Application Guide for the CC-RH C Compiler for RH850 Devices: Programming Techniques

R20UT3845EJ0100
Rev.1.00
Sep. 08, 2016

Contents

1. Introduction

This application guide describes methods of programming for efficiency in terms of code size, speed of execution, and data size for users developing programs with the CC-RH compiler for RH850 devices.

1.1 Overview

The methods of programming which are efficient in terms of code size, execution speed, and data size are classified under the following three headings.

- Options
- Extended Language
- Methods of coding

Note that descriptions of items such as examples of generated code may be changed due to upgrading of the CC-RH compiler.

2. Options

This chapter describes the effects on code size, speed of execution, and data size when options for CC-RH are specified.

The degrees of the effects depend on the details of the source code.

2.1 Compiler Options

√: Improved, x: Worsened, Δ: Depends on the situation, —: No effect

Option	Code Size	Speed of Execution	Data Size	Remarks
-Xenum_type=auto	Δ	Δ	√	In the case of variables of small types, this may lead to the generation of sign-extended or zero-extended operands in instructions. In this way, the effects on code size and speed depend on the situation.
-Xvolatile	x	x	—	
-Onothing	x	x	x	Since static variables that are only referred to by unused functions are not deleted, this option may also have an adverse effect on data size.
-Odefault	—	—	—	
-Osize	√	Δ	√	The deletion of static variables that are only referred to by unused functions has a positive effect on data size.
-Ospeed	Δ	√	√	The deletion of static variables that are only referred to by unused functions has a positive effect on data size.
-Ounroll=small value	√	x	—	
-Ounroll=large value	x	√	—	
-Oinline=0	√	x	—	
-Oinline=1	—	—	—	
-Oinline=2	x	√	—	
-Oinline=3	√	√	—	
-Oinline_size=small value	√	x	—	
-Oinline_size=large value	x	√	—	
-Odelete_static_func=on	√	—	√	The deletion of static variables that are only referred to by unused functions has a positive effect on data size.
-Opipeline=on	—	√	—	
-Otail_call=on	√	√	—	
-Omap	√	√	—	
-Osmap	√	√	—	
-Xintermodule	√	√	—	
-Xinline_strcpy	x	√	—	
-Xmerge_string	—	—	√	
-Xalias=ansi	√	√	—	
-Xmerge_files	√	√	—	
-Xwhole_program	√	√	—	
-Xpack=1,2	x	x	√	
-Xbit_order	—	—	—	

-Xswitch=ifelse	Δ	Δ	—	This improves efficiency if there are few labels in switch statements.
-Xswitch=binary	x	Δ	—	This improves efficiency if there are many labels in switch statements.
-Xswitch=table	Δ	Δ	—	This improves efficiency if there are few labels in switch statements.
-Xreg_mode=22,common	Δ	Δ	—	If functions take up many registers, this will worsen both the code size and speed.
-Xreserve_r2	Δ	Δ	—	If functions take up many registers, this will worsen both the code size and speed.
-Xep=fix	x	x	—	If functions take up many registers, this will worsen both the code size and speed. When the –Omap option is specified, the -Xep=fix option must also be specified.
-Xfloat=soft	x	x	—	
-Xcall_jump=32	x	x	—	
-Xfar_jump	x	x	—	
-Xdiv	—	x	—	If this option is omitted, divq or divqu instructions will be generated to handle division. Although these instructions run at high speed, the number of cycles for execution differs with the values of the operands.
-Xcheck_div_ov	x	x	—	
-Xuse_fmaf	√	√	—	This option may change the result of executing the program.
-Xunordered_cmpf	x	x	—	
-Xmulti_level	—	—	—	
-Xpatch=dw_access	x	x	—	Usage notes on the MCU may become applicable if this option is not specified.
-Xpatch=switch	x	x	—	Usage notes on the MCU may become applicable if this option is not specified.
-Xpatch=syncp	x	x	—	Usage notes on the MCU may become applicable if this option is not specified.
-Xdbl_size=4	√	√	√	This option may change the result of executing the program.
-Xround=zero	—	—	—	This option may change the result of executing the program.
-Xalign4	x	√	—	
-Xstack_protector	x	x	—	This option can only be specified if the compiler is covered by a registered license for the professional edition.
-Xsection	√	√	—	

2.2 Assembler Option

√: Improved, x: Worsened, Δ: Depends on the situation, —: No effect

Option	**Code Size**	**Speed of Execution**	**Data Size**	**Remarks**
-Xasm_far_jump	x	x	—	

2.3 Link options

√: Improved, x: Worsened, Δ: Depends on the situation, —: No effect

Option	Code Size	Speed of Execution	Data Size	Remarks
-padding	x	—	x	
-overrun_fetch	x	—	x	
-aligned_section	x	√	x	

3. Extended Language

This chapter describes the effects on code size, speed of execution, and data size of #pragma directives among the extended language.

3.1 #pragma Directives

√: Improved, x: Worsened, Δ: Depends on the situation, —: No effect

Directive	Code Size	Speed of Execution	Data Size	Remarks
#pragma section	√	√	Δ	Specifying the following attributes improves efficiency in terms of code size and speed of execution. r0_disp16, r0_disp23, ep_disp4, ep_disp5, ep_disp7, ep_disp8, ep_disp16, ep_disp23, gp_disp16, gp_disp23, zconst, zconst23 In the case of many small sections, empty spaces are generated between the sections and this may increase the data size.
#pragma inline	x	√	—	
#pragma noinline	√	x	—	
#pragma interrupt	—	—	—	When some registers need not be saved and restored and the interrupt specifications are changed (enable, fpu, and callt and so on), this improves the code size and speed of execution.
#pragma pack	x	x	√	
#pragma align4	x	√	—	
#pragma stack_protector	x	x	—	This option can only be specified if the compiler is covered by a registered license for the professional edition.

4. Methods of Coding

This chapter describes the effects on code size, speed of execution, and data size through particular methods for the coding of user programs.

4.1 Reducing the Code Size

The following shows examples before and after modification when the –Osize option for CC-RH V1.04.00 is specified.

4.1.1 Using if-else Statements Instead of switch Statements

A switch statement uses a table jump for the branch code, so the generated branch code consists of two-stage branches, which increases the code size. Branch code for values which are not among the case labels may also be included.

Altering switch statements for which the case labels are few to take the form of if statements may reduce the code size. The recommended number of case labels for this approach is fewer than five.

Note: Similar efficiency may be obtained by the -Xswitch=ifelse option.

As a Switch Statement	As if-else Statements

<pre>int fw(int x) {
  switch(x) {
  case 0:
    sub0();
    break;
  case 1:
    sub1();
    break;
  case 2:
    sub2();
    break;
  case 3:
    sub3();
    break;
  }
  return 0;
}</pre>
	<pre>int fb(int x) {
	  if (x == 0) {
	    sub0();
	  }
	  else if (x == 1) {
	    sub1();
	  }
	  else if (x == 2) {
	    sub2();
	  }
	  else if (x == 3) {
	    sub3();
	  }
	  return 0;
	}</pre>

