4	`.L2:`	
`st.w  r10, $_s`	4	`ld.w  $_t, r12`	4
`jbr  .L3`	2	`add  4294967295, r12`	2
`.L2:`		`st.w  r12, $_t`	4
`ld.w  $_t, r14`	4	`.L3:`	
`add  4294967295, r14`	2	`ld.w  $_s, r13`	4
`st.w  r14, $_t`	4	`add  1, r13`	2
`st.w  r10, $_s`	4	`st.w  r13, $_s`	4
`.L3:`			
Total code size	52 bytes	Total code size	48 bytes

[Output Assembly List for the V850E]

Before Modification		After Modification	
Program	Size [bytes]	Program	Size [bytes]
`ld.w $_s, r10`	4	`ld.w -4+0x4[sp], r10`	4
`add 1, r10`	2	`and 0xff00ff00, r10`	8
`ld.w -4+0x4[sp], r12`	4	`je .L2`	2
`and 0xff00ff00, r12`	8	`ld.w $_t, r11`	4
`je .L2`	2	`add 1, r11`	2
`ld.w $_t, r13`	4	`st.w r11, $_t`	4
`add 1, r13`	2	`jbr .L3`	2
`st.w r13, $_t`	4	`.L2:`	
`st.w r10, $_s`	4	`ld.w $_t, r12`	4
`jbr .L3`	2	`add 4294967295, r12`	2
`.L2:`		`st.w r12, $_t`	4
`ld.w $_t, r14`	4	`.L3:`	
`add 4294967295, r14`	2	`ld.w $_s, r13`	4
`st.w r14, $_t`	4	`add 1, r13`	2
`st.w r10, $_s`	4	`st.w r13, $_s`	4
`.L3:`			
Total code size	50 bytes	Total code size	46 bytes

2.7 Using a Temporary Variable to Consolidate the Calls to the Same Function with Different Arguments That Appear After Each Branch Clause at the Location After the Control Logic Merges Together

If a different argument is used when calling the same function after each branch clause, move the function call after the location where the control logic merges together again if it can be moved there. At this time, at each call location, assign the argument that differs to a temporary variable and use that temporary variable as the argument in the function call.

A sample program is shown below.

Remark s is assumed to be an external variable.

Before Modification	After Modification

`if (s) {`
`  func(0, 1, 2);`
`}`
`else {`
`  func(0, 1, 3);`
`}`
	`int tmp;`
	
	`if (s) {`
	`  tmp = 2;`
	`}`
	`else {`
	`  tmp = 3;`
	`}`
	`func(0, 1, tmp);`

[Output Assembly List for the V850]

Before Modification		After Modification	
Program	Size [bytes]	Program	Size [bytes]
`  ld.w  $_s, r10`	4	`  ld.w  $_s, r10`	4
`  cmp  r0, r10`	2	`  cmp  r0, r10`	2
`  je  .L9`	2	`  je  .L2`	2
`  mov  1, r7`	2	`  mov  2, r11`	2
`  mov  2, r8`	2	`  jbr  .L7`	2
`  mov  r0, r6`	2	`.L2:`	
`  jarl  _func, lp`	4	`  mov  3, r11`	2
`  jbr  .L3`	2	`.L7:`	
`.L9:`		`  mov  1, r7`	2
`  mov  1, r7`	2	`  mov  r0, r6`	2
`  mov  3, r8`	2	`  mov  r11, r8`	2
`  mov  r0, r6`	2	`  jarl  _func, lp`	4
`  jarl  _func, lp`	4		
`.L3:`			
Total code size	30 bytes	Total code size	24 bytes

[Output Assembly List for the V850E]

Before Modification		After Modification	
Program	Size [bytes]	Program	Size [bytes]
`ld.w $_s, r10`	4	`ld.w $_s, r10`	4
`cmp r0, r10`	2	`cmp r0, r10`	2
`je .L9`	2	`je .L2`	2
`mov 1, r7`	2	`mov 2, r11`	2
`mov 2, r8`	2	`jbr .L7`	2
`mov r0, r6`	2	`.L2:`	
`jarl _func, lp`	4	`mov 3, r11`	2
`jbr .L3`	2	`.L7:`	
`.L9:`		`mov 1, r7`	2
`mov 1, r7`	2	`mov r0, r6`	2
`mov 3, r8`	2	`mov r11, r8`	2
`mov r0, r6`	2	`jarl _func, lp`	4
`jarl _func, lp`	4		
`.L3:`			
Total code size	30 bytes	Total code size	24 bytes

2.8 Replacing a Complex if Statement with One That Is Logically Equivalent

When the same processing is executed according to multiple cases in a combination of if~else statements, if the "multiple cases" can be consolidated by using a separate condition, consolidate them and remove the excess portions.

A sample program is shown below.

Remark When the conditions that the initial value of x is 0 and the values of s and t are either 0 or 1 are consolidated, they can be transformed as shown in this example. s, t, u, and v are assumed to be external variables.

Before Modification

```
int x;

if (!s) {
  if (t) {
    x = 1;
  }
}
else {
  if (!t) {
    x = 1;
  }
}
if (x) {
  if ((++u) >= v) {
    u = 0;
  }
  else {
    x = 0;
  }
}
```

After Modification

```
int x;

if (! (s ^ t)) {
  if ((++u) >= v) {
    u = 0;
    x = 1;
  }
}
```

[Output Assembly List for the V850]

Before Modification		After Modification	
Program	Size [bytes]	Program	Size [bytes]
`ld.w $_s, r11`	4	`ld.w $_s, r11`	4
`cmp r0, r11`	2	`ld.w $_t, r12`	4
`jne .L2`	2	`xor r12, r11`	2
`ld.w $_t, r12`	4	`jne .L2`	2
`cmp r0, r12`	2	`ld.w $_u, r10`	4
`je .L4`	2	`add 1, r10`	2
`mov 1, r13`	2	`st.w r10, $_u`	4
`st.w r13, -4+0x4[sp]`	4	`ld.w $_v, r14`	4
`jbr .L4`	2	`cmp r14, r10`	2
`.L2:`		`jlt .L2`	2
`ld.w $_t, r14`	4	`st.w r0, $_u`	4
`cmp r0, r14`	2	`.L2:`	
`jne .L4`	2		
`mov 1, r15`	2		
`st.w r15, -4+0x4[sp]`	4		
`.L4:`			
`ld.w -4+0x4[sp], r16`	4		
`cmp r0, r16`	2		
`je .L6`	2		
`ld.w $_u, r10`	4		
`add 1, r10`	2		
`ld.w $_v, r18`	4		
`cmp r18, r10`	2		
`jlt .L7`	2		
`st.w r0, $_u`	4		
`jbr .L6`	2		
`.L7:`			
`st.w r10, $_u`	4		
`.L6:`			
Total code size	70 bytes	Total code size	34 bytes

[Output Assembly List for the V850E]

Before Modification		After Modification	
Program	Size [bytes]	Program	Size [bytes]
`ld.w $_s, r11`	4	`ld.w $_s, r11`	4
`cmp r0, r11`	2	`ld.w $_t, r12`	4
`jne .L2`	2	`xor r12, r11`	2
`ld.w $_t, r12`	4	`jne .L2`	2
`cmp r0, r12`	2	`ld.w $_u, r10`	4
`je .L4`	2	`add 1, r10`	2
`mov 1, r13`	2	`st.w r10, $_u`	4
`st.w r13, -4+0x4[sp]`	4	`ld.w $_v, r14`	4
`jbr .L4`	2	`cmp r14, r10`	2
`.L2:`		`jlt .L2`	2
`ld.w $_t, r14`	4	`st.w r0, $_u`	4
`cmp r0, r14`	2	`.L2:`	
`jne .L4`	2		
`mov 1, r15`	2		
`st.w r15, -4+0x4[sp]`	4		
`.L4:`			
`ld.w -4+0x4[sp], r16`	4		
`cmp r0, r16`	2		
`je .L6`	2		
`ld.w $_u, r10`	4		
`add 1, r10`	2		
`ld.w $_v, r18`	4		
`cmp r18, r10`	2		
`jlt .L7`	2		
`st.w r0, $_u`	4		
`jbr .L6`	2		
`.L7:`			
`st.w r10, $_u`	4		
`.L6:`			
Total code size	70 bytes	Total code size	34 bytes

2.9 Transforming a for or while Loop Into a goto Loop

When a loop begins with a conditional decision expression such as a for or while loop, the CA850 generates the conditional decision expression twice as shown in the following image diagram.

This kind of loop transformation is performed by the front end (syntax analyzer), which is the initial phase of the compiler. The initial conditional decision is often eliminated by the subsequent optimization, but this kind of transformation is performed because it is advantageous from the viewpoint of increasing execution speed. However, when this description is not eliminated, the code size is increased.

[Image diagram]

Syntax	Image

for (statement-1; expression-2; statement-3) {
loop-body
}
	statement-1;
	if (expression-2) {
	do {
	loop-body
	statement-3;
	} while (expression-2) ;
	}

while (expression-1) {
loop-body
}
	if (expression-1) {
	do {
	loop-body
	} while (expression-1) ;
	}

Therefore, if the first conditional decision expression is not eliminated by the optimization, transforming to a loop formed by goto statements as follows enables the number of conditional decision expressions to be reduced to one.

Write the coding as shown in the following image diagram.

