

RH850 用 C コンパイラ CC-RH V2 アプリケーションガイド プログラミング・テクニック編

目次

第 1 章 はじめに	3
1.1 概要	3
第 2 章 オプション	4
2.1 コンパイル・オプション	4
2.2 アセンブル・オプション	6
2.3 リンク・オプション	6
第 3 章 拡張言語	7
3.1 #pragma 指令	7
第 4 章 コーディング方法	8
4.1 コードサイズの削減	8
4.1.1 switch 文の代わりに if~else 文を使う	8
4.1.2 関数の出口を 1 箇所にまとめる	9
4.1.3 複数箇所にある外部変数アクセスをテンポラリ変数により 1 箇所にまとめる	10
4.1.4 条件分岐先の同一文は分岐前に移動する	11
4.1.5 複雑な if 文を論理的に等しいものに置き換える	12
4.1.6 short, char 型の変数は int 型にする	13
4.1.7 switch 文の共通な case の処理をまとめる	14
4.1.8 インライン展開する関数は static 関数にする	15
4.1.9 for ループを do while ループに置き換える	16
4.1.10 条件比較の結果によって 0 または 1 を代入する場合は条件式を直接代入する	17
4.1.11 2 のべき乗の除算はシフト演算に置き換える	17
4.1.12 外部変数および静的変数の符号を変える	18
4.1.13 構造体を活用する	18
4.1.14 2 ビット以上のビットフィールドは char 型に変更する	19
4.1.15 自動変数として使用できる大域変数は自動変数として定義する	19
4.1.16 定数参照の際には絶対値の小さい値を割り当てる	20
4.1.17 関数をモジュール化する	20
4.1.18 switch 文の代わりにテーブルを活用する	21
4.1.19 割り込み関数の仕様を見直す	22
4.1.20 ライブラリ化する	23
4.2 実行速度の改善	24
4.2.1 ループ展開する	24
4.2.2 除数は定数を使用する	25
4.2.3 ループ制御変数の型を変更する	26

4.3	データサイズの削減.....	27
4.3.1	データを整列する.....	27
4.3.2	構造体メンバを整列する.....	27
4.3.3	初期値を持つ変数を const 変数あるいは bss 属性セクションに変更する.....	28
4.3.4	アライン・ホールを減らす.....	28

第 5 章	改定履歴.....	29
--------------	------------------	-----------

第1章 はじめに

本書は、RH850 用コンパイラ CC-RH を使用してプログラムを開発されるユーザーを対象に、コードサイズ・実行速度・データサイズに効果的なプログラミング方法をガイドします。

1.1 概要

コードサイズ・実行速度・データサイズに効果的なプログラミング方法として、以下の3つの項目に分けてそれぞれ説明します。

- オプション
- 拡張言語
- コーディング方法

本アプリケーションノート記載の測定結果、アセンブリ言語展開コードは CC-RH V2.02 を用いて取得しています。

なお今後の CC-RH のリビジョンアップにより、生成コード例等、記載内容が変更される可能性がありますことをご了承ください。

第2章 オプション

本章では、CC-RH のオプション指定によるコードサイズ・実行速度・データサイズへの影響を示します。
なお、効果の程度はソース・プログラムの内容によって変わります。

2.1 コンパイル・オプション

○…改善する ×…劣化する △…状況に依存する —…変化しない

オプション	コード サイズ	実行 速度	データ サイズ	備考
-Xenum_type=auto	△	△	○	型が小さくなると、符号拡張・ゼロ拡張命令を生成する場合があります。そのため、コードサイズ・実行速度は状況に依存します。
-Xvolatile	×	×	—	
-Xuse_fp16=on	×	×	○	professional 版のライセンスが登録されている場合にのみ指定できます。
-Onothing	×	×	×	未使用関数からのみ参照される静的変数が削除されないため、データサイズにも影響があります。
-Odefault	—	—	—	
-Osize	○	△	○	未使用関数からのみ参照される静的変数が削除される場合にデータサイズに効果があります。
-Ospeed	△	○	○	未使用関数からのみ参照される静的変数が削除される場合にデータサイズに効果があります。
-Ounroll=小さい値	○	×	—	
-Ounroll=大きい値	×	○	—	
-Oinline=0	○	×	—	
-Oinline=1	—	—	—	
-Oinline=2	×	○	—	
-Oinline=3	○	○	—	
-Oinline_size=小さい値	○	×	—	
-Oinline_size=大きい値	×	○	—	
-Oinline_init	△	○	○	
-Odelete_static_func=on	○	—	○	未使用関数からのみ参照される静的変数が削除される場合にデータサイズに効果があります。
-Opipeline=on	—	○	—	
-Otail_call=on	○	○	—	
-Omap	○	○	—	
-Osmap	○	○	—	
-Xintermodule	○	○	—	
-Xinline_strcpy	×	○	—	
-Xmerge_string	—	—	○	
-Xalias=ansi	○	○	—	
-Xmerge_files	○	○	—	
-Xwhole_program	○	○	—	
-library=intrinsic	×	○	—	
-goptimize	○	—	○	
-Xpack=1,2	×	×	○	
-Xbit_order	—	—	—	