`_fw:`
`        .stack _fw = 4`
`        prepare 0x00000001, 0x00000000`
`        cmp 0x00000003, r6`
`        bh9 .BB.LABEL.1_6`
`.BB.LABEL.1_1:  ; entry`
`        shl 0x00000001, r6`
`        jmp #.SWITCH.LABEL.1_7[r6]`
`.SWITCH.LABEL.1_7:`
`        br9 .BB.LABEL.1_2`
`        br9 .BB.LABEL.1_3`
`        br9 .BB.LABEL.1_4`
`        br9 .BB.LABEL.1_5`
`.SWITCH.LABEL.1_7.END:`
`.BB.LABEL.1_2:  ; bb`
`        jarl _sub0, r31`
`        br9 .BB.LABEL.1_6`
`.BB.LABEL.1_3:  ; bb3`
`        jarl _sub1, r31`
`        br9 .BB.LABEL.1_6`
`.BB.LABEL.1_4:  ; bb5`
`        jarl _sub2, r31`
`        br9 .BB.LABEL.1_6`
`.BB.LABEL.1_5:  ; bb7`
`        jarl _sub3, r31`
`.BB.LABEL.1_6:  ; bb9`
`        mov 0x00000000, r10`
`        dispose 0x00000000, 0x00000001, [r31]`
	`_fb:`
	`        .stack _fb = 4`
	`        prepare 0x00000001, 0x00000000`
	`        cmp 0x00000000, r6`
	`        bnz9 .BB.LABEL.2_2`
	`.BB.LABEL.2_1:  ; if_then_bb`
	`        jarl _sub0, r31`
	`        br9 .BB.LABEL.2_8`
	`.BB.LABEL.2_2:  ; if_else_bb`
	`        cmp 0x00000001, r6`
	`        bnz9 .BB.LABEL.2_4`
	`.BB.LABEL.2_3:  ; if_then_bb9`
	`        jarl _sub1, r31`
	`        br9 .BB.LABEL.2_8`
	`.BB.LABEL.2_4:  ; if_else_bb11`
	`        cmp 0x00000002, r6`
	`        bnz9 .BB.LABEL.2_6`
	`.BB.LABEL.2_5:  ; if_then_bb16`
	`        jarl _sub2, r31`
	`        br9 .BB.LABEL.2_8`
	`.BB.LABEL.2_6:  ; if_else_bb18`
	`        cmp 0x00000003, r6`
	`        bnz9 .BB.LABEL.2_8`
	`.BB.LABEL.2_7:  ; if_then_bb23`
	`        jarl _sub3, r31`
	`.BB.LABEL.2_8:  ; if_break_bb27`
	`        mov 0x00000000, r10`
	`        dispose 0x00000000, 0x00000001, [r31]`

52 bytes	48 bytes

4.1.2 Unifying Multiple Exit Points for Functions

If a switch statement or if-else statement contains the exit points of a function, multiple segments of code for exit from the function will be generated. Adding a return statement to the end of the function leads to a single code segment for exit from the function, and this may reduce the code size.

<table>
<tr><th>Multiple Exit Points</th><th>Single Exit Point</th></tr>
<tr><td><pre>
extern int s;
void fw (int x) {
  switch (x) {
  case 0:
    s = 0;
    break;
  case 1000:
    s = 0x5555;
    break;
  case 2000:
    s = 0xAAAA;
    break;
  case 3000:
    s = 0xFFFF;
    break;
  default:
    break;
  }
}
</pre></td><td><pre>
extern int s;
int fb (int x) {
  switch (x) {
  case 0:
    s = 0;
    break;
  case 1000:
    s = 0x5555;
    break;
  case 2000:
    s = 0xAAAA;
    break;
  case 3000:
    s = 0xFFFF;
    break;
  default:
    break;
  }
  return 0;
}
</pre></td></tr>
<tr><td><pre>
_fw:
        .stack _fw = 0
        cmp 0x00000000, r6
        movhi HIGHW1(#_s), r0, r2
        bz9 .BB.LABEL.1_5
.BB.LABEL.1_1: ; entry
        addi 0xFFFFFC18, r6, r5
        bz9 .BB.LABEL.1_6
.BB.LABEL.1_2: ; entry
        addi 0xFFFFFC18, r5, r5
        bz9 .BB.LABEL.1_7
.BB.LABEL.1_3: ; entry
        addi 0xFFFFFC18, r5, r0
        bz9 .BB.LABEL.1_8
.BB.LABEL.1_4: ; return
        jmp [r31]
.BB.LABEL.1_5: ; bb
        st.w r0, LOWW(#_s)[r2]
        jmp [r31]
.BB.LABEL.1_6: ; bb1
        movea 0x00005555, r0, r5
        st.w r5, LOWW(#_s)[r2]
        jmp [r31]
.BB.LABEL.1_7: ; bb2
        ori 0x0000AAAA, r0, r5
        st.w r5, LOWW(#_s)[r2]
        jmp [r31]
.BB.LABEL.1_8: ; bb3
        ori 0x0000FFFF, r0, r5
        st.w r5, LOWW(#_s)[r2]
        jmp [r31]
</pre></td><td><pre>
_fb:
        .stack _fb = 0
        cmp 0x00000000, r6
        movhi HIGHW1(#_s), r0, r2
        bz9 .BB.LABEL.2_5
.BB.LABEL.2_1: ; entry
        addi 0xFFFFFC18, r6, r5
        bz9 .BB.LABEL.2_6
.BB.LABEL.2_2: ; entry
        addi 0xFFFFFC18, r5, r5
        bz9 .BB.LABEL.2_7
.BB.LABEL.2_3: ; entry
        addi 0xFFFFFC18, r5, r0
        bz9 .BB.LABEL.2_9
.BB.LABEL.2_4: ; bb6
        mov 0x00000000, r10
        jmp [r31]
.BB.LABEL.2_5: ; bb
        st.w r0, LOWW(#_s)[r2]
        br9 .BB.LABEL.2_4
.BB.LABEL.2_6: ; bb2
        movea 0x00005555, r0, r5
        br9 .BB.LABEL.2_8
.BB.LABEL.2_7: ; bb3
        ori 0x0000AAAA, r0, r5
.BB.LABEL.2_8: ; bb3
        st.w r5, LOWW(#_s)[r2]
        br9 .BB.LABEL.2_4
.BB.LABEL.2_9: ; bb4
        ori 0x0000FFFF, r0, r5
        br9 .BB.LABEL.2_8
</pre></td></tr>
<tr><td>64 bytes</td><td>58 bytes</td></tr>
</table>

4.1.3 Using Temporary Variables to Reduce Multiple Lines of Code for Access to an External Variable into a Single Line for This Purpose

Accessing to temporary variables is more likely to generate register transfer code than accessing to an external variable. When code for accessing to an external variable is decreased by using temporary variables, the code size may be reduced.

Multiple Access to an External Variable	Single Access to an External Variable

<pre>extern int s;
int fw(int x) {
  switch (x) {
  case 0:
    s = 0;
    break;
  case 1000:
    s = 0x5555;
    break;
  case 2000:
    s = 0xAAAA;
    break;
  case 3000:
    s = 0xFFFF;
  }
  return 0;
}</pre>
	<pre>extern int s;
	int fb(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:
	  return 0;
	}</pre>

<pre>_fw:
        .stack _fw = 0
        cmp 0x00000000, r6
        movhi HIGHW1(#_s), r0, r2
        bz9 .BB.LABEL.1_5
.BB.LABEL.1_1: ; entry
        addi 0xFFFFFC18, r6, r5
        bz9 .BB.LABEL.1_6
.BB.LABEL.1_2: ; entry
        addi 0xFFFFFC18, r5, r5
        bz9 .BB.LABEL.1_7
.BB.LABEL.1_3: ; entry
        addi 0xFFFFFC18, r5, r0
        bz9 .BB.LABEL.1_9
.BB.LABEL.1_4: ; bb5
        mov 0x00000000, r10
        jmp [r31]
.BB.LABEL.1_5: ; bb
        st.w r0, LOWW(#_s)[r2]
        br9 .BB.LABEL.1_4
.BB.LABEL.1_6: ; bb2
        movea 0x00005555, r0, r5
        br9 .BB.LABEL.1_8
.BB.LABEL.1_7: ; bb3
        ori 0x0000AAAA, r0, r5
.BB.LABEL.1_8: ; bb3
        st.w r5, LOWW(#_s)[r2]
        br9 .BB.LABEL.1_4
.BB.LABEL.1_9: ; bb4
        ori 0x0000FFFF, r0, r5
        br9 .BB.LABEL.1_8</pre>
	<pre>_fb:
	        .stack _fb = 0
	        cmp 0x00000000, r6
	        mov 0x00000000, r2
	        bz9 .BB.LABEL.2_6
	.BB.LABEL.2_1: ; if_else_bb
	        addi 0xFFFFFC18, r6, r0
	        movea 0x00005555, r0, r2
	        bz9 .BB.LABEL.2_6
	.BB.LABEL.2_2: ; if_else_bb10
	        addi 0xFFFFF830, r6, r0
	        bnz9 .BB.LABEL.2_4
	.BB.LABEL.2_3: ;
	if_else_bb10.if_break_bb25_crit_edge
	        ori 0x0000AAAA, r0, r2
	        br9 .BB.LABEL.2_6
	.BB.LABEL.2_4: ; if_else_bb16
	        addi 0xFFFFF448, r6, r0
	        bnz9 .BB.LABEL.2_7
	.BB.LABEL.2_5: ;
	if_else_bb16.if_break_bb25_crit_edge
	        ori 0x0000FFFF, r0, r2
	.BB.LABEL.2_6: ; if_break_bb25
	        movhi HIGHW1(#_s), r0, r5
	        st.w r2, LOWW(#_s)[r5]
	.BB.LABEL.2_7: ; label
	        mov 0x00000000, r10
	        jmp [r31]</pre>

58 bytes	50 bytes

4.1.4 Moving Identical Expressions in More than One Conditional Branch Destination before the Conditional Branch

When there are identical expressions in more than one conditional branch destination, move and unify them into one section before the conditional branch.