[Image diagram]

for loop	statement-1; loop_bgn: if (! expression-2) goto loop_end; loop-body statement-3; goto loop_bgn; loop_end:
while loop	loop_bgn: if (! expression-1) goto loop_end; loop-body goto loop_bgn; loop_end:

A sample program is shown below.

Remark s and array[ ] are assumed to be external variables.

Before Modification	After Modification

`int i;`
 	
`for (i = 0; i < s; ++i) {`
`  array[i] = array[i+1];`
`}`
	`int i;`
	
	`i = 0;`
	`bgn_loop:`
	`if (i >= s) goto end_loop;`
	`  array[i] = array[i+1];`
	`  ++i;`
	`  goto bgn_loop;`
	`end_loop:`
	`  ;`

[Output Assembly List for the V850]

Before Modification		After Modification	
Program	Size [bytes]	Program	Size [bytes]
`ld.w $_s, r12`	4	`mov r0, r10`	2
`cmp r0, r12`	2	`.L16:`	
`jle .L18`	2	`ld.w $_s, r11`	4
`movea $_array, gp, r10`	4	`cmp r11, r10`	2
`mov r0, r11`	2	`jge .L20`	2
`.L16:`		`mov r10, r12`	2
`mov r11, r13`	2	`shl 2, r12`	2
`shl 2, r13`	2	`movea $_array, gp, r13`	4
`add r10, r13`	2	`add r12, r13`	2
`ld.w 4[r13], r14`	4	`ld.w 4[r13], r14`	4
`st.w r14, [r13]`	4	`st.w r14, [r13]`	4
`add 1, r11`	2	`add 1, r10`	2
`ld.w $_s, r15`	4	`jbr .L16`	2
`cmp r15, r11`	2	`.L20:`	
`jlt .L16`	2		
`.L18:`			
Total code size	38 bytes	Total code size	32 bytes

[Output Assembly List for the V850E]

Before Modification		After Modification	
Program	Size [bytes]	Program	Size [bytes]
`ld.w $_s, r12`	4	`mov r0, r10`	2
`cmp r0, r12`	2	`.L16:`	
`jle .L18`	2	`ld.w $_s, r11`	4
`movea $_array, gp, r10`	4	`cmp r11, r10`	2
`mov r0, r11`	2	`jge .L20`	2
`.L16:`		`mov r10, r12`	2
`mov r11, r13`	2	`shl 2, r12`	2
`shl 2, r13`	2	`movea $_array, gp, r13`	4
`add r10, r13`	2	`add r12, r13`	2
`ld.w 4[r13], r14`	4	`ld.w 4[r13], r14`	4
`st.w r14, [r13]`	4	`st.w r14, [r13]`	4
`add 1, r11`	2	`add 1, r10`	2
`ld.w $_s, r15`	4	`jbr .L16`	2
`cmp r15, r11`	2	`.L20:`	
`jlt .L16`	2		
`.L18:`			
Total code size	38 bytes	Total code size	32 bytes

2.10 Unrolling a Loop

When the execution count is small and the loop body is small, the coding size of the unrolled loop may be smaller. In this case, the execution speed is also increased.

A sample program is shown below.

Remark array[ ] is assumed to be an external variable.

Before Modification	After Modification

`int i;`
 	
`for (i = 0; i < 4; i++) {`
`  array[i] = 0;`
`}`
	`int *p;`
	
	`p = array;`
	`*p = 0;`
	`*(p+1) = 0;`
	`*(p+2) = 0;`
	`*(p+3) = 0;`

[Output Assembly List for the V850]

Before Modification		After Modification	
Program	Size [bytes]	Program	Size [bytes]
`movea $_array, gp, r10`	4	`st.w r0, $_array`	4
`mov r0, r11`	2	`st.w r0, $_array+4`	4
`.L15:`		`st.w r0, $_array+8`	4
`mov r11, r12`	2	`st.w r0, $_array+12`	4
`shl 2, r12`	2		
`add r10, r12`	2		
`st.w r0, [r12]`	4		
`add 1, r11`	2		
`cmp 4, r11`	2		
`jlt .L15`	2		
Total code size	22 bytes	Total code size	16 bytes

[Output Assembly List for the V850E]

Before Modification		After Modification	
Program	Size [bytes]	Program	Size [bytes]
`movea $_array, gp, r10`	4	`st.w r0, $_array`	4
`mov r0, r11`	2	`st.w r0, $_array+4`	4
`.L15:`		`st.w r0, $_array+8`	4
`mov r11, r12`	2	`st.w r0, $_array+12`	4
`shl 2, r12`	2		
`add r10, r12`	2		
`st.w r0, [r12]`	4		
`add 1, r11`	2		
`cmp 4, r11`	2		
`jlt .L15`	2		
Total code size	22 bytes	Total code size	16 bytes

2.11 Shortening the Lifespan of a Variable

If there is an interval between the location where a value is assigned to a stack variable and the location where that value is actually referenced, a register is occupied during that interval and the opportunities for other variables being allocated to that register are reduced. Although the assignment of values is often moved to the end of the interval by the compiler optimization, this optimization may not be performed if the interval contains a function call.

In this case, changing the coding so that the assignment is performed immediately before the value is actually referenced will increase the register allocation opportunities of other variables, reduce memory accesses, and decrease the code size.

A sample program is shown below.

Remark s1, s2, s3, and array[ ] are assumed to be external variables.

Before Modification

```
int i = 0, j = 0, k = 0, m = 0;

 :

 while ((k & 0xFF) != 0xFF) {
   k = s1;
   if (k & 0xFF00) {
     if (m != 1) {
       s2 += 2;
       m = 1;
       array[15+i+j] = 0xFF;
       j++;
     }
   }
 }
```

After Modification

```
int i, j, k, m;

 :

 i = 0;
 j = 0;
 k = 0;
 m = 0;
 while ((k & 0xFF) != 0xFF) {
   k = s1;
   if (k & 0xFF00) {
     if (m != 1) {
       s2 += 2;
       m = 1;
       array[15+i+j] = 0xFF;
       j++;
     }
   }
 }
```

[Output Assembly List for the V850]

Before Modification		After Modification	
Program	Size [bytes]	Program	Size [bytes]
`ld.w $_s1, r12`	4	`ld.w $_s1, r14`	4
`and 0xff00, r12`	4	`andi 0xff00, r14, r11`	4
`mov r0, r29`	2	`and 0xff, r14`	4
`mov r0, r28`	2	`mov r0, r13`	2
`.L21:`		`mov r0, r12`	2
`ld.w $_s1, r11`	4	`.L22:`	
`cmp r0, r12`	2	`cmp r0, r11`	2
`je .L27`	2	`je .L28`	2
`cmp 1, r29`	2	`cmp 1, r12`	2
`je .L27`	2	`je .L28`	2
`ld.w $_s2, r13`	4	`ld.w $_s2, r15`	4
`add 2, r13`	2	`add 2, r15`	2
`st.w r13, $_s2`	4	`st.w r15, $_s2`	4
`mov 1, r29`	2	`mov 1, r12`	2
`mov r28, r14`	2	`mov r13, r16`	2
`shl 2, r14`	2	`shl 2, r16`	2
`movea $_array, gp, r15`	4	`movea $_array, gp, r17`	4
`add r14, r15`	2	`add r16, r17`	2
`mov 255, r16`	4	`mov 255, r18`	4
`st.w r16, 60[r15]`	4	`st.w r18, 60[r17]`	4
`add 1, r28`	2	`add 1, r13`	2
`.L27:`		`.L28:`	
`andi 0xff, r11, r17`	4	`addi -255, r14, r0`	4
`addi -255, r17, r0`	4	`jne .L22`	2
`jne .L21`	2		
Total code size	66 bytes	Total code size	62 bytes

[Output Assembly List for the V850E]

Before Modification				After Modification			
Program			Size [bytes]	Program			Size [bytes]
	ld.w	$_s1, r12	4		ld.w	$_s1, r14	4
	and	0xff00, r12	4		andi	0xff00, r14, r11	4
	mov	r0, r29	2		zxb	r14	2
	mov	r0, r28	2		mov	r0, r13	2
.L21:					mov	r0, r12	2
	ld.w	$_s1, r11	4	.L22:			
	cmp	r0, r12	2		cmp	r0, r11	2
	je	.L27	2		je	.L28	2
	cmp	1, r29	2		cmp	1, r12	2
	je	.L27	2		je	.L28	2
	ld.w	$_s2, r13	4		ld.w	$_s2, r15	4
	add	2, r13	2		add	2, r15	2
	st.w	r13, $_s2	4		st.w	r15, $_s2	4
	mov	1, r29	2		mov	1, r12	2
	mov	r28, r14	2		mov	r13, r16	2
	shl	2, r14	2		shl	2, r16	2
	movea	$_array, gp, r15	4		movea	$_array, gp, r17	4
	add	r14, r15	2		add	r16, r17	2
	mov	255, r16	4		mov	255, r18	4
	st.w	r16, 60[r15]	4		st.w	r18, 60[r17]	4
	add	1, r28	2		add	1, r13	2
.L27:				.L28:			
	zxb	r11	2		addi	-255, r14, r0	4
	addi	-255, r11, r0	4		jne	.L22	2
	jne	.L21	2				
Total code size			64 bytes	Total code size			60 bytes

2.12 Eliminating an Induction Variable

A variable that controls a loop is called an induction variable, and optimization that eliminates an induction variable by changing the loop control so that another variable is used is referred to as "eliminating an induction variable."