-Xswitch=ifelse	△	△	—	switch ラベルが少ない場合は効率向上する傾向があります。
-Xswitch=binary	×	△	—	switch ラベルが多い場合は効率向上する傾向があります。
-Xswitch=table	△	△	—	switch ラベルが少ない場合は効率向上する傾向があります。
-Xreg_mode=22,common	△	△	—	レジスタを多く消費する関数ではコードサイズと実行速度とも劣化する場合があります。
-Xreserve_r2	△	△	—	レジスタを多く消費する関数ではコードサイズと実行速度とも劣化する場合があります。
-r4=none	△	△	—	gp 相対セクションを利用できなくなるため、コードサイズと実行速度が劣化する場合があります。
-Xep=fix	×	×	—	レジスタを多く消費する関数ではコードサイズと実行速度とも劣化する場合があります。 -Omap オプションを使用する場合は -Xep=fix オプションも同時指定する必要があります。
-Xfloat=soft	×	×	—	
-Xfxu=on	×	×	—	
-Xcall_jump=32	×	×	—	
-Xfar_jump	×	×	—	
-Xdiv	—	×	—	本オプションを省略すると、除算に対して divq, divqu 命令を生成します。divq, divqu 命令は高速ですが、実行サイクル数がオペランドの値により変わります。
-Xcheck_div_ov	×	×	—	
-relaxed_math	○	○	—	浮動小数点演算においてプログラムの実行結果が変わる場合があります。
-Xuse_fmaf	○	○	—	浮動小数点演算においてプログラムの実行結果が変わる場合があります。
-use_recipf	○	○	—	浮動小数点演算においてプログラムの実行結果が変わる場合があります。
-approximate	△	○	—	浮動小数点演算においてプログラムの実行結果が変わる場合があります。
-Xunordered_cmpf	×	×	—	
-Xmulti_level	—	—	—	
-Xpatch=dw_access	×	×	—	本オプションを指定しない場合、マイコンの注意事項に該当する可能性があります。
-Xpatch=switch	×	×	—	本オプションを指定しない場合、マイコンの注意事項に該当する可能性があります。
-Xpatch=syncp	×	×	—	本オプションを指定しない場合、マイコンの注意事項に該当する可能性があります。
-Xdbl_size=4	○	○	○	本オプションを指定した場合、プログラムの実行結果が変わる場合があります。
-Xround=zero	—	—	—	本オプションを指定した場合、プログラムの実行結果が変わる場合があります。
-Xalign4	×	○	—	
-Xstack_protector	×	×	—	professional 版のライセンスが登録されている場合にのみ指定できます。
-Xsection	○	○	—	
-store_reg	×	×	—	professional 版のライセンスが登録されている場合にのみ指定できます。
-control_flow_integrity	×	×	—	professional 版のライセンスが登録されている場合にのみ指定できます。

2.2 アセンブル・オプション

○…改善する ×…劣化する △…状況に依存する —…変化しない

オプション	コード サイズ	実行 速度	データ サイズ	備考
-Xasm_far_jump	×	×	—	

2.3 リンク・オプション

○…改善する ×…劣化する △…状況に依存する —…変化しない

オプション	コード サイズ	実行 速度	データ サイズ	備考
-padding	×	—	×	
-overrun_fetch	×	—	×	
-optimize	○	—	○	
-aligned_section	×	○	×	