Identical Expressions Following a Branch	Expression before the Branch

<pre>extern int s;
int fw(int x) {
  if (x >= 0) {
    if (x > func(0, 1, 2)) {
      s++;
    }
  }
  else {
    if (x < -func(0, 1, 2)) {
      s--;
    }
  }
  return 0;
}</pre>
	<pre>extern int s;
	int fb(int x) {
	  int tmp;
	  tmp = func(0, 1, 2);
	  if (x >= 0) {
	    if (x > **tmp**) {
	      s++;
	    }
	  }
	  else {
	    if (x < **-tmp**) {
	      s--;
	    }
	  }
	  return 0;
	}</pre>

<table>
<tr>
<td>

```
_fw:
        .stack _fw = 12
        prepare 0x00000061, 0x00000000
        addi 0x00000000, r6, r20
        movhi HIGHW1(#_s), r0, r21
        bn9 .BB.LABEL.1_3
.BB.LABEL.1_1:  ; if_then_bb
        mov 0x00000002, r8
        mov 0x00000001, r7
        mov 0x00000000, r6
        jarl _func, r31
        cmp r20, r10
        bge9 .BB.LABEL.1_5
.BB.LABEL.1_2:  ; if_then_bb9
        ld.w LOWW(#_s)[r21], r20
        add 0x00000001, r20
        st.w r20, LOWW(#_s)[r21]
        br9 .BB.LABEL.1_5
.BB.LABEL.1_3:  ; if_else_bb
        mov 0x00000002, r8
        mov 0x00000001, r7
        mov 0x00000000, r6
        jarl _func, r31
        subr r0, r10
        cmp r10, r20
        bge9 .BB.LABEL.1_5
.BB.LABEL.1_4:  ; if_then_bb18
        ld.w LOWW(#_s)[r21], r2
        add 0xFFFFFFFF, r2
        st.w r2, LOWW(#_s)[r21]
.BB.LABEL.1_5:  ; if_break_bb22
        mov 0x00000000, r10
        dispose 0x00000000, 0x00000061, [r31]
```

</td>
<td>

```
_fb:
        .stack _fb = 8
        prepare 0x00000041, 0x00000000
        mov r6, r20
        mov 0x00000002, r8
        mov 0x00000001, r7
        mov 0x00000000, r6
        jarl _func, r31
        cmp 0x00000000, r20
        movhi HIGHW1(#_s), r0, r2
        bn9 .BB.LABEL.2_3
.BB.LABEL.2_1:  ; if_then_bb
        cmp r20, r10
        bge9 .BB.LABEL.2_5
.BB.LABEL.2_2:  ; if_then_bb11
        ld.w LOWW(#_s)[r2], r20
        add 0x00000001, r20
        st.w r20, LOWW(#_s)[r2]
        br9 .BB.LABEL.2_5
.BB.LABEL.2_3:  ; if_else_bb
        subr r0, r10
        cmp r10, r20
        bge9 .BB.LABEL.2_5
.BB.LABEL.2_4:  ; if_then_bb20
        ld.w LOWW(#_s)[r2], r5
        add 0xFFFFFFFF, r5
        st.w r5, LOWW(#_s)[r2]
.BB.LABEL.2_5:  ; if_break_bb24
        mov 0x00000000, r10
```

</td>
</tr>
<tr>
<td>72 bytes</td>
<td>62 bytes</td>
</tr>
</table>

4.1.5 Replacing a Sequence of Complicated if Statement with a Simple Statement Having the Same Logical Meaning

When a sequence of if statements and conditional expressions is complicated, replace them with a simple expression which has the same meaning.

<table>
<tr><th>Complicated Sequence</th><th>Single if Statement</th></tr>
<tr><td><pre>
extern int x;
int fw (int s, int t) {
  s &= 1;
  t &= 1;
  if (!s) {
    if (t) {
      x = 1;
    }
  }
  else {
    if (!t) {
      x = 1;
    }
  }
  return 0;
}
</pre></td><td><pre>
extern int x;
int fb (int s, int t) {
  s &= 1;
  t &= 1;
  if (! (s ^ t)) {
    x = 1;
  }
  return 0;
}
</pre></td></tr>
<tr><td><pre>
_fw:
        .stack _fw = 0
        andi 0x00000001, r7, r2
        andi 0x00000001, r6, r0
        movhi HIGHW1(#_x), r0, r5
        bnz9 .BB.LABEL.1_2
.BB.LABEL.1_1: ; bb9.thread
        cmp 0x00000000, r2
        bnz9 .BB.LABEL.1_3
        br9 .BB.LABEL.1_4
.BB.LABEL.1_2: ; if_else_bb
        cmp 0x00000000, r2
        bnz9 .BB.LABEL.1_4
.BB.LABEL.1_3: ; if_then_bb27
        mov 0x00000001, r2
        st.w r2, LOWW(#_x)[r5]
.BB.LABEL.1_4: ; if_break_bb29
        mov 0x00000000, r10
        jmp [r31]
</pre></td><td><pre>
_fb:
        .stack _fb = 0
        xor r6, r7
        andi 0x00000001, r7, r0
        bnz9 .BB.LABEL.2_2
.BB.LABEL.2_1: ; if_then_bb
        movhi HIGHW1(#_x), r0, r2
        mov 0x00000001, r5
        st.w r5, LOWW(#_x)[r2]
.BB.LABEL.2_2: ; if_break_bb
        mov 0x00000000, r10
        jmp [r31]
</pre></td></tr>
<tr><td>34 bytes</td><td>22 bytes</td></tr>
</table>

4.1.6 Converting short- or char-Type Variables into the int Type

In accord with the ANSI-C specification, the CC-RH compiler converts short- or char-type operations into the int type before generating code for the operations. Type conversion is also produced when an int-type value is substituted for a short- or char-type variable. Defining variables as the int type in the first place can reduce additional type conversion.

Note: When the type of a variable is converted into the int type, the range of variables or values obtained by the operation will be changed. If you change the type, take care that this does not affect the operation of the program.

char-Type Variables	int-Type Variables

`unsigned char fw(unsigned char a, unsigned char b, unsigned char c) {`
`  unsigned char t = a + b;`
`  return t >> c;`
`}`
	`unsigned int fb(`**`unsigned int a, unsigned int b, unsigned int c`**`) {`
	`  unsigned int t = a + b;`
	`  return t >> c;`
	`}`

`_fw:`
`    .stack _fw = 0`
`    add r6, r7`
`    zxb r7`
`    shr r8, r7`
`    mov r7, r10`
`    zxb r10`
`    jmp [r31]`
	`_fb:`
	`    .stack _fb = 0`
	`    add r6, r7`
	`    mov r7, r10`
	`    shr r8, r10`
	`    jmp [r31]`

14 bytes	10 bytes

4.1.7 Unifying Common case Processing in switch Statements

When the branch destinations of multiple case labels have the same processing, move the case labels and unify the processing.