Although this optimization is also included in the CA850, since the conditions for which it is applied are limited, not all cases can be optimized.

Therefore, perform this optimization "manually" by altering the program as follows.

A sample program is shown below.

Remark x and *table are assumed to be external variables.

Before Modification	After Modification

`int i;`
 	
`for (i = 0; *(table + i) != NULL; ++i) {`
`  if ((*(table + i) & 0xFF) == x) {`
`  return(*(table + i) & 0xFF00);`
`  }`
`}`
	`const unsigned short *p;`
	
	`for (p = table; *p != NULL; ++p) {`
	`  if ((*p & 0xFF) == x) {`
	`  return(*p & 0xFF00);`
	`  }`
	`}`

[Output Assembly List for the V850]

Before Modification		After Modification	
Program	Size [bytes]	Program	Size [bytes]
`ld.w $_table, r12`	4	`ld.w $_table, r12`	4
`ld.h [r12], r13`	4	`ld.h [r12], r13`	4
`and 0xffff, r13`	4	`and 0xffff, r13`	4
`je .L1`	2	`je .L1`	2
`ld.h [r12], r10`	4	`ld.h [r12], r10`	4
`mov r0, r11`	2	`mov r12, r11`	2
`.L2:`		`.L2:`	
`andi 0xff, r10, r14`	4	`andi 0xff, r10, r14`	4
`ld.h $_x, r15`	4	`ld.h $_x, r15`	4
`and 0xffff, r15`	4	`and 0xffff, r15`	4
`cmp r14, r15`	2	`cmp r14, r15`	2
`jne .L5`	2	`jne .L5`	2
`andi 0xff00, r10, r16`	4	`andi 0xff00, r10, r16`	4
`st.w r16, -4+0x4[sp]`	4	`st.w r16, -4+0x4[sp]`	4
`jbr .L1`	2	`jbr .L1`	2
`.L5:`		`.L5:`	
`add 1, r11`	2	`add 2, r11`	2
`mov r11, r10`	2	`ld.h [r11], r10`	4
`shl 1, r10`	2	`and 0xffff, r10`	4
`add r12, r10`	2	`jne .L2`	2
`ld.h [r10], r10`	4	`.L1:`	
`and 0xffff, r10`	4	`ld.w -4+0x4[sp], r10`	4
`jne .L2`	2		
`.L1:`			
`ld.w -4+0x4[sp], r10`	4		
Total code size	68 bytes	Total code size	62 bytes

[Output Assembly List for the V850E]

Before Modification		After Modification	
Program	Size [bytes]	Program	Size [bytes]
`ld.w $_table, r12`	4	`ld.w $_table, r12`	4
`ld.hu [r12], r13`	4	`ld.hu [r12], r13`	4
`cmp r0, r13`	2	`cmp r0, r13`	2
`je .L1`	2	`je .L1`	2
`ld.hu [r12], r10`	4	`ld.hu [r12], r10`	4
`mov r0, r11`	2	`mov r12, r11`	2
`.L2:`		`.L2:`	
`mov r10, r14`	2	`mov r10, r14`	2
`zxb r14`	2	`zxb r14`	2
`ld.hu $_x, r15`	4	`ld.hu $_x, r15`	4
`cmp r14, r15`	2	`cmp r14, r15`	2
`jne .L5`	2	`jne .L5`	2
`andi 0xff00, r10, r16`	4	`andi 0xff00, r10, r16`	4
`st.w r16, -4+0x4[sp]`	4	`st.w r16, -4+0x4[sp]`	4
`jbr .L1`	2	`jbr .L1`	2
`.L5:`		`.L5:`	
`add 1, r11`	2	`add 2, r11`	2
`mov r11, r10`	2	`ld.hu [r11], r10`	4
`shl 1, r10`	2	`cmp r0, r10`	2
`add r12, r10`	2	`jne .L2`	2
`ld.hu [r10], r10`	4	`.L1:`	
`cmp r0, r10`	2	`ld.w -4+0x4[sp], r10`	4
`jne .L2`	2		
`.L1:`			
`ld.w -4+0x4[sp], r10`	4		
Total code size	60 bytes	Total code size	54 bytes

2.13 Setting (unsigned) int Type for (unsigned) short or char Type Variables

According to the (unsigned) ANSI-C specifications, since (unsigned) short or (unsigned) char type variables are extended to int type or unsigned int type when calculations are performed, type conversion instructions often are generated for programs that use these kinds of variables (in particular, when these kinds of variables are allocated to registers).

Since this type conversion will be unnecessary if (unsigned) int type is set, the code size is reduced.

In particular, (unsigned) int type should be used as much as possible for stack variables, which are allocated to registers relatively often.

A sample program is shown below.

Remark array[ ] and *p are assumed to be external variables.

Before Modification	After Modification

`unsigned char i;`
 	
`for(i = 0; i < 4; i++) {`
`  array[2 + i] = *(p + i);`
`}`
	`int i;`
	
	`for(i = 0; i < 4; i++) {`
	`  array[2 + i] = *(p + i);`
	`}`

[Output Assembly List for the V850]

Before Modification		After Modification	
Program	Size [bytes]	Program	Size [bytes]
`movea $_array, gp, r10`	4	`movea $_array, gp, r10`	4
`mov r0, r12`	2	`mov r0, r12`	2
`.L2:`		`.L2:`	
`mov r12, r11`	2	`mov r12, r11`	2
`shl 2, r11`	2	`shl 2, r11`	2
`mov r10, r13`	2	`mov r10, r13`	2
`add r11, r13`	2	`add r11, r13`	2
`ld.w $_p, r15`	4	`ld.w $_p, r15`	4
`add r11, r15`	2	`add r11, r15`	2
`ld.w [r15], r15`	4	`ld.w [r15], r15`	4
`st.w r15, 8[r13]`	4	`st.w r15, 8[r13]`	4
`add 1, r12`	2	`add 1, r12`	2
`and 0xff, r12`	4	`cmp 4, r12`	2
`cmp 4, r12`	2	`jlt .L2`	2
`jlt .L2`	2		
Total code size	38 bytes	Total code size	34 bytes

[Output Assembly List for the V850E]

Before Modification		After Modification	
Program	Size [bytes]	Program	Size [bytes]
`movea $_array, gp, r10`	4	`movea $_array, gp, r10`	4
`mov r0, r12`	2	`mov r0, r12`	2
`.L2:`		`.L2:`	
`mov r12, r11`	2	`mov r12, r11`	2
`shl 2, r11`	2	`shl 2, r11`	2
`mov r10, r13`	2	`mov r10, r13`	2
`add r11, r13`	2	`add r11, r13`	2
`ld.w $_p, r15`	4	`ld.w $_p, r15`	4
`add r11, r15`	2	`add r11, r15`	2
`ld.w [r15], r15`	4	`ld.w [r15], r15`	4
`st.w r15, 8[r13]`	4	`st.w r15, 8[r13]`	4
`add 1, r12`	2	`add 1, r12`	2
`zxb r12`	2	`cmp 4, r12`	2
`cmp 4, r12`	2	`jlt .L2`	2
`jlt .L2`	2		
Total code size	36 bytes	Total code size	34 bytes

2.14 Consolidating in a Single Statement When an Assigned Value Is Referenced in the Statement Following the Assignment Statement

When an assigned value is referenced in the statement following an assignment statement, consolidating the two statements in a single statement by replacing the referencing location with the assignment statement may eliminate an extra register transfer and reduce the code size. However, in many cases, the code size will not change because the extra register transfer has been eliminated by the compiler optimization.

A sample program is shown below.

Remark s and t are assumed to be external variables.

Before Modification	After Modification

`--s;`
`if (s == 0) {`
`  t++;`
`}`
	`if (--s == 0) {`
	`  t++;`
	`}`

In this example, the same code size will be produced by compiler optimization.

2.15 Eliminating the if~else Statement When the Branch Destinations of the if~else Statement Are return Statements That Return the Result of the Branch Condition

When each of the branch destinations of an if~else statement contains only a return statement and the corresponding return values are the branch condition results themselves, eliminate the if~else statement and make the program return the value of the branch condition expression.

A sample program is shown below.

Remark s1 and s2 are assumed to be external variables.

Before Modification	After Modification

`if (s1 == s2) {`
`  return(1);`
`}`
`return(0);`
	`return (s1 == s2);`

[Output Assembly List for the V850]

Before Modification		After Modification	
Program	Size [bytes]	Program	Size [bytes]
`ld.w $_s1, r11`	4	`ld.w $_s1, r11`	4
`ld.w $_s2, r12`	4	`ld.w $_s2, r12`	4
`cmp r12, r11`	2	`cmp r12, r11`	2
`jne .L2`	2	`setfe r10`	4
`mov 1, r10`	2		
`jbr .L1`	2		
`.L2:`			
`mov r0, r10`	2		
`.L1:`			
Total code size	18 bytes	Total code size	14 bytes

[Output Assembly List for the V850E]

Before Modification		After Modification	
Program	Size [bytes]	Program	Size [bytes]
`ld.w $_s1, r11`	4	`ld.w $_s1, r11`	4
`ld.w $_s2, r12`	4	`ld.w $_s2, r12`	4
`cmp r12, r11`	2	`cmp r12, r11`	2
`jne .L2`	2	`setfe r10`	4
`mov 1, r10`	2		
`jbr .L1`	2		
`.L2:`			
`mov r0, r10`	2		
`.L1:`			
Total code size	18 bytes	Total code size	14 bytes

2.16 Changing the Condition and Setting the Operand to 15 or –16 When One of the Operands of a Comparison Operation Is the Constant 16 or –17

When one of the operands of the comparison instruction cmp is not immediate data in the range –16 to 15, the assembler performs instruction expansion and the comparison instruction will become multiple instructions.