第3章 拡張言語

本章では、拡張言語の中で#pragma 指令によるコードサイズ・実行速度・データサイズへの影響を示します。

3.1 #pragma 指令

○…改善する ×…劣化する △…状況に依存する —…変化しない

指令	コード サイズ	実行 速度	データ サイズ	備考
#pragma section	○	○	△	コードサイズと実行速度に効果があるのは、次の属性指定を使用した場合です。 r0_disp16, r0_disp23, ep_disp4, ep_disp5, ep_disp7, ep_disp8, ep_disp16, ep_disp23, gp_disp16, gp_disp23, zconst, zconst23 セクションを細かく区切ると、セクション間に隙間ができてデータサイズが増大する場合があります。
#pragma inline	×	○	—	
#pragma noinline	○	×	—	
#pragma interrupt	—	—	—	退避・復帰が不要なレジスタがある場合は、割り込み仕様(enable/fpu/callt 等)を変更するとコードサイズ・実行速度が向上します。
#pragma block_interrupt	×	×	—	
#pragma pack	×	×	○	
#pragma align4	×	○	—	
#pragma stack_protector	×	×	—	professional 版のライセンスが登録されている場合にのみ指定できます。
#pragma register_group	×	×	—	professional 版のライセンスが登録されている場合にのみ指定できます。

第4章 コーディング方法

本章では、ユーザープログラムのコーディング方法によりコードサイズ・実行速度・データサイズを改善する方法を説明します。

4.1 コードサイズの削減

変更前後の例は、-Osize オプション指定時のものです。

4.1.1 switch文の代わりにif~else文を使う

switch 文の分岐コードはテーブルジャンプを使用した2段階分岐となり、コードサイズが増大する要因になります。また case ラベルにない値に対する分岐コードを含みます。

case ラベルの数が少ない場合に switch 文を if 文に書き換えるとコードサイズが小さくなる可能性があります。5 個未満が目安です。

注意: -Xswitch=ifelse オプションで同様の効果を得られる場合があります。

注意: G3M 以外のコアではコードサイズが増大する可能性があります。

変更前	変更後
<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>
<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: ; switch_clause_bb jarl _sub0, r31 br9 .BB.LABEL.1_6 .BB.LABEL.1_3: ; switch_clause_bb2 jarl _sub1, r31 br9 .BB.LABEL.1_6 .BB.LABEL.1_4: ; switch_clause_bb3 jarl _sub2, r31 br9 .BB.LABEL.1_6</pre>	<pre>_fb: .stack_fb = 4 prepare 0x00000001, 0x00000000 cmp 0x00000000, r6 bnz9 .BB.LABEL.1_2 .BB.LABEL.1_1: ; if_then_bb jarl _sub0, r31 br9 .BB.LABEL.1_8 .BB.LABEL.1_2: ; if_else_bb cmp 0x00000001, r6 bnz9 .BB.LABEL.1_4 .BB.LABEL.1_3: ; if_then_bb8 jarl _sub1, r31 br9 .BB.LABEL.1_8 .BB.LABEL.1_4: ; if_else_bb9 cmp 0x00000002, r6 bnz9 .BB.LABEL.1_6 .BB.LABEL.1_5: ; if_then_bb14 jarl _sub2, r31 br9 .BB.LABEL.1_8 .BB.LABEL.1_6: ; if_else_bb15 cmp 0x00000003, r6 bnz9 .BB.LABEL.1_8</pre>

<pre>.BB.LABEL.1_5: ; switch_clause_bb4 jarl _sub3, r31 .BB.LABEL.1_6: ; switch_break_bb mov 0x00000000, r10 dispose 0x00000000, 0x00000001, [r31]</pre>	<pre>.BB.LABEL.1_7: ; if_then_bb20 jarl _sub3, r31 .BB.LABEL.1_8: ; if_break_bb23 mov 0x00000000, r10 dispose 0x00000000, 0x00000001, [r31]</pre>
52バイト	48バイト

4.1.2 関数の出口を1箇所にまとめる

switch 文や if~else 文が関数の出口にある場合、関数出口のコードを複数生成する場合があります。関数末尾に return 文を1つ追加することで関数出口のコードを1つだけ生成して、コードサイズが小さくなる可能性があります。