Same Processing at Multiple Destinations	Unified Processing

<pre>extern int x;
int fw (void) {
 switch(x) {
  case 0:
    dummy1();
    break;
  case 1:
    dummy1();
    break;
  case 2:
    dummy1();
    break;
  case 3:
    dummy2();
    break;
  case 4:
    dummy2();
    break;
  default:
    break;
 }
}</pre>
	<pre>extern int x;
	int fb (void) {
	 switch(x) {
	  case 0:
	  case 1:
	  case 2:
	    dummy1();
	    break;
	  case 3:
	  case 4:
	    dummy2();
	    break;
	  default:
	    break;
	 }
	}</pre>

<pre>_fw:
        .stack _fw = 0
        movhi HIGHW1(#_x), r0, r2
        ld.w LOWW(#_x)[r2], r2
        cmp 0x00000004, r2
        bh9 .BB.LABEL.1_3
.BB.LABEL.1_1: ; entry
        shl 0x00000001, r2
        jmp #.SWITCH.LABEL.1_6[r2]
.SWITCH.LABEL.1_6:
        br9 .BB.LABEL.1_2
        br9 .BB.LABEL.1_4
        br9 .BB.LABEL.1_4
        br9 .BB.LABEL.1_5
        br9 .BB.LABEL.1_5
.SWITCH.LABEL.1_6.END:
.BB.LABEL.1_2: ; bb
        jr _dummy1
.BB.LABEL.1_3: ; return
        jmp [r31]
.BB.LABEL.1_4: ; bb5
        jr _dummy1
.BB.LABEL.1_5: ; bb9
        jr _dummy2</pre>
	<pre>_fb:
	        .stack _fb = 0
	        movhi HIGHW1(#_x), r0, r2
	        ld.w LOWW(#_x)[r2], r2
	        cmp 0x00000003, r2
	        bl9 .BB.LABEL.2_3
	.BB.LABEL.2_1: ; entry
	        add 0xFFFFFFFD, r2
	        cmp 0x00000002, r2
	        bl9 .BB.LABEL.2_4
	.BB.LABEL.2_2: ; return
	        jmp [r31]
	.BB.LABEL.2_3: ; bb
	        jr _dummy1
	.BB.LABEL.2_4: ; bb3
	        jr _dummy2</pre>

44 bytes	28 bytes

4.1.8 Changing Inline-Expanded Functions to Static Functions

When other source files do not refer to an inline-expanded function, change the function to a static function. Some code in the function will be removed and the code size may be reduced.

<table>
<tr><th>Inline-Expanded Function</th><th>Static Function</th></tr>
<tr><td><pre>
extern int s, t;
#pragma inline fwsub
void fwsub() {
  int tmp;
  tmp = s;
  s = t;
  t = tmp;
}
void fw() {
  if(s == 1){
    fwsub();
  }
}
</pre></td><td><pre>
extern int s, t;
#pragma inline fbsub
static void fbsub() {
  int tmp;
  tmp = s;
  s = t;
  t = tmp;
}
void fb()
{
  if(s == 1){
    fbsub();
  }
}
</pre></td></tr>
<tr><td><pre>
_fwsub:
        .stack _fwsub = 0
        movhi HIGHW1(#_t), r0, r2
        ld.w LOWW(#_t)[r2], r5
        movhi HIGHW1(#_s), r0, r6
        ld.w LOWW(#_s)[r6], r7
        st.w r5, LOWW(#_s)[r6]
        st.w r7, LOWW(#_t)[r2]
        jmp [r31]
_fw:
        .stack _fw = 0
        movhi HIGHW1(#_s), r0, r2
        ld.w LOWW(#_s)[r2], r5
        cmp 0x00000001, r5
        bnz9 .BB.LABEL.2_2
.BB.LABEL.2_1: ; if_then_bb
        movhi HIGHW1(#_t), r0, r6
        ld.w LOWW(#_t)[r6], r7
        st.w r7, LOWW(#_s)[r2]
        st.w r5, LOWW(#_t)[r6]
.BB.LABEL.2_2: ; return
        jmp [r31]
</pre></td><td><pre>
_fb:
        .stack _fb = 0
        movhi HIGHW1(#_s), r0, r2
        ld.w LOWW(#_s)[r2], r5
        cmp 0x00000001, r5
        bnz9 .BB.LABEL.3_2
.BB.LABEL.3_1: ; if_then_bb
        movhi HIGHW1(#_t), r0, r6
        ld.w LOWW(#_t)[r6], r7
        st.w r7, LOWW(#_s)[r2]
        st.w r5, LOWW(#_t)[r6]
.BB.LABEL.3_2: ; return
        jmp [r31]
</pre></td></tr>
<tr><td>56 bytes</td><td>30 bytes</td></tr>
</table>

4.1.9 Using Comparison to 0 as the Condition for Ending Loop Iterations

Using comparison to 0 as the condition to end iterations of a loop may eliminate the need for a register to retain the condition of the number of loops and thus reduce the code size.

<table>
<tr><th>Comparison with a Non-Zero Variable</th><th>Comparison with Zero</th></tr>
<tr><td><pre>
extern int array[10][10];
void fw(int nSize, int mSize) {
  int i;
  int *p;
  int s;
  p = &array[0][0];
  s = nSize * mSize;
  for(i = 0; i < s; i++){
    *p++ = 0;
  }
}
</pre></td><td><pre>
extern int array[10][10];
void fb(int nSize, int mSize) {
  int i;
  int *p;
  p = &array[0][0];
  for(i = nSize * mSize; i > 0; i--){
    *p++ = 0;
  }
}
</pre></td></tr>
<tr><td><pre>
_fw:
        .stack _fw = 0
        mul r6, r7, r0
        mov 0x00000000, r2
        mov #_array, r5
.BB.LABEL.1_1: ; bb8
        cmp r7, r2
        bge9 .BB.LABEL.1_3
.BB.LABEL.1_2: ; bb
        st.w r0, 0x00000000[r5]
        add 0x00000004, r5
        add 0x00000001, r2
        br9 .BB.LABEL.1_1
.BB.LABEL.1_3: ; return
        jmp [r31]
</pre></td><td><pre>
_fb:
        .stack _fb = 0
        mul r6, r7, r0
        mov #_array, r2
.BB.LABEL.2_1: ; bb8
        cmp 0x00000000, r7
        ble9 .BB.LABEL.2_3
.BB.LABEL.2_2: ; bb
        st.w r0, 0x00000000[r2]
        add 0x00000004, r2
        add 0xFFFFFFFF, r7
        br9 .BB.LABEL.2_1
.BB.LABEL.2_3: ; return
        jmp [r31]
</pre></td></tr>
<tr><td>28 bytes</td><td>26 bytes</td></tr>
</table>

4.1.10 Replacing for Loops with do-while Loops

Replaced a for statement with a do-while statement if it is clear that the loop is executed at least once may reduce the code size.

<table>
<tr><th>for Loop</th><th>do-while Loop</th></tr>
<tr><td><pre>extern int array[10][10];
void fw(int nSize, int mSize) {
  int i;
  int *p;
  int s;
  p = &array[0][0];
  s = nSize * mSize;
  for(i = 0; i < s; i++){
    *p++ = 0;
  }
}</pre></td><td><pre>extern int array[10][10];
void fb(int nSize, int mSize) {
  int i;
  int *p;
  int s;
  p = &array[0][0];
  s = nSize * mSize;
  i = 0;
  do {
    *p++ = 0;
    i++;
  } while (i < s);
}</pre></td></tr>
<tr><td><pre>_fw:
        .stack _fw = 0
        mul r6, r7, r0
        mov 0x00000000, r2
        mov #_array, r5
.BB.LABEL.1_1: ; bb8
        cmp r7, r2
        bge9 .BB.LABEL.1_3
.BB.LABEL.1_2: ; bb
        st.w r0, 0x00000000[r5]
        add 0x00000004, r5
        add 0x00000001, r2
        br9 .BB.LABEL.1_1
.BB.LABEL.1_3: ; return
        jmp [r31]</pre></td><td><pre>_fb:
        .stack _fb = 0
        mul r6, r7, r0
        mov 0x00000000, r2
        mov #_array, r5
.BB.LABEL.2_1: ; bb
        st.w r0, 0x00000000[r5]
        add 0x00000004, r5
        add 0x00000001, r2
        cmp r7, r2
        blt9 .BB.LABEL.2_1
.BB.LABEL.2_2: ; return
        jmp [r31]</pre></td></tr>
<tr><td>28 bytes</td><td>26 bytes</td></tr>
</table>

Replacing another kind of conditional expression with an equality or inequality operator may also reduce the code size.