Therefore, if the operand value is held to the range from –16 to 15 by changing the condition, the expansion will be suppressed and the coding size will be reduced.

Specifically, change the value used for the comparison in the conditional expression of a statement such as a for or if statement.

A sample program is shown below.

Remark s is assumed to be an external variable.

Before Modification	After Modification

`int i;`
 	
`for (i = 0; i < 16; i++) {`
`  s++;`
`}`
	`int i;`
	
	`for (i = 0; i <= 15; i++) {`
	`  s++;`
	`}`

In this example, the same code size will be produced by compiler optimization.

2.17 Initializing a Variable

When an auto variable is used in a function without being initialized, the code size is increased because that variable remains in memory without being allocated to a register.

In the example shown below, when the variable a does not correspond to either case of the switch statement, it is referenced by the return statement without being initialized. Actually, even if it always corresponds to one of the cases, it may be treated as if it is not initialized because its value is not known when the program is analyzed during register allocation. In this case, no register will be allocated during CA850 register allocation.

Therefore, if the variable is allocated to a register by adding an initialization, the code size can be reduced.

A sample program is shown below.

Remark a is assumed to be an auto variable.

Before Modification	After Modification

`int a;`
 	
`switch(x) {`
`  case 0:`
`  a = 0;`
`  break;`
`  case 1:`
`  a = 1;`
`}`
`return a;`
	`int a = 0;`
	
	`switch(x) {`
	`  case 0:`
	`  a = 0;`
	`  break;`
	`  case 1:`
	`  a = 1;`
	`}`
	`return a;`

[Output Assembly List for the V850]

Before Modification		After Modification	
Program	Size [bytes]	Program	Size [bytes]
`cmp r0, r6`	2	`cmp r0, r6`	2
`je .L4`	2	`mov r0, r11`	2
`cmp 1, r6`	2	`je .L3`	2
`je .L5`	2	`cmp 1, r6`	2
`jbr .L3`	2	`jne .L3`	2
`.L4:`		`mov 1, r11`	2
`st.w r0, -4+0x4[sp]`	4	`.L3:`	
`jbr .L3`	2	`mov r11, r10`	2
`.L5:`			
`mov 1, r11`	2		
`st.w r11, -4+0x4[sp]`	4		
`.L3:`			
`ld.w -4+0x4[sp], r10`	4		
Total code size	26 bytes	Total code size	14 bytes

[Output Assembly List for the V850E]

Before Modification		After Modification	
Program	Size [bytes]	Program	Size [bytes]
`cmp r0, r6`	2	`cmp r0, r6`	2
`je .L4`	2	`mov r0, r11`	2
`cmp 1, r6`	2	`je .L3`	2
`je .L5`	2	`cmp 1, r6`	2
`jbr .L3`	2	`jne .L3`	2
`.L4:`		`mov 1, r11`	2
`st.w r0, -4+0x4[sp]`	4	`.L3:`	
`jbr .L3`	2	`mov r11, r10`	2
`.L5:`			
`mov 1, r11`	2		
`st.w r11, -4+0x4[sp]`	4		
`.L3:`			
`ld.w -4+0x4[sp], r10`	4		
Total code size	26 bytes	Total code size	14 bytes

2.18 Declaring void for a Function Having No Return Value

Declare void for a function that has no return value.
The extra register transfer instruction will be eliminated.

A sample program is shown below.

Before Modification	After Modification

`func(int a)`
`{`
`  switch(a) {`
`    case 0:`
`      s = 0;`
`      break;`
`    case 1:`
`      s = 1;`
`  }`
`}`
	`void func(int a)`
	`{`
	`  switch(a) {`
	`    case 0:`
	`      s = 0;`
	`      break;`
	`    case 1:`
	`      s = 1;`
	`  }`
	`}`

[Output Assembly List for the V850]

Before Modification		After Modification	
Program	Size [bytes]	Program	Size [bytes]
`cmp r0, r6`	2	`cmp r0, r6`	2
`je .L4`	2	`je .L4`	2
`cmp 1, r6`	2	`cmp 1, r6`	2
`je .L5`	2	`je .L5`	2
`jbr .L3`	2	`jbr .L3`	2
`.L4:`		`.L4:`	
`st.w r0, $_s`	4	`st.w r0, $_s`	4
`jbr .L3`	2	`jbr .L3`	2
`.L5:`		`.L5:`	
`mov 1, r11`	2	`mov 1, r10`	2
`st.w r11, $_s`	4	`st.w r10, $_s`	4
`.L3:`		`.L3:`	
`mov r0, r10`	2	`jmp [lp]`	2
`jmp [lp]`	2		
Total code size	26 bytes	Total code size	24 bytes

[Output Assembly List for the V850E]

Before Modification		After Modification	
Program	Size [bytes]	Program	Size [bytes]
`cmp r0, r6`	2	`cmp r0, r6`	2
`je .L4`	2	`je .L4`	2
`cmp 1, r6`	2	`cmp 1, r6`	2
`je .L5`	2	`je .L5`	2
`jbr .L3`	2	`jbr .L3`	2
`.L4:`		`.L4:`	
`st.w r0, $_s`	4	`st.w r0, $_s`	4
`jbr .L3`	2	`jbr .L3`	2
`.L5:`		`.L5:`	
`mov 1, r11`	2	`mov 1, r10`	2
`st.w r11, $_s`	4	`st.w r10, $_s`	4
`.L3:`		`.L3:`	
`mov r0, r10`	2	`jmp [lp]`	2
`jmp [lp]`	2		
Total code size	26 bytes	Total code size	24 bytes

2.19 Consolidating Common case Processing in a switch Statement

If the identical case processing descriptions of a switch statement are consolidated, the code size may be reduced.

A sample program is shown below.

Remark x is assumed to be an external variable.

Before Modification	After Modification

switch(x) {
  case 0:
  dummy1();
  break;
  case 1:
  dummy1();
  break;
  case 2:
  dummy1();
  break;
  case 3:
  dummy2();
  break;
  case 4:
  dummy2();
  break;
  default:
  break;
}
	switch(x) {
	  case 0:
	  case 1:
	  case 2:
	  dummy1();
	  break;
	  case 3:
	  case 4:
	  dummy2();
	  break;
	  default:
	  break;
	}

[Output Assembly List for the V850]

Before Modification		After Modification	
Program	Size [bytes]	Program	Size [bytes]
`ld.w -4+.A2[sp], r10`	4	`ld.w -4+.A2[sp], r10`	4
`cmp 4, r10`	2	`cmp 4, r10`	2
`jh .L3`	2	`jh .L3`	2
`shl 1, r10`	2	`shl 1, r10`	2
`add tp, r10`	2	`add tp, r10`	2
`ld.h .L10[r10], r11`	8	`ld.h .L10[r10], r11`	8
`add .L10, r11`	10	`add .L10, r11`	10
`add tp, r11`	2	`add tp, r11`	2
`jmp [r11]`	2	`jmp [r11]`	2
`.L10:`		`.L10:`	
`.hword .L13-.L10`	2	`.hword .L13-.L10`	2
`.hword .L16-.L10`	2	`.hword .L13-.L10`	2
`.hword .L19-.L10`	2	`.hword .L13-.L10`	2
`.hword .L22-.L10`	2	`.hword .L16-.L10`	2
`.hword .L25-.L10`	2	`.hword .L16-.L10`	2
`.L13:`		`.L13:`	
`jarl _dummy1, lp`	4	`jarl _dummy1, lp`	4
`jbr .L3`	2	`jbr .L3`	2
`.L16:`		`.L16:`	
`jarl _dummy1, lp`	4	`jarl _dummy2, lp`	4
`jbr .L3`	2	`.L3:`	
`.L19:`			
`jarl _dummy1, lp`	4		
`jbr .L3`	2		
`.L22:`			
`jarl _dummy2, lp`	4		
`jbr .L3`	2		
`.L25:`			
`jarl _dummy2, lp`	4		
`.L3:`			
Total code size	72 bytes	Total code size	54 bytes

[Output Assembly List for the V850E]

Before Modification		After Modification	
Program	Size [bytes]	Program	Size [bytes]
`ld.w -4+.A2[sp], r10`	4	`ld.w -4+.A2[sp], r10`	4
`cmp 4, r10`	2	`cmp 4, r10`	2
`jh .L3`	2	`jh .L3`	2
`switch r10`	2	`switch r10`	2
`.L10:`		`.L10:`	
`.shword .L13-.L10`	2	`.shword .L13-.L10`	2
`.shword .L16-.L10`	2	`.shword .L13-.L10`	2
`.shword .L19-.L10`	2	`.shword .L13-.L10`	2
`.shword .L22-.L10`	2	`.shword .L16-.L10`	2
`.shword .L25-.L10`	2	`.shword .L16-.L10`	2
`.L13:`		`.L13:`	
`jarl _dummy1, lp`	4	`jarl _dummy1, lp`	4
`jbr .L3`	2	`jbr .L3`	2
`.L16:`		`.L16:`	
`jarl _dummy1, lp`	4	`jarl _dummy2, lp`	4
`jbr .L3`	2	`.L3:`	
`.L19:`			
`jarl _dummy1, lp`	4		
`jbr .L3`	2		
`.L22:`			
`jarl _dummy2, lp`	4		
`jbr .L3`	2		
`.L25:`			
`jarl _dummy2, lp`	4		
`.L3:`			
Total code size	48 bytes	Total code size	30 bytes

2.20 Consolidating return Statements Having Identical Values

If descriptions of return statements that return the same value are consolidated, the code size may be reduced.