変更前	変更後
<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>	<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>
<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: ; switch_clause_bb st.w r0, LOWW(#_s)[r2] jmp [r31] .BB.LABEL.1_6: ; switch_clause_bb1 movea 0x00005555, r0, r5 st.w r5, LOWW(#_s)[r2] jmp [r31] .BB.LABEL.1_7: ; switch_clause_bb2 ori 0x0000AAAA, r0, r5 st.w r5, LOWW(#_s)[r2] jmp [r31] .BB.LABEL.1_8: ; switch_clause_bb3 ori 0x0000FFFF, r0, r5 st.w r5, LOWW(#_s)[r2] jmp [r31]</pre>	<pre>_fb: .stack_fb = 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: ; switch_break_bb mov 0x00000000, r10 jmp [r31] .BB.LABEL.1_5: ; switch_clause_bb st.w r0, LOWW(#_s)[r2] br9 .BB.LABEL.1_4 .BB.LABEL.1_6: ; switch_clause_bb1 movea 0x00005555, r0, r5 br9 .BB.LABEL.1_8 .BB.LABEL.1_7: ; switch_clause_bb2 ori 0x0000AAAA, r0, r5 .BB.LABEL.1_8: ; switch_clause_bb2 st.w r5, LOWW(#_s)[r2] br9 .BB.LABEL.1_4 .BB.LABEL.1_9: ; switch_clause_bb3 ori 0x0000FFFF, r0, r5 br9 .BB.LABEL.1_8</pre>
64バイト	58バイト

4.1.3 複数箇所にある外部変数アクセスをテンポラリ変数により1箇所にまとめる

テンポラリ変数へのアクセスは外部変数へのアクセスと比べてレジスタ転送コードになる傾向が強いため、テンポラリ変数を活用して外部変数へのアクセス箇所を減らすことでコードサイズが小さくなる可能性があります。

変更前	変更後
<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: ; switch_break_bb mov 0x00000000, r10 jmp [r31] .BB.LABEL.1_5: ; switch_clause_bb st.w r0, LOWW(#_s)[r2] br9 .BB.LABEL.1_4 .BB.LABEL.1_6: ; switch_clause_bb1 movea 0x00005555, r0, r5 br9 .BB.LABEL.1_8 .BB.LABEL.1_7: ; switch_clause_bb2 ori 0x0000AAAA, r0, r5 .BB.LABEL.1_8: ; switch_clause_bb2 st.w r5, LOWW(#_s)[r2] br9 .BB.LABEL.1_4 .BB.LABEL.1_9: ; switch_clause_bb3 ori 0x0000FFFF, r0, r5 br9 .BB.LABEL.1_8</pre>	<pre>_fb: .stack_fb = 0 cmp 0x00000000, r6 mov 0x00000000, r2 bz9 .BB.LABEL.1_6 .BB.LABEL.1_1: ; if_else_bb addi 0xFFFFFC18, r6, r0 movea 0x00005555, r0, r2 bz9 .BB.LABEL.1_6 .BB.LABEL.1_2: ; if_else_bb9 addi 0xFFFFF830, r6, r0 bnz9 .BB.LABEL.1_4 .BB.LABEL.1_3: ; if_else_bb9.if_break_bb24_crit_edge ori 0x0000AAAA, r0, r2 br9 .BB.LABEL.1_6 .BB.LABEL.1_4: ; if_else_bb15 addi 0xFFFFF448, r6, r0 bnz9 .BB.LABEL.1_7 .BB.LABEL.1_5: ; if_else_bb15.if_break_bb24_crit_edge ori 0x0000FFFF, r0, r2 .BB.LABEL.1_6: ; if_break_bb24 movhi HIGHW1(#_s), r0, r5 st.w r2, LOWW(#_s)[r5] .BB.LABEL.1_7: ; label mov 0x00000000, r10 jmp [r31]</pre>
58バイト	50バイト