<table>
<tr><th>Comparison</th><th>Inequality</th></tr>
<tr><td><pre>
extern int array[10][10];
void fw(int nSize, int mSize) {
  int i;
  int *p;
  int s;
  p = &array[0][0];
  s = nSize * mSize;
  for(i = 0; i < s; i++){
    *p++ = 0;
  }
}
</pre></td><td><pre>
extern int array[10][10];
void fb2(int nSize, int mSize) {
  int i;
  int *p;
  int s;
  p = &array[0][0];
  s = nSize * mSize;
  i = 0;
  do {
    *p++ = 0;
    i++;
  } while (i != s);
}
</pre></td></tr>
<tr><td><pre>
_fw:
        .stack _fw = 0
        mul r6, r7, r0
        mov 0x00000000, r2
        mov #_array, r5
.BB.LABEL.1_1: ; bb8
        cmp r7, r2
        bge9 .BB.LABEL.1_3
.BB.LABEL.1_2: ; bb
        st.w r0, 0x00000000[r5]
        add 0x00000004, r5
        add 0x00000001, r2
        br9 .BB.LABEL.1_1
.BB.LABEL.1_3: ; return
        jmp [r31]
</pre></td><td><pre>
_fb2:
        .stack _fb2 = 0
        mul r6, r7, r0
        mov #_array, r2
.BB.LABEL.3_1: ; bb
        st.w r0, 0x00000000[r2]
        add 0x00000004, r2
        loop r7, .BB.LABEL.3_1
.BB.LABEL.3_2: ; return
        jmp [r31]
</pre></td></tr>
<tr><td>28 bytes</td><td>22 bytes</td></tr>
</table>

4.1.11 Direct Assignment from a Conditional Expression when 0 or 1 is Assigned According to the Result of a Conditional Comparison

When 0 or 1 is assigned in the conditional branch destinations of an if statement, directly assign the result of the conditional expression.

<table>
<tr><th>Assignment at Branches</th><th>Assignment from the Conditional Expression</th></tr>
<tr><td><pre>
extern int flag, s;
int fw(void) {
  if(s > 100){
    flag = 1;
  }
  else{
    flag = 0;
  }
  return 0;
}
</pre></td><td><pre>
extern int flag, s;
int fb(void) {
  flag = (s > 100);
  return 0;
}
</pre></td></tr>
<tr><td><pre>
_fw:
        .stack _fw = 0
        movhi HIGHW1(#_s), r0, r2
        ld.w LOWW(#_s)[r2], r2
        addi 0xFFFFFF9B, r2, r0
        mov 0x00000001, r2
        bge9 .BB.LABEL.1_2
.BB.LABEL.1_1: ; if_else_bb
        mov 0x00000000, r2
.BB.LABEL.1_2: ; if_break_bb
        movhi HIGHW1(#_flag), r0, r5
        st.w r2, LOWW(#_flag)[r5]
        mov 0x00000000, r10
        jmp [r31]
</pre></td><td><pre>
_fb:
        .stack _fb = 0
        movhi HIGHW1(#_s), r0, r2
        ld.w LOWW(#_s)[r2], r2
        addi 0xFFFFFF9C, r2, r0
        setf 0x0000000F, r2
        movhi HIGHW1(#_flag), r0, r5
        st.w r2, LOWW(#_flag)[r5]
        mov 0x00000000, r10
        jmp [r31]
</pre></td></tr>
<tr><td>30 bytes</td><td>28 bytes</td></tr>
</table>

4.1.12 Replacing Division by Powers of Two with Shift Operations

If it is cleared that the divisor in division is a power of two and the dividend is a positive value, replace the division with a shift operation.

Note: Changing the type of the dividend to unsigned int may similarly increase efficiency.

Division by a Power of Two	Shift Operation

`extern int s;`
`void fw(void) {`
`  s = s / 2;`
`}`
	`extern int s;`
	`void fb(void) {`
	`  s = s >> 1;`
	`}`

`_fw:`
`        .stack _fw = 0`
`        movhi HIGHW1(#_s), r0, r2`
`        ld.w LOWW(#_s)[r2], r5`
`        mov 0x00000002, r6`
`        divh r6, r5`
`        st.w r5, LOWW(#_s)[r2]`
`        jmp [r31]`
	`_fb:`
	`        .stack _fb = 0`
	`        movhi HIGHW1(#_s), r0, r2`
	`        ld.w LOWW(#_s)[r2], r5`
	`        sar 0x00000001, r5`
	`        st.w r5, LOWW(#_s)[r2]`
	`        jmp [r31]`

18 bytes	16 bytes

4.1.13 Giving a Sign to External and Static Variables

If an expression includes an external or static variable which make relative reference to ep and the type of which can be signed or unsigned, declaring the variable as signed may reduce the code size.

<table>
<tr><th>Unsigned</th><th>Signed</th></tr>
<tr><td><pre>/* -Osmap */
unsigned char cw[256];
int fw() {
  return cw[0] + cw[16];
}</pre></td><td><pre>/* -Osmap */
signed char cb[256];
int fb() {
  return cb[0] + cb[16];
}</pre></td></tr>
<tr><td><pre>_fw:
        .stack _fw = 4
        prepare 0x00000800, 0x00000000
        mov #_cw+0x00000010, r30
        sld.bu 0x00000000[r30], r10
        ld.bu 0xFFFFFFF0[r30], r2
        add r2, r10
        dispose 0x00000000, 0x00000800, [r31]</pre></td><td><pre>_fb:
        .stack _fb = 4
        prepare 0x00000800, 0x00000000
        mov #_cb, r30
        sld.b 0x00000010[r30], r10
        sld.b 0x00000000[r30], r2
        add r2, r10
        dispose 0x00000000, 0x00000800, [r31]</pre></td></tr>
<tr><td>22 bytes</td><td>20 bytes</td></tr>
</table>

4.1.14 Using Structures

In cases of repeated reference to related data in the same function, using a structure makes it easy for the compiler to generate code using relative access and this can be expected to improve efficiency. Passing the data as an argument also improves the efficiency. Since relative access places a limit on the range of access, it is effective when the data which are frequently accessed are placed at the top of the structure.

Note: Similar efficiency may be obtained with the –Omap or –Osmap option.

<table>
<tr><th>Without a Structure</th><th>With a Structure</th></tr>
<tr><td><pre>int a, b, c;
void fw() {
    a = 1;
    b = 2;
    c = 3;
}</pre></td><td><pre>struct s{
    int a;
    int b;
    int c;
} st;
void fb() {
    register struct s *p=&st;
    p->a = 1;
    p->b = 2;
    p->c = 3;
}</pre></td></tr>
<tr><td><pre>_fw:
        .stack _fw = 0
        movhi HIGHW1(#_a), r0, r2
        mov 0x00000001, r5
        st.w r5, LOWW(#_a)[r2]
        movhi HIGHW1(#_b), r0, r2
        mov 0x00000002, r5
        st.w r5, LOWW(#_b)[r2]
        movhi HIGHW1(#_c), r0, r2
        mov 0x00000003, r5
        st.w r5, LOWW(#_c)[r2]
        jmp [r31]</pre></td><td><pre>_fb:
        .stack _fb = 0
        mov #_st, r2
        mov 0x00000001, r5
        st.w r5, 0x00000000[r2]
        mov 0x00000002, r5
        st.w r5, 0x00000004[r2]
        mov 0x00000003, r5
        st.w r5, 0x00000008[r2]
        jmp [r31]</pre></td></tr>
<tr><td>32 bytes</td><td>26 bytes</td></tr>
</table>

4.1.15 Changing Bit Fields with Two or More Bits to the char Type

When a bit field has two or more bits, change the bit field to the char type.

Note: This will increase the amount of RAM in use.

Bit Fields	char

`struct {`
`  unsigned char b0:1;`
`  unsigned char b1:2;`
`} dw;`
`unsigned char dummy;`
`int fw(void) {`
`  if(dw.b1){`
`    dummy++;`
`  }`
`  return 0;`
`}`
	`struct {`
	`  unsigned char b0:1;`
	`  `**`unsigned char b1;`**
	`} db;`
	`unsigned char dummy;`
	`int fb(void) {`
	`  if(db.b1){`
	`    dummy++;`
	`  }`
	`  return 0;`
	`}`

`_fw:`
`        .stack _fw = 0`
`        movhi HIGHW1(#_dw), r0, r2`
`        ld.bu LOWW(#_dw)[r2], r2`
`        andi 0x00000006, r2, r0`
`        bnz9 .BB.LABEL.1_2`
`.BB.LABEL.1_1:  ; if_break_bb`
`        mov 0x00000000, r10`
`        jmp [r31]`
`.BB.LABEL.1_2:  ; if_then_bb`
`        movhi HIGHW1(#_dummy), r0, r2`
`        ld.b LOWW(#_dummy)[r2], r5`
`        add 0x00000001, r5`
`        st.b r5, LOWW(#_dummy)[r2]`
`        br9 .BB.LABEL.1_1`
	`_fb:`
	`        .stack _fb = 0`
	`        movhi HIGHW1(#_db+0x00000001), r0,`
	`r2`
	`        ld.bu LOWW(#_db+0x00000001)[r2], r2`
	`        cmp 0x00000000, r2`
	`        bnz9 .BB.LABEL.2_2`
	`.BB.LABEL.2_1:  ; if_break_bb`
	`        mov 0x00000000, r10`
	`        jmp [r31]`
	`.BB.LABEL.2_2:  ; if_then_bb`
	`        movhi HIGHW1(#_dummy), r0, r2`
	`        ld.b LOWW(#_dummy)[r2], r5`
	`        add 0x00000001, r5`
	`        st.b r5, LOWW(#_dummy)[r2]`
	`        br9 .BB.LABEL.2_1`

34 bytes	32 bytes

4.1.16 Declaring Global Variables as Automatic Variables Where Possible

If a variable can be used as an automatic variable, declare it as such, rather than as an external variable. Since the value of an external variable may be changed by calling a function or operations affecting a pointer, optimization will be less efficient if an external variable is used.