A sample program is shown below.

Remark s, t, and u are assumed to be external variables.

Before Modification	After Modification

`if(s == 1)return 0xff;`
`if(t == 1)return 0xff;`
`if(u == 1)return 0xff;`
`return 0x0;`
	`if((s == 1) || (t == 1) || (u == 1))return 0xff;`
	`return 0x0;`

[Output Assembly List for the V850]

Before Modification		After Modification	
Program	Size [bytes]	Program	Size [bytes]
`ld.w $_s, r11`	4	`ld.w $_s, r11`	4
`cmp 1, r11`	2	`cmp 1, r11`	2
`jne .L2`	2	`je .L3`	2
`mov 255, r10`	4	`ld.w $_t, r12`	4
`jbr .L1`	2	`cmp 1, r12`	2
`.L2:`		`je .L3`	2
`ld.w $_t, r12`	4	`ld.w $_u, r13`	4
`cmp 1, r12`	2	`cmp 1, r13`	2
`jne .L3`	2	`je .L3`	2
`mov 255, r10`	4	`mov r0, r10`	2
`jbr .L1`	2	`jbr .L1`	2
`.L3:`		`.L3:`	
`ld.w $_u, r13`	4	`mov 255, r10`	4
`cmp 1, r13`	2	`.L1:`	
`jne .L4`	2		
`mov 255, r10`	4		
`jbr .L1`	2		
`.L4:`			
`mov r0, r10`	2		
`.L1:`			
Total code size	44 bytes	Total code size	32 bytes

[Output Assembly List for the V850E]

Before Modification		After Modification	
Program	Size [bytes]	Program	Size [bytes]
`ld.w $_s, r11`	4	`ld.w $_s, r11`	4
`cmp 1, r11`	2	`cmp 1, r11`	2
`jne .L2`	2	`je .L3`	2
`mov 255, r10`	4	`ld.w $_t, r12`	4
`jbr .L1`	2	`cmp 1, r12`	2
`.L2:`		`je .L3`	2
`ld.w $_t, r12`	4	`ld.w $_u, r13`	4
`cmp 1, r12`	2	`cmp 1, r13`	2
`jne .L3`	2	`je .L3`	2
`mov 255, r10`	4	`mov r0, r10`	2
`jbr .L1`	2	`jbr .L1`	2
`.L3:`		`.L3:`	
`ld.w $_u, r13`	4	`mov 255, r10`	4
`cmp 1, r13`	2	`.L1:`	
`jne .L4`	2		
`mov 255, r10`	4		
`jbr .L1`	2		
`.L4:`			
`mov r0, r10`	2		
`.L1:`			
Total code size	44 bytes	Total code size	32 bytes

2.21 Making an Expanded Inline Function into a static Function

When a function for which inline expansion is specified is not referenced from another file, making it into a static function will cause the code for the function itself to be eliminated by the optimization. As a result, the code size may be reduced. However, when the program[Note] contains an assembly language description, the code for the function itself is output because this optimization is not performed. In this case, make the function that contains the assembly language description a separate file.

Note This refers to the program (including the include file) that is to be compiled.

Example For a static function and a normal function
Note that the function name has been changed for the sake of explanation in the Before Modification and After Modification coding.

Before Modification	After Modification

`#pragma inline func1sub`
 	
`int s,t;`
 	
`void func1sub()`
`{`
`  int tmp;`
 	
`  tmp = s;`
`  s = t;`
`  t = tmp;`
`}`
 	
`void func1()`
`{`
`  if(s == 1){`
`  func1sub();`
`  }`
`}`
	`#pragma inline func2sub`
	
	`int s,t;`
	
	`static void func2sub()`
	`{`
	`  int tmp;`
	
	`  tmp = s;`
	`  s = t;`
	`  t = tmp;`
	`}`
	
	`void func2()`
	`{`
	`  if(s == 1){`
	`  func2sub();`
	`  }`
	`}`

Although the code for the function func1sub is output, no code is output for the function func2sub.

Example For a function containing an assembly language description and a function not containing an assembly language description

Note that the function name has been changed for the sake of explanation in the Before Modification and After Modification coding.

Before Modification

```
#pragma inline func3sub

int s,t;

static void func3sub()
{
  int tmp;

  tmp = s;
  s = t;
  t = tmp;
}

void func3()
{
  if(s == 1){
    func3sub();
  }
}

void dummy(void)
{
  __asm("nop");
}
```

After Modification

```
#pragma inline func4sub

int s,t;

static void func4sub()
{
  int tmp;

  tmp = s;
  s = t;
  t = tmp;
}

void func4()
{
  if(s == 1){
    func4sub();
  }
}
```

Since the program shown in the Before Modification coding contains an assembly language description, the code for the function func3sub is output. However, since the program shown in the After Modification coding does not contain an assembly language description, no code is output for the function func4sub.

CHAPTER 3 INCREASING EXECUTION SPEED

This chapter describes examples in which the effective speed of processing that is essentially entrusted to the compiler can be increased when a human operator alters the description.

3.1 Using a Pointer for Consecutive Accesses to an Array

Use a pointer for consecutive accesses to an array within a loop. If a pointer is not used, processing for obtaining the actual address from the array subscript may be output each time.

A sample program is shown below.

Remark sum and array[ ] are assumed to be external variables.

Before Modification	After Modification

`int i;`
 	
`sum = 0;`
`for(i = 0 ; i < 10 ; i++){`
`  sum += array[i];`
`}`
	`int i,*p;`
	
	`sum = 0;`
	`p = &array[0];`
	`for(i = 0 ; i < 10 ; i++){`
	`  sum += *p++;`
	`}`

[Output Assembly List for the V850]

Before Modification			After Modification		
Program		Size [bytes]	Program		Size [bytes]
	st.w r0, $_sum	4		movea $_array, gp, r12	4
	movea $_array, gp, r10	4		st.w r0, $_sum	4
	mov r0, r12	2		mov r0, r13	2
	mov r0, r11	2		mov r0, r11	2
.L16:			.L17:		
	mov r11, r14	2		mov r12, r14	2
	shl 2, r14	2		**add 4, r12**	2
	add r10, r14	2		ld.w [r14], r14	4
	ld.w [r14], r14	4		add r14, r13	2
	add r14, r12	2		add 1, r11	2
	add 1, r11	2		st.w r13, $_sum	4
	st.w r12, $_sum	4		cmp 10, r11	2
	cmp 10, r11	2		jlt .L17	2
	jlt .L16	2			
Total code size		34 bytes	Total code size		32 bytes

[Output Assembly List for the V850E]

Before Modification		After Modification	
Program	Size [bytes]	Program	Size [bytes]
`st.w r0, $_sum`	4	`movea $_array, gp, r12`	4
`movea $_array, gp, r10`	4	`st.w r0, $_sum`	4
`mov r0, r12`	2	`mov r0, r13`	2
`mov r0, r11`	2	`mov r0, r11`	2
`.L16:`		`.L17:`	
`mov r11, r14`	2	`mov r12, r14`	2
`shl 2, r14`	2	**`add 4, r12`**	2
`add r10, r14`	2	`ld.w [r14], r14`	4
`ld.w [r14], r14`	4	`add r14, r13`	2
`add r14, r12`	2	`add 1, r11`	2
`add 1, r11`	2	`st.w r13, $_sum`	4
`st.w r12, $_sum`	4	`cmp 10, r11`	2
`cmp 10, r11`	2	`jlt .L17`	2
`jlt .L16`	2		
Total code size	34 bytes	Total code size	32 bytes

3.2 Replacing an Access to an External Variable with an Access to a Temporary Variable

Avoid using an external variable as much as possible within a loop.

Since an address calculation or memory access (load/store instruction) may be output every time, replace the access to an external variable with an access to a temporary variable.

A sample program is shown below.

Remark sum and array[ ] are assumed to be external variables.

Before Modification	After Modification

`int i;`
`int *p;`
 	
`sum = 0;`
`p = &array[0];`
`for(i = 0; i < 10; i++){`
`  sum += *p++;`
`}`
	`int i;`
	`int *p;`
	`int tmp;`
	
	`tmp = 0;`
	`p = &array[0];`
	`for(i = 0; i < 10; i++){`
	`  tmp += *p++;`
	`}`
	`sum = tmp;`

[Output Assembly List for the V850]

Before Modification			After Modification		
Program		Size [bytes]	Program		Size [bytes]
`movea`	`$_array, gp, r12`	4	`movea`	`$_array, gp, r12`	4
`st.w`	`r0, $_sum`	4	`mov`	`r0, r11`	2
`mov`	`r0, r13`	2	`mov`	`r0, r13`	2
`mov`	`r0, r11`	2	`.L18:`		
`.L17:`			`mov`	`r12, r14`	2
`mov`	`r12, r14`	2	`add`	`4, r12`	2
`add`	`4, r12`	2	`ld.w`	`[r14], r14`	4
`ld.w`	`[r14], r14`	4	**`add`**	**`r14, r13`**	2
`add`	**`r14, r13`**	2	`add`	`1, r11`	2
`add`	`1, r11`	2	`cmp`	`10, r11`	2
`st.w`	**`r13, $_sum`**	4	`jlt`	`.L18`	2
`cmp`	`10, r11`	2	`st.w`	`r13, $_sum`	4
`jlt`	`.L17`	2			
Total code size		32 bytes	Total code size		28 bytes

[Output Assembly List for the V850E]

Before Modification		After Modification	
Program	Size [bytes]	Program	Size [bytes]
`movea $_array, gp, r12`	4	`movea $_array, gp, r12`	4
`st.w r0, $_sum`	4	`mov r0, r11`	2
`mov r0, r13`	2	`mov r0, r13`	2
`mov r0, r11`	2	`.L18:`	
`.L17:`		`mov r12, r14`	2
`mov r12, r14`	2	`add 4, r12`	2
`add 4, r12`	2	`ld.w [r14], r14`	4
`ld.w [r14], r14`	4	**`add r14, r13`**	2
`add r14, r13`	2	`add 1, r11`	2
`add 1, r11`	2	`cmp 10, r11`	2
`st.w r13, $ sum`	4	`jlt .L18`	2
`cmp 10, r11`	2	`st.w r13, $_sum`	4
`jlt .L17`	2		
Total code size	32 bytes	Total code size	28 bytes

3.3 Not Using a Variable Expression for a Loop Ending Condition

Avoid using a variable expression as much as possible for a loop ending condition. Use a temporary variable instead.

If a variable expression is used, a calculation for the ending expression comparison may be output every time.

A sample program is shown below.

Remark array[ ][ ] is assumed to be an external variable.

Before Modification	After Modification

`int  i;`
`int  nSize;`
`int  mSize;`
`int  *p;`
 	
`p = &array[0][0];`
`for(i = 0; i < nSize * mSize; i++){`
`  *p++ = 0;`
`}`
	`int  i;`
	`int  nSize;`
	`int  mSize;`
	`int  *p;`
	`int  s;`
	
	`p = &array[0][0];`
	`s = nSize * mSize;`
	`for(i = 0; i < s; i++){`
	`  *p++ = 0;`
	`}`

In this example, the same code size will be produced by compiler optimization.

3.4 Using a Comparison with Zero for the Loop Ending Condition

If a comparison with zero expression is used for the loop ending condition, the calculation of the ending condition each time through the loop may become faster. Also, the number of registers that are used may be reduced.

A sample program is shown below.

Remark array[ ][ ] is assumed to be an external variable.

Before Modification	After Modification

`int i;`
`int nSize;`
`int mSize;`
`int *p;`
`int s;`
 	
`p = &array[0][0];`
`s = nSize * mSize;`
`for(i = 0; i < s; i++){`
`  *p++ = 0;`
`}`
	`int i;`
	`int nSize;`
	`int mSize;`
	`int *p;`
	
	`p = &array[0][0];`
	`for(i = nSize * mSize; i > 0; i--){`
	`  *p++ = 0;`
	`}`

[Output Assembly List for the V850]

Before Modification		After Modification	
Program	Size [bytes]	Program	Size [bytes]
`ld.w -4+0x8[sp], r7`	4	`ld.w -4+0x8[sp], r7`	4
`ld.w -8+0x8[sp], r6`	4	`ld.w -8+0x8[sp], r6`	4
`jarl ___mul, lp`	4	`jarl ___mul, lp`	4
`cmp r0, r6`	2	`cmp r0, r6`	2
`mov r6, r13`	2	`mov r6, r10`	2
`jle .L24`	2	`jle .L23`	2
`movea $_array, gp, r11`	4	`movea $_array, gp, r11`	4
`mov r0, r12`	2	`mov r10, r12`	2
`.L22:`		`.L21:`	
`mov r11, r10`	2	`mov r11, r10`	2
`add 4, r11`	2	`add 4, r11`	2
`st.w r0, [r10]`	4	`st.w r0, [r10]`	4
`add 1, r12`	2	`add 4294967295, r12`	2
`cmp r12, r13`	2	`cmp r0, r12`	2
`jgt .L22`	2	`jgt .L21`	2
`.L24:`		`.L23:`	
Total code size	38 bytes	Total code size	38 bytes

Remark In this example, the total code size does not change.

[Output Assembly List for the V850E]

Before Modification		After Modification	
Program	Size [bytes]	Program	Size [bytes]
`ld.w -4+0x8[sp], r13`	4	`ld.w -4+0x8[sp], r10`	4
`ld.w -8+0x8[sp], r14`	4	`ld.w -8+0x8[sp], r13`	4
`mul r14, r13, r0`	4	`mul r13, r10, r0`	4
`cmp r0, r13`	2	`cmp r0, r10`	2
`jle .L24`	2	`jle .L23`	2
`movea $_array, gp, r11`	4	`movea $_array, gp, r11`	4
`mov r0, r12`	2	`mov r10, r12`	2
`.L22:`		`.L21:`	
`mov r11, r10`	2	`mov r11, r10`	2
`add 4, r11`	2	`add 4, r11`	2
`st.w r0, [r10]`	4	`st.w r0, [r10]`	4
`add 1, r12`	2	`add 4294967295, r12`	2
`cmp r12, r13`	2	`cmp r0, r12`	2
`jgt .L22`	2	`jgt .L21`	2
`.L24:`		`.L23:`	
Total code size	36 bytes	Total code size	36 bytes

Remark In this example, the total code size does not change.

3.5 Unrolling a Loop

If the looping frequency is reduced, the overhead for the branching instructions due to the loop will be reduced.

A sample program is shown below.

Remark array[ ] is assumed to be an external variable.

Before Modification	After Modification

`int  i;`
`int  *p;`
 	
`p = array;`
`for(i = N; i > 0; i--){`
`  *p++ = 0;`
`}`
	`int  i;`
	`int  *p;`
	
	`p = array;`
	`for(i = N >> 2; i > 0; i--){  /* N/4 */`
	`  *p++ = 0;`
	`  *p++ = 0;`
	`  *p++ = 0;`
	`  *p++ = 0;`
	`}`
	`for(i = N & 3; i > 0; i--){  /* N mod 4 */`
	`  *p++ = 0;`
	`}`

[Output Assembly List for the V850]

Before Modification		After Modification	
Program	Size [bytes]	Program	Size [bytes]
`movea  $_array, gp, r12`	4	`mov  2, r6`	2
`mov  10, r11`	2	`movea  $_array, gp, r10`	4
`.L16:`		`mov  r6, r11`	2
`mov  r12, r10`	2	`.L16:`	
`add  4, r12`	2	**`addi  4, r10, r13`**	4
`st.w  r0, [r10]`	4	**`st.w  r0, [r10]`**	4
`add  4294967295, r11`	2	**`addi  4, r13, r14`**	4
`cmp  r0, r11`	2	**`st.w  r0, [r13]`**	4
`jgt  .L16`	2	**`addi  4, r14, r7`**	4
		`st.w  r0, [r14]`	4
		`addi  4, r7, r10`	4
		`st.w  r0, [r7]`	4
		`add  4294967295, r11`	2
		`cmp  r0, r11`	2
		`jgt  .L16`	2
		`mov  r6, r12`	2
		`.L23:`	
		`mov  r10, r11`	2
		`add  4, r10`	2
		`st.w  r0, [r11]`	4
		`add  4294967295, r12`	2
		`cmp  r0, r12`	2
		`jgt  .L23`	2
Total code size	20 bytes	Total code size	62 bytes

[Output Assembly List for the V850E]

Before Modification		After Modification	
Program	Size [bytes]	Program	Size [bytes]
`movea $_array, gp, r12`	4	`mov 2, r6`	2
`mov 10, r11`	2	`movea $_array, gp, r10`	4
`.L16:`		`mov r6, r11`	2
`mov r12, r10`	2	`.L16:`	
`add 4, r12`	2	**`addi 4, r10, r13`**	4
`st.w r0, [r10]`	4	**`st.w r0, [r10]`**	4
`add 4294967295, r11`	2	**`addi 4, r13, r14`**	4
`cmp r0, r11`	2	**`st.w r0, [r13]`**	4
`jgt .L16`	2	**`addi 4, r14, r7`**	4
		`st.w r0, [r14]`	4
		`addi 4, r7, r10`	4
		`st.w r0, [r7]`	4
		`add 4294967295, r11`	2
		`cmp r0, r11`	2
		`jgt .L16`	2
		`mov r6, r12`	2
		`.L23:`	
		`mov r10, r11`	2
		`add 4, r10`	2
		`st.w r0, [r11]`	4
		`add 4294967295, r12`	2
		`cmp r0, r12`	2
		`jgt .L23`	2
Total code size	20 bytes	Total code size	62 bytes

3.6 Optimizing a Pointer

A sample program is shown below.