External Variable	Automatic Variable

`int tmp;`
`void fw(int* a, int* b) {`
`    tmp = *a;`
`    *a = *b;`
`    *b = tmp;`
`}`
	`void fb(int* a, int* b) {`
	`    int tmp;`
	`    tmp = *a;`
	`    *a = *b;`
	`    *b = tmp;`
	`}`

`_fw:`
`        .stack _fw = 0`
`        ld.w 0x00000000[r6], r2`
`        movhi HIGHW1(#_tmp), r0, r5`
`        st.w r2, LOWW(#_tmp)[r5]`
`        ld.w 0x00000000[r7], r2`
`        st.w r2, 0x00000000[r6]`
`        ld.w LOWW(#_tmp)[r5], r2`
`        st.w r2, 0x00000000[r7]`
`        jmp [r31]`
	`_fb:`
	`        .stack _fb = 0`
	`        ld.w 0x00000000[r7], r2`
	`        ld.w 0x00000000[r6], r5`
	`        st.w r2, 0x00000000[r6]`
	`        st.w r5, 0x00000000[r7]`
	`        jmp [r31]`

30 bytes	18 bytes

4.1.17 Assigning Small Absolute Values when Referring to Constants

When a constant value is used for a macro definition or an enumerated type, assigning a small absolute value may reduce the code size may be reduced.

<table>
<tr><th>Larger Value</th><th>Smaller Value</th></tr>
<tr><td><pre>#define   CODEW (567)
extern int data;
void fw() {
    data= CODEW;
}</pre></td><td><pre>#define   CODEB (15)
extern int data;
void fb() {
    data= CODEB;
}</pre></td></tr>
<tr><td><pre>_fw:
        .stack _fw = 0
        movhi HIGHW1(#_data), r0, r2
        movea 0x00000237, r0, r5
        st.w r5, LOWW(#_data)[r2]
        jmp [r31]</pre></td><td><pre>_fb:
        .stack _fb = 0
        movhi HIGHW1(#_data), r0, r2
        mov 0x0000000F, r5
        st.w r5, LOWW(#_data)[r2]
        jmp [r31]</pre></td></tr>
<tr><td>14 bytes</td><td>12 bytes</td></tr>
</table>

4.1.18 Modularizing Functions

Defining calling and called functions in the same file may reduce the code size.

<table>
<tr><th>Separate Files</th><th>Same File</th></tr>
<tr><td><pre>extern void fwsub();
void fw() {
  fwsub();
}</pre></td><td><pre>void fbsub() {
...
}
void fb() {
  fbsub();
}</pre></td></tr>
<tr><td><pre>_fw:
        .stack _fw = 0
        jr32 _fwsub</pre></td><td><pre>_fb:
        .stack _fb = 0
        br9 _fbsub</pre></td></tr>
<tr><td>4 bytes</td><td>2 bytes</td></tr>
</table>

4.1.19 Using Tables Instead of switch Statements

Using tables instead of switch statements may reduce the code size.

<table>
<tr><th>switch Statement</th><th>Equivalent Table-Based Code</th></tr>
<tr><td><pre>int fw(int i) {
  char ch;
  switch (i) {
  case 0:
    ch = 'a'; break;
  case 1:
    ch = 'x'; break;
  case 2:
    ch = 'b'; break;
  default:
    ch = 0; break;
  }
  return (ch);
}</pre></td><td><pre>const char chbuf[] = { 'a', 'x', 'b' };
int fb(int i) {
  if ((unsigned int)i < 3) {
    return (chbuf[i]);
  }
  return (0);
}</pre></td></tr>
<tr><td><pre>_fw:
        .stack _fw = 0
        cmp 0x00000002, r6
        bh9 .BB.LABEL.1_3
.BB.LABEL.1_1: ; entry
        shl 0x00000001, r6
        jmp #.SWITCH.LABEL.1_7[r6]
.SWITCH.LABEL.1_7:
        br9 .BB.LABEL.1_2
        br9 .BB.LABEL.1_4
        br9 .BB.LABEL.1_5
.SWITCH.LABEL.1_7.END:
.BB.LABEL.1_2: ; entry.bb5_crit_edge
        movea 0x00000061, r0, r10
        br9 .BB.LABEL.1_6
.BB.LABEL.1_3: ; bb4
        mov 0x00000000, r10
        br9 .BB.LABEL.1_6
.BB.LABEL.1_4: ; bb2
        movea 0x00000078, r0, r10
        br9 .BB.LABEL.1_6
.BB.LABEL.1_5: ; bb3
        movea 0x00000062, r0, r10
.BB.LABEL.1_6: ; bb5
        sxb r10
        jmp [r31]</pre></td><td><pre>_fb:
        .stack _fb = 0
        cmp 0x00000002, r6
        bh9 .BB.LABEL.2_2
.BB.LABEL.2_1: ; if_then_bb
        mov #_chbuf, r2
        add r6, r2
        ld.b 0x00000000[r2], r10
        jmp [r31]
.BB.LABEL.2_2: ; bb9
        mov 0x00000000, r10
        jmp [r31]
_chbuf:
        .db 0x61
        .db 0x78
        .db 0x62</pre></td></tr>
<tr><td>42 bytes</td><td>25 bytes</td></tr>
</table>

4.1.20 Reviewing the Specifications of Interrupt Functions

Specifying -Xreg_mode=22 and -Xreserve_r2 decreases the number of registers to be saved and restored in interrupt functions and thus may reduce the code size.

<table>
<tr><th>Specifying -Xreg_mode=32</th><th>Specifying -Xreg_mode=22 and -Xreserve_r2</th></tr>
<tr><td><pre>/* -Xreg_mode=32 */
#pragma interrupt fw
void fw() {
  sub();
}</pre></td><td><pre>/* -Xreg_mode=22, -Xreserve_r2 */
#pragma interrupt fb (enable=false,
callt=false, fpu=false)
void fb() {
  sub();
}</pre></td></tr>
<tr><td><pre>_fw:
        .stack _fw = 88
        movea 0xFFFFFFAC, r3, r3
        st.w r1, 0x00000010[r3]
        st.w r2, 0x00000014[r3]
        st.w r5, 0x00000018[r3]
        st23.dw r6, 0x0000001C[r3]
        st23.dw r8, 0x00000024[r3]
        st23.dw r10, 0x0000002C[r3]
        st23.dw r12, 0x00000034[r3]
        st23.dw r14, 0x0000003C[r3]
        st23.dw r16, 0x00000044[r3]
        st23.dw r18, 0x0000004C[r3]
        stsr 0x00000010, r8, 0x00000000
        stsr 0x00000011, r9, 0x00000000
        st23.dw r8, 0x00000000[r3]
        stsr 0x00000007, r8, 0x00000000
        stsr 6, r9
        st23.dw r8, 0x00000008[r3]
        prepare 0x00000001, 0x00000000
        jarl _sub, r31
        dispose 0x00000000, 0x00000001
        ld23.dw 0x00000008[r3], r8
        ldsr r9, 6
        ldsr r8, 0x00000007, 0x00000000
        ld23.dw 0x00000000[r3], r8
        ldsr r9, 0x00000011, 0x00000000
        ldsr r8, 0x00000010, 0x00000000
        ld23.dw 0x0000004C[r3], r18
        ld23.dw 0x00000044[r3], r16
        ld23.dw 0x0000003C[r3], r14
        ld23.dw 0x00000034[r3], r12
        ld23.dw 0x0000002C[r3], r10
        ld23.dw 0x00000024[r3], r8
        ld23.dw 0x0000001C[r3], r6
        ld.w 0x00000018[r3], r5
        ld.w 0x00000014[r3], r2
        ld.w 0x00000010[r3], r1
        movea 0x00000054, r3, r3
        eiret</pre></td><td><pre>_fb:
        .stack _fb20 = 64
        movea 0xFFFFFFC4, r3, r3
        st.w r1, 0x00000010[r3]
        st.w r5, 0x00000014[r3]
        st23.dw r6, 0x00000018[r3]
        st23.dw r8, 0x00000020[r3]
        st23.dw r10, 0x00000028[r3]
        st23.dw r12, 0x00000030[r3]
        st.w r14, 0x00000038[r3]
        stsr 0x00000010, r8, 0x00000000
        stsr 0x00000011, r9, 0x00000000
        st23.dw r8, 0x00000000[r3]
        stsr 0x00000007, r8, 0x00000000
        stsr 6, r9
        st23.dw r8, 0x00000008[r3]
        prepare 0x00000001, 0x00000000
        jarl _sub, r31
        dispose 0x00000000, 0x00000001
        ld23.dw 0x00000008[r3], r8
        ldsr r9, 6
        ldsr r8, 0x00000007, 0x00000000
        ld23.dw 0x00000000[r3], r8
        ldsr r9, 0x00000011, 0x00000000
        ldsr r8, 0x00000010, 0x00000000
        ld.w 0x00000038[r3], r14
        ld23.dw 0x00000030[r3], r12
        ld23.dw 0x00000028[r3], r10
        ld23.dw 0x00000020[r3], r8
        ld23.dw 0x00000018[r3], r6
        ld.w 0x00000014[r3], r5
        ld.w 0x00000010[r3], r1
        movea 0x0000003C, r3, r3
        eiret</pre></td></tr>
<tr><td>188 bytes</td><td>152 bytes</td></tr>
</table>