Remark array[ ] is assumed to be an external variable and N is assumed to be 10.

Before Modification	After Modification

`int i;`
`int *p;`
 	
`p = array;`
`for(i = N >> 2; i > 0; i--){ /* N/4 */`
`*p++ = 0;`
`*p++ = 0;`
`*p++ = 0;`
`*p++ = 0;`
`}`
`for(i = N & 3; i > 0; i--){ /* N mod 4 */`
`*p++ = 0;`
`}`
	`int i;`
	`int *p;`
	
	`p = array;`
	`for(i = N >> 2; i > 0; i--){ /* N/4 */`
	`*(p + 0) = 0;`
	`*(p + 1) = 0;`
	`*(p + 2) = 0;`
	`*(p + 3) = 0;`
	`p += 4;`
	`}`
	`for(i = N & 3; i > 0; i--){ /* N mod 4 */`
	`*p++ = 0;`
	`}`

[Output Assembly List for the V850]

Before Modification				After Modification			
Program			Size [bytes]	Program			Size [bytes]
	mov	2, r6	2		mov	2, r10	2
	movea	$_array, gp, r10	4		movea	$_array, gp, r12	4
	mov	r6, r11	2		mov	r10, r11	2
.L16:				.L16:			
	addi	**4, r10, r13**	4		**st.w**	**r0, [r12]**	4
	st.w	**r0, [r10]**	4		**st.w**	**r0, 4[r12]**	4
	addi	**4, r13, r14**	4		**st.w**	**r0, 8[r12]**	4
	st.w	**r0, [r13]**	4		**st.w**	**r0, 12[r12]**	4
	addi	**4, r14, r7**	4		add	16, r12	4
	st.w	**r0, [r14]**	4		add	4294967295, r11	2
	addi	**4, r7, r10**	4		cmp	r0, r11	2
	st.w	**r0, [r7]**	4		jgt	.L16	2
	add	4294967295, r11	2		mov	r10, r11	2
	cmp	r0, r11	2	.L23:			
	jgt	.L16	2		mov	r12, r10	2
	mov	r6, r12	2		add	4, r12	2
.L23:					st.w	r0, [r10]	4
	mov	r10, r11	2		add	4294967295, r11	2
	add	4, r10	2		cmp	r0, r11	2
	st.w	r0, [r11]	4		jgt	.L23	2
	add	4294967295, r12	2				
	cmp	r0, r12	2				
	jgt	.L23	2				
Total code size			62 bytes	Total code size			50 bytes

[Output Assembly List for the V850E]

Before Modification		After Modification	
Program	Size [bytes]	Program	Size [bytes]
`mov 2, r6`	2	`mov 2, r10`	2
`movea $_array, gp, r10`	4	`movea $_array, gp, r12`	4
`mov r6, r11`	2	`mov r10, r11`	2
`.L16:`		`.L16:`	
`addi 4, r10, r13`	4	**`st.w r0, [r12]`**	4
`st.w r0, [r10]`	4	**`st.w r0, 4[r12]`**	4
`addi 4, r13, r14`	4	**`st.w r0, 8[r12]`**	4
`st.w r0, [r13]`	4	**`st.w r0, 12[r12]`**	4
`addi 4, r14, r7`	4	`add 16, r12`	4
`st.w r0, [r14]`	4	`add 4294967295, r11`	2
`addi 4, r7, r10`	4	`cmp r0, r11`	2
`st.w r0, [r7]`	4	`jgt .L16`	2
`add 4294967295, r11`	2	`mov r10, r11`	2
`cmp r0, r11`	2	`.L23:`	
`jgt .L16`	2	`mov r12, r10`	2
`mov r6, r12`	2	`add 4, r12`	2
`.L23:`		`st.w r0, [r10]`	4
`mov r10, r11`	2	`add 4294967295, r11`	2
`add 4, r10`	2	`cmp r0, r11`	2
`st.w r0, [r11]`	4	`jgt .L23`	2
`add 4294967295, r12`	2		
`cmp r0, r12`	2		
`jgt .L23`	2		
Total code size	62 bytes	Total code size	50 bytes

3.7 Outputting a setf Instruction to Output 0 or 1 According to the Result of a Conditional Comparison

Branch instructions are output by the if~else statement. As a result, the pipeline, which is the greatest feature of a RISC CPU, is disrupted. Coding that is not an if~else statement can be written so that a setf instruction (that outputs 0 or 1 according to the result of a conditional comparison) is output for assigning a variable.

A sample program is shown below.

Remark s and flag are assumed to be external variables.
It is also assumed that flag = 1 if the variable s is greater than 100, and flag = 0 if the variable s is less than or equal to 100.

Before Modification	After Modification

`if( s > 100){`
`  flag = 1;`
`}`
`else{`
`  flag = 0;`
`}`
	`flag = ( s > 100);`

[Output Assembly List for the V850]

Before Modification		After Modification	
Program	Size [bytes]	Program	Size [bytes]
`ld.w $_s, r10`	4	`ld.w $_s, r10`	4
`addi -100, r10, r0`	4	`cmp 100, r10`	6
`jle .L2`	2	`setfgt r11`	4
`mov 1, r11`	2	`st.w r11, $_flag`	4
`st.w r11, $_flag`	4		
`jbr .L3`	2		
`.L2:`			
`st.w r0, $_flag`	4		
`.L3:`			
Total code size	22 bytes	Total code size	18 bytes

[Output Assembly List for the V850E]

Before Modification		After Modification	
Program	Size [bytes]	Program	Size [bytes]
`ld.w $_s, r10`	4	`ld.w $_s, r10`	4
`addi -100, r10, r0`	4	`cmp 100, r10`	6
`jle .L2`	2	`setfgt r11`	4
`mov 1, r11`	2	`st.w r11, $_flag`	4
`st.w r11, $_flag`	4		
`jbr .L3`	2		
`.L2:`			
`st.w r0, $_flag`	4		
`.L3:`			
Total code size	22 bytes	Total code size	18 bytes

3.8 Using at Most Four Arguments

A function is usually called with arguments.

The values of the first four arguments are copied to registers r6, r7, r8, and r9 to call the function. The called side performs its processing by using the values of these registers as the first four arguments. The fifth and subsequent arguments are loaded in the stack when the function is called.

The called side fetches the fifth and subsequent arguments from the stack. If the number of arguments is increased from four to five, the overhead when the function is called increases because memory access is added to processing that had been accomplished by using only registers. Therefore, remember to use at most four arguments when designing the passing of arguments.

3.9 Using at Most 10 Local Variables (auto Variables) and Only 6 or 7 If Possible

Local variables (auto variables) are allocated to registers.

The V800 compiler (in 32-register mode) can use a total of 20 registers for variables within a single function since 10 registers are used as work registers and 10 registers for register variables.

If the processing within a single function is time consuming, many local variables should be used. All 20 registers can even be used for local variables.

If a function does not take very much time, it should use only the 10 work registers.

If the time overhead for pushing/popping registers that are used for register variables is taken into account, the use of register variables should be avoided. The compiler freely decides whether or not to use register variables.

Holding the number of local variables to approximately 6 or 7 is recommended.

The remaining 3 or 4 registers are used as real work registers.

3.10 Rearranging an Expression in Advance

When an expression is written, the compiler will normally expand the expression. However, the number of calculations can be reduced depending on how the expression is written.

A sample program is shown below.

Remark x and y are assumed to be external variables.

Before Modification (7 calculations)	After Modification (6 calculations)
`y = 7*x*x*x+5*x*x+x;`	`y = x*(7*x*x+5*x+1);`

[Output Assembly List for the V850]

Before Modification		After Modification	
Program	Size [bytes]	Program	Size [bytes]
`ld.w $_x, r7`	4	`ld.w $_x, r6`	4
`mov r7, r6`	2	`mov r6, r13`	2
`mov r7, r11`	2	`mov r6, r7`	2
`jarl ___mul, lp`	4	`jarl ___mul, lp`	4
`mov r6, r14`	2	`mov r6, r11`	2
`mov r6, r7`	2	`shl 3, r6`	2
`mov r11, r6`	2	`sub r11, r6`	2
`jarl ___mul, lp`	4	`mov r13, r7`	2
`mov r6, r13`	2	`shl 2, r13`	2
`shl 3, r13`	2	`add r7, r13`	2
`sub r6, r13`	2	`add r13, r6`	2
`mov r14, r10`	2	`add 1, r6`	2
`shl 2, r14`	2	`jarl ___mul, lp`	4
`add r10, r14`	2	`st.w r6, $_y`	4
`add r14, r13`	2		
`add r11, r13`	2		
`st.w r13, $_y`	4		
Total code size	42 bytes	Total code size	36 bytes

[Output Assembly List for the V850E]

Before Modification		After Modification	
Program	Size [bytes]	Program	Size [bytes]
`ld.w $_x, r11`	4	`ld.w $_x, r12`	4
`mov r11, r13`	2	`mov r12, r11`	2
`mul r11, r13, r0`	4	`mul r12, r11, r0`	4
`mov r13, r12`	2	`mul 7, r11, r0`	4
`mul r11, r12, r0`	4	`mov r12, r10`	2
`mul 7, r12, r0`	4	`mul 5, r12, r0`	4
`mul 5, r13, r0`	4	`add r12, r11`	2
`add r13, r12`	2	`add 1, r11`	2
`add r11, r12`	2	`mul r10, r11, r0`	4
`st.w r12, $_y`	4	`st.w r11, $_y`	4
Total code size	32 bytes	Total code size	32 bytes

Remark In the V850E, the same code size will be produced by compiler optimization.

3.11 Replacing a Multiplication or Division Involving a Power of 2 with a Shift Operation

If a formula contains a multiplication or division involving a power of 2 (2, 4, 8, 16, 32, ...), the execution speed can be increased by replacing it with a shift instruction.

Although this normally should be left to the compiler, a calculation that is a power of 2 should explicitly be replaced with a shift instruction when coding.

Note, however, that this method will create a problem if negative values are handled. Only use it when positive values are handled. If unsigned type is used for handling positive values, no problem will occur.

A sample program is shown below.

Remark s, t, and u are assumed to be external variables (unsigned int).

Before Modification	After Modification

`s = s / 2;`
`t = t * 8;`
`u = u * 64;`
	`s = s >> 1;`
	`t = t << 3;`
	`u = u << 6;`

In this example, the same code size will be produced by compiler optimization.

CHAPTER 4 DEFINING VARIABLES

This chapter summarizes precautions that should be kept in mind when defining variables.

4.1 Data Alignment

Data in memory must be aligned according to the device architecture. Therefore, the compiler places variables so that they are aligned properly (by inserting padding areas without changing the order).

The basic alignment conditions are that char-type data is at a one-byte boundary, short-type data is at a two-byte boundary, and int-type data is at a four-byte boundary.

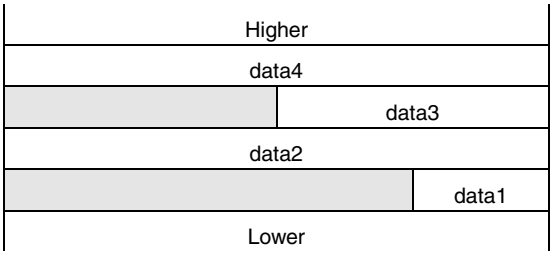
When defining variables, arrange them so that they are in decreasing data length.

A sample program is shown below.

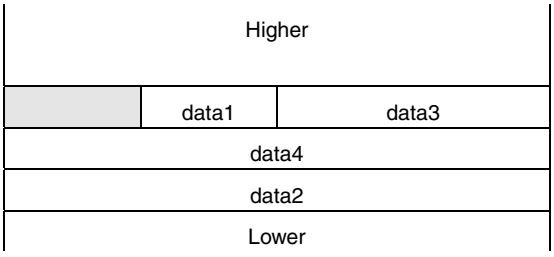
Before Modification	After Modification
struct{ char data1; long data2; short data3; long data4; }data;	struct{ long data2; long data4; short data3; char data1; }data;

The memory allocation is shown below.

Before Modification (16 bytes)



After Modification (12 bytes)



Caution The shaded areas indicate padding areas.

4.2 volatile Specification

When coding external I/Os such as ports or external variables used for interrupt servicing, include a volatile specification. If volatile is not specified, an unexpected operation may occur due to the C compiler optimization.

A sample program is shown below.

Remark a is assumed to be an external variable.

Before Modification

```
#define  PORT1    *((unsigned char  *)0x100000)      /* Address 0x100000 8 bits */
#define  PORT2    *((unsigned short *)0x100004)      /* Address 0x100000 16 bits */
#define  PORT3    *((unsigned int   *)0x100008)      /* Address 0x100000 32 bits */

struct bitf {                                        /* Bit field */
  unsigned char bit00:1;
  unsigned char bit01:1;
  unsigned char bit02:1;
  unsigned char bit03:1;
  unsigned char bit04:1;
  unsigned char bit05:1;
  unsigned char bit06:1;
  unsigned char bit07:1;
};

#define  PORTb    ((struct bitf *)0x100000)->bit00   /* Address 0x100000 0th bit */

void func()
{
         PORT1 = 0xFF;     /* Write to PORT1 */
         a = PORT1;        /* Read from PORT1 */
         PORTb = 1;        /* Set PORTb */
}
```

After Modification

```
#define  PORT1    *((volatile unsigned char  *)0x100000)     /* Address 0x100000 8 bits */
#define  PORT2    *((volatile unsigned short *)0x100004)     /* Address 0x100000 16 bits */
#define  PORT3    *((volatile unsigned int   *)0x100008)     /* Address 0x100000 32 bits */

struct bitf {                                                /* Bit field */
  unsigned char bit00:1;
  unsigned char bit01:1;
  unsigned char bit02:1;
  unsigned char bit03:1;
  unsigned char bit04:1;
  unsigned char bit05:1;
  unsigned char bit06:1;
  unsigned char bit07:1;
};

#define  PORTb    ((struct bitf *)0x100000)->bit00          /* Address 0x100000 0th bit */

void func()
{
  PORT1 = 0xFF;    /* Write to PORT1 */
  a = PORT1;       /* Read from PORT1 */
  PORTb = 1;       /* Set PORTb */
}
```

[Output Assembly List for the V850]

Before Modification		After Modification	
Program	Size [bytes]	Program	Size [bytes]
`mov 1048576, r10`	4	`mov 1048576, r10`	4
`mov 255, r11`	4	`mov 255, r11`	4
`st.b r11, [r10]`	4	`st.b r11, [r10]`	4
`st.b r11, $_a`	4	**`ld.b [r10], r12`**	4
`set1 0, [r10]`	4	`st.b r12, $_a`	4
		`set1 0, [r10]`	4
Total code size	20 bytes	Total code size	24 bytes

[Output Assembly List for the V850E]

Before Modification		After Modification	
Program	Size [bytes]	Program	Size [bytes]
`mov 1048576, r10`	4	`mov 1048576, r10`	4
`mov 255, r12`	4	`mov 255, r11`	4
`st.b r12, [r10]`	4	`st.b r11, [r10]`	4
`sxb r12`	2	**`ld.b [r10], r12`**	4
`st.b r12, $_a`	4	`st.b r12, $_a`	4
`set1 0, [r10]`	4	`set1 0, [r10]`	4
Total code size	22 bytes	Total code size	24 bytes

4.3 Read-Only Variables

Specify the const variable as the variable for read only.
The variable definition will be allocated to ROM, reducing the size of the RAM variable area.

4.4 Reducing Alignment Between Files

When variables are defined in multiple files, an alignment hole may be generated between files during linking.
The alignment hole between files can be avoided by gathering variable definitions in 1 file.

A sample program is shown below.

Before Modification	After Modification

`-- sub1_1.c --`
`long data1 = 0;`
`char data2 = 0;`
 	
`-- sub1_2.c --`
`char data3 = 0;`
`char data4 = 0;`
	`-- sub2.c --`
	`long data1 = 0;`
	`char data2 = 0;`
	`char data3 = 0;`
	`char data4 = 0;`

[Link map]

Before Modification				
.sdata		0x00ffe00c	0x0000000a	
	.sdata	0x00ffe00c	0x00000005	sub1_1.o
		0x00ffe011	0x00000003	*(align-hole)*
	.sdata	0x00ffe014	0x00000002	sub1_2.o

After Modification				
.sdata		0x00ffe00c	0x00000007	
	.sdata	0x00ffe00c	0x00000007	sub2.o

4.5 Consolidating Flags

Consolidate flags that have a capacity within a few bits in bit field.

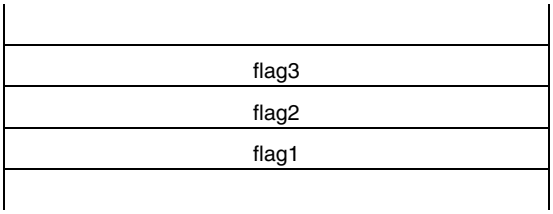
A sample program is shown below.

Before Modification	After Modification

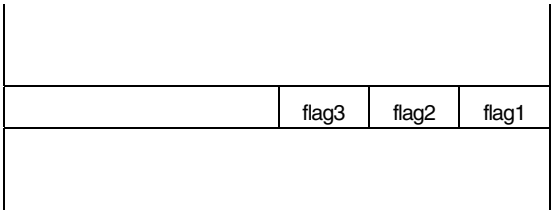
`unsigned char flag1;`
`unsigned char flag2;`
`unsigned char flag3;`
 	
`flag1 = 1;`
`flag2 = 1;`
`flag3 = 1;`
	`struct bitf {`
	`  unsigned char flag1:1;`
	`  unsigned char flag2:1;`
	`  unsigned char flag3:1;`
	`} flags;`
	
	`flags.flag1 = 1;`
	`flags.flag2 = 1;`
	`flags.flag3 = 1;`

The memory allocation is shown below.

Before Modification



After Modification



4.6 Reducing Nesting Levels of Functions

Reducing the nesting of functions by changing the algorithm may reduce the usage rate of the stack. Since the operation may not be affected much by inline expansion, the algorithm must be changed.

APPENDIX INDEX

[W]

[MEMO]

NEC

Facsimile Message

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.

From:

Name

Company

Tel. FAX

Address

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
Market Communication 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

P.R. China
NEC Electronics Shanghai, Ltd.
Fax: +86-21-6841-1137

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	❑	❑	❑	❑