In addition, reviewing the interrupt specifications of #pragma interrupt may enable decreasing the number of registers to be saved and restored and thus reduce the code size.

#pragma Interrupt without Options	#pragma Interrupt with Suitable Options

`#pragma interrupt fw`
`void fw() {`
`  sub();`
`}`
	`#pragma interrupt fb` **(enable=false, callt=false, fpu=false)**
	`void fb() {`
	`  sub();`
	`}`

`_fw:`
`.stack _fw = 88`
`movea 0xFFFFFFAC, r3, r3`
`st.w r1, 0x00000010[r3]`
`st.w r2, 0x00000014[r3]`
`st.w r5, 0x00000018[r3]`
`st23.dw r6, 0x0000001C[r3]`
`st23.dw r8, 0x00000024[r3]`
`st23.dw r10, 0x0000002C[r3]`
`st23.dw r12, 0x00000034[r3]`
`st23.dw r14, 0x0000003C[r3]`
`st23.dw r16, 0x00000044[r3]`
`st23.dw r18, 0x0000004C[r3]`
`stsr 0x00000010, r8, 0x00000000`
`stsr 0x00000011, r9, 0x00000000`
`st23.dw r8, 0x00000000[r3]`
`stsr 0x00000007, r8, 0x00000000`
`stsr 6, r9`
`st23.dw r8, 0x00000008[r3]`
`prepare 0x00000001, 0x00000000`
`jarl _sub, r31`
`dispose 0x00000000, 0x00000001`
`ld23.dw 0x00000008[r3], r8`
`ldsr r9, 6`
`ldsr r8, 0x00000007, 0x00000000`
`ld23.dw 0x00000000[r3], r8`
`ldsr r9, 0x00000011, 0x00000000`
`ldsr r8, 0x00000010, 0x00000000`
`ld23.dw 0x0000004C[r3], r18`
`ld23.dw 0x00000044[r3], r16`
`ld23.dw 0x0000003C[r3], r14`
`ld23.dw 0x00000034[r3], r12`
`ld23.dw 0x0000002C[r3], r10`
`ld23.dw 0x00000024[r3], r8`
`ld23.dw 0x0000001C[r3], r6`
`ld.w 0x00000018[r3], r5`
`ld.w 0x00000014[r3], r2`
`ld.w 0x00000010[r3], r1`
`movea 0x00000054, r3, r3`
`eiret`
	`_fb:`
	`.stack _fb20 = 72`
	`movea 0xFFFFFFBC, r3, r3`
	`st.w r1, 0x00000000[r3]`
	`st.w r2, 0x00000004[r3]`
	`st.w r5, 0x00000008[r3]`
	`st23.dw r6, 0x0000000C[r3]`
	`st23.dw r8, 0x00000014[r3]`
	`st23.dw r10, 0x0000001C[r3]`
	`st23.dw r12, 0x00000024[r3]`
	`st23.dw r14, 0x0000002C[r3]`
	`st23.dw r16, 0x00000034[r3]`
	`st23.dw r18, 0x0000003C[r3]`
	`prepare 0x00000001, 0x00000000`
	`jarl _sub, r31`
	`dispose 0x00000000, 0x00000001`
	`ld23.dw 0x0000003C[r3], r18`
	`ld23.dw 0x00000034[r3], r16`
	`ld23.dw 0x0000002C[r3], r14`
	`ld23.dw 0x00000024[r3], r12`
	`ld23.dw 0x0000001C[r3], r10`
	`ld23.dw 0x00000014[r3], r8`
	`ld23.dw 0x0000000C[r3], r6`
	`ld.w 0x00000008[r3], r5`
	`ld.w 0x00000004[r3], r2`
	`ld.w 0x00000000[r3], r1`
	`movea 0x00000044, r3, r3`
	`eiret`

188 bytes	132 bytes

4.1.21 Placing Functions and Variables in Libraries

For processing used by multiple projects, **finely separating the functions or variables in libraries** means that only the functions and variables required for each project are linked, which reduces the code size.

4.2 Speeding Execution up

The examples below are with the –Ospeed option for CC-RH V1.04.00 specified.

4.2.1 Expanding Loops

Reducing the number of loops decreases the overhead of branch instructions.

Note: Although the compiler also performs similar optimization, the effect depends on the statements in the source code and on whether or not the –Ounroll option is specified.

<table>
<tr><th>Loop for a Single Case</th><th>Separate Loops for Two Cases</th></tr>
<tr><td><pre>extern int array[];
void fw(void) {
  int i;
  int *p;
  p = array;
  for(i = 16; i > 0; i--){
    *p++ = 0;
  }
}</pre></td><td><pre>extern int array[];
void fb(void) {
  int i;
  int *p;
  p = array;
  for(i = 16 >> 2; i > 0; i--){ /* N/4 */
    *p++ = 0;
    *p++ = 0;
    *p++ = 0;
    *p++ = 0;
  }
  for(i = 16 & 3; i > 0; i--){ /* N mod 4 */
    *p++ = 0;
  }
}</pre></td></tr>
<tr><td><pre>_fw:
        .stack _fw = 0
        mov #_array, r2
        mov 0x00000004, r5
.BB.LABEL.1_1: ; bb.split.clone
        st.w r0, 0x00000000[r2]
        st.w r0, 0x00000004[r2]
        st.w r0, 0x00000008[r2]
        st.w r0, 0x0000000C[r2]
        movea 0x00000010, r2, r2
        loop r5, .BB.LABEL.1_1
.BB.LABEL.1_2: ; return
        jmp [r31]</pre></td><td><pre>_fb:
        .stack _fb = 0
        mov #_array, r2
        st.w r0, 0x00000000[r2]
        st.w r0, 0x00000004[r2]
        st.w r0, 0x00000008[r2]
        st.w r0, 0x0000000C[r2]
        st.w r0, 0x00000010[r2]
        st.w r0, 0x00000014[r2]
        st.w r0, 0x00000018[r2]
        st.w r0, 0x0000001C[r2]
        st.w r0, 0x00000020[r2]
        st.w r0, 0x00000024[r2]
        st.w r0, 0x00000028[r2]
        st.w r0, 0x0000002C[r2]
        st.w r0, 0x00000030[r2]
        st.w r0, 0x00000034[r2]
        st.w r0, 0x00000038[r2]
        st.w r0, 0x0000003C[r2]
        jmp [r31]</pre></td></tr>
<tr><td>34 bytes and 32 clock cycles</td><td>72 bytes and 29 clock cycles</td></tr>
</table>

4.2.2 Using Constants as Divisors

In terms of optimization, division by constants extends to operations other than division. Thus, use division of constants where possible.

Note: Smaller differences between the number of valid bits in the divisor and dividend speed up the divq instruction of the RH850.

Division by a Variable	Division by a Constant

`int fw(int x, int y) {`
`  return x/y;`
`}`
	`int fb(int x) {`
	`  return x/3;`
	`}`

`_fw:`
`        .stack _fw = 0`
`        mov r6, r10`
`        divq r7, r10, r0`
`        jmp [r31]`
	`_fb:`
	`        .stack _fb = 0`
	`        mov 0x55555556, r2`
	`        mul r2, r6, r10`
	`        mov r10, r2`
	`        shr 0x0000001F, r2`
	`        add r2, r10`
	`        jmp [r31]`

8 bytes and 5 to 20 clock cycles	18 bytes and 4 to 5 clock cycles

4.2.3 Changing the Type of the Loop Control Variable

Changing a loop control variable to a signed 4-byte integer type (signed int or signed long) raises the likelihood of optimization in the form of loop unrolling, which increases the speed of execution.

Loop Control Variable Not Changed

```
extern int ub;
extern char a[16];
void fw() {
    unsigned char i;
    for(i=0;i<ub;i++) {
        a[i]=0;
    }
}
```

```
_fw:
        .stack _fw = 0
        movhi HIGHW1(#_ub), r0, r2
        ld.w LOWW(#_ub)[r2], r2
        cmp 0x00000000, r2
        ble9 .BB.LABEL.1_3
.BB.LABEL.1_1:  ; entry.bb_crit_edge
        mov 0x00000000, r5
        mov #_a, r6
.BB.LABEL.1_2:  ; bb
        andi 0x000000FF, r5, r7
        add 0x00000001, r5
        add r6, r7
        st.b r0, 0x00000000[r7]
        andi 0x000000FF, r5, r7
        cmp r2, r7
        blt9 .BB.LABEL.1_2
.BB.LABEL.1_3:  ; return
        jmp [r31]
```

42 bytes and 47 clock cycles

Signed Loop Control Variable

```
extern int ub;
extern char a[16];
void fb() {
    int i;
    for(i=0;i<ub;i++) {
        a[i]=0;
    }
}
```

```
_fb:
        .stack _fb24 = 0
        movhi HIGHW1(#_ub), r0, r2
        ld.w LOWW(#_ub)[r2], r2
        cmp 0x00000000, r2
        ble9 .BB.LABEL.1_7
.BB.LABEL.1_1:   ; bb.nph
        cmp 0x00000003, r2
        mov 0x00000000, r5
        bnh9 .BB.LABEL.1_5
.BB.LABEL.1_2:   ; preheader.ul
        mov 0xFFFFFFFC, r5
        and r2, r5
        mov r2, r6
        sar 0x00000002, r6
        mov #_a, r7
.BB.LABEL.1_3:   ; bb.split.clone
        st.b r0, 0x00000000[r7]
        st.b r0, 0x00000001[r7]
        st.b r0, 0x00000002[r7]
        st.b r0, 0x00000003[r7]
        add 0x00000004, r7
        loop r6, .BB.LABEL.1_3
.BB.LABEL.1_4:   ; exit.ul
        cmp r2, r5
        bz9 .BB.LABEL.1_7
.BB.LABEL.1_5:   ; bb.split.preheader
        sub r5, r2
        mov #_a, r6
        add r6, r5
.BB.LABEL.1_6:   ; bb.split
        st.b r0, 0x00000000[r5]
        add 0x00000001, r5
        loop r2, .BB.LABEL.1_6
.BB.LABEL.1_7:   ; return
        jmp [r31]
```

80 bytes and 41 clock cycles

4.3 Reducing Amounts of Data

Reducing amounts of data is not only effective in reducing the size of the data area (*.data.R, etc.) for the initial values generated in mapping from ROM to RAM, but also shortens the times taken for processing to transfer the initial values and for clearing to zero.

4.3.1 Aligning Data

Addresses to be accessed must usually be aligned to specific boundaries for the memory access instructions of the RH850. The compiler thus aligns (inserting padding areas without changing the order) and locates variables.

As alignment conditions, the char type, short type, and int type are aligned with 1-byte, 2-byte, and 4-byte boundaries, respectively. Declare the variables in order from longer to shorter.

Mixed Lengths	Longer to Shorter

`char aw;`
`int  bw;`
`char cw;`
`int  dw;`
`char ew;`
	`int  bb;`
	`int  db;`
	`char ab;`
	`char cb;`
	`char eb;`

`_aw:`
`        .ds (1)`
`        .align 4`
`_bw:`
`        .ds (4)`
`_cw:`
`        .ds (1)`
`        .align 4`
`_dw:`
`        .ds (4)`
`_ew:`
`        .ds (1)`
	`_bb:`
	`        .ds (4)`
	`        .align 4`
	`_db:`
	`        .ds (4)`
	`_ab:`
	`        .ds (1)`
	`_cb:`
	`        .ds (1)`
	`_eb:`
	`        .ds (1)`

17 bytes	11 bytes

4.3.2 Aligning Structure Members

As in the case of variables, structure members generally must also be aligned when allocated. Allocating members in order from longer to shorter decreases the amount of memory taken up by variables.

Note: When the structure packing function is used, members are not aligned but padded for allocation. This decreases the amount of space taken up by the member variables but increases the size of the code required for access to the member variables.

Mixed Lengths	Longer to Shorter

`struct {`
`char aw;`
`int  bw;`
`char cw;`
`int  dw;`
`char ew;`
`} fw;`
	`struct {`
	`int  bb;`
	`int  db;`
	`char ab;`
	`char cb;`
	`char eb;`
	`} fb;`

`_fw:` `        .ds (20)`	`_fb:` `        .ds (12)`
20 bytes	12 bytes

4.3.3 Changing Variables with Initial Values to const Variables or Moving them to the Section with the bss Attribute

When external and static variables have initial values, the initial values are copied from ROM to RAM when the program is started. Accordingly, an area of memory twice the original size is occupied in memory as a whole and copying takes time.

Variables with values that do not change must be changed to const variables and allocated to ROM.

The bss attribute section is initialized to 0 in the startup routine. **When external and static variables with the initial value 0 are allocated to the bss attribute section, the initial values need not be specified,** reducing the amount of data for the initial values of variables.

4.3.4 Reducing Holes Due to Alignment

The first addresses of each data section must be aligned with 4-byte boundaries. If variable definitions are separated in multiple files, holes due to alignment will be generated between variables from different files at the time of linkage. Consolidating definitions in a single file reduces the incidence of such holes.

Two Files	One File
==w1.c== int v; char w; ==w2.c== char y; char z;	**int** v; **char** w; **char** y; **char** z;
10 bytes	7 bytes

Separating sections for the allocation of variables by the #pragma section directive even in a single file also leads to holes due to alignment. Consolidating multiple sections as one also reduces the incidence of holes due to alignment.

Separated Sections	Consolidated Sections
#pragma section fw1 char fw; #pragma section fw2 char fw2;	#pragma section fb **char** fb1; **char** fb2;
5 bytes	4 bytes

5. Revision History

		Description	
Rev.	**Date**	**Page**	**Summary**
1.00	August 05, 2016	-	First edition

SALES OFFICES Renesas Electronics Corporation http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics America Inc.
2801 Scott Boulevard Santa Clara, CA 95050-2549, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
9251 Yonge Street, Suite 8309 Richmond Hill, Ontario Canada L4C 9T3
Tel: +1-905-237-2004

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-585-100, Fax: +44-1628-585-900

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-6503-0, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
Room 1709, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100191, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 301, Tower A, Central Towers, 555 Langao Road, Putuo District, Shanghai, P. R. China 200333
Tel: +86-21-2226-0888, Fax: +86-21-2226-0999

Renesas Electronics Hong Kong Limited
Unit 1601-1611, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2265-6688, Fax: +852 2886-9022

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei 10543, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre, Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300

Renesas Electronics Malaysia Sdn.Bhd.
Unit 1207, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics India Pvt. Ltd.
No.777C, 100 Feet Road, HAL II Stage, Indiranagar, Bangalore, India
Tel: +91-80-67208700, Fax: +91-80-67208777

Renesas Electronics Korea Co., Ltd.
12F., 234 Teheran-ro, Gangnam-Gu, Seoul, 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

Colophon 5.0