4.1.4 条件分岐先の同一文は分岐前に移動する

条件分岐の各々の分岐先に同一の文がある場合、条件分岐の前に移動して1箇所まとめてください。

変更前	変更後
<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>
<pre>_fw: .stack_fw = 12 prepare 0x00000061, 0x00000000 mov 0x00000002, r8 mov 0x00000001, r7 addi 0x00000000, r6, r20 movhi HIGHW1(#_s), r0, r21 mov 0x00000000, r6 bn9 .BB.LABEL.1_3 .BB.LABEL.1_1: ; if_then_bb jarl _func, r31 cmp r20, r10 bge9 .BB.LABEL.1_5 .BB.LABEL.1_2: ; if_then_bb8 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 jarl _func, r31 subr r0, r10 cmp r10, r20 bge9 .BB.LABEL.1_5 .BB.LABEL.1_4: ; if_then_bb17 ld.w LOWW(#_s)[r21], r2 add 0xFFFFFFFF, r2 st.w r2, LOWW(#_s)[r21] .BB.LABEL.1_5: ; if_break_bb21 mov 0x00000000, r10 dispose 0x00000000, 0x00000061, [r31]</pre>	<pre>_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.1_3 .BB.LABEL.1_1: ; if_then_bb cmp r20, r10 bge9 .BB.LABEL.1_5 .BB.LABEL.1_2: ; if_then_bb10 ld.w LOWW(#_s)[r2], r20 add 0x00000001, r20 st.w r20, LOWW(#_s)[r2] br9 .BB.LABEL.1_5 .BB.LABEL.1_3: ; if_else_bb subr r0, r10 cmp r10, r20 bge9 .BB.LABEL.1_5 .BB.LABEL.1_4: ; if_then_bb19 ld.w LOWW(#_s)[r2], r5 add 0xFFFFFFFF, r5 st.w r5, LOWW(#_s)[r2] .BB.LABEL.1_5: ; if_break_bb23 mov 0x00000000, r10 dispose 0x00000000, 0x00000041, [r31]</pre>
66バイト	62バイト

4.1.5 複雑なif文を論理的に等しいものに置き換える

if文の条件式が複雑である場合、等しい意味の簡単な式に置き換えてください。

変更前	変更後
<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>	<pre>extern int x; int fb (int s, int t) { s &= 1; t &= 1; if (!(s ^ t)) { x = 1; } return 0; }</pre>
<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: ; bb8.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_bb26 mov 0x00000001, r2 st.w r2, LOWW(#_x)[r5] .BB.LABEL.1_4: ; if_break_bb28 mov 0x00000000, r10 jmp [r31]</pre>	<pre>_fb: .stack_fb = 0 xor r6, r7 andi 0x00000001, r7, r0 bnz9 .BB.LABEL.1_2 .BB.LABEL.1_1: ; if_then_bb movhi HIGHW1(#_x), r0, r2 mov 0x00000001, r5 st.w r5, LOWW(#_x)[r2] .BB.LABEL.1_2: ; if_break_bb mov 0x00000000, r10 jmp [r31]</pre>
34バイト	22バイト

4.1.6 short, char型の変数はint型にする

CC-RH は ANSI-C の仕様により、short, char 型の演算を演算前に int 型に型変換してから演算するコードを生成します。また int 型の値を short, char 型の変数に代入するときにも型変換が発生します。あらかじめ int 型で変数定義しておくことで余分な型変換処理を削減できます。

注意:int 型に変更することで、変数や演算結果がとり得る値の範囲が変わります。型を変更する場合はプログラムの動作に影響が無いように注意してください。

変更前	変更後
<pre>unsigned char fw(unsigned char a, unsigned char b, unsigned char c) { unsigned char t = a + b; return t >> c; }</pre>	<pre>unsigned int fb(unsigned int a, unsigned int b, unsigned int c) { unsigned int t = a + b; return t >> c; }</pre>
<pre>_fw: .stack _fw = 0 add r6, r7 zxb r7 shr r8, r7 mov r7, r10 zxb r10 jmp [r31]</pre>	<pre>_fb: .stack _fb = 0 add r6, r7 mov r7, r10 shr r8, r10 jmp [r31]</pre>
14バイト	10バイト

4.1.7 switch文の共通なcaseの処理をまとめる

複数の case ラベルの分岐先の処理が同一である場合、case ラベルを移動して処理を 1 箇所に合わせてください。

変更前	変更後
<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: ; switch_clause_bb jr _dummy1 .BB.LABEL.1_3: ; return jmp [r31] .BB.LABEL.1_4: ; switch_clause_bb2 jr _dummy1 .BB.LABEL.1_5: ; switch_clause_bb4 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.1_3 .BB.LABEL.1_1: ; entry add 0xFFFFFFFFD, r2 cmp 0x00000002, r2 bl9 .BB.LABEL.1_4 .BB.LABEL.1_2: ; return jmp [r31] .BB.LABEL.1_3: ; switch_clause_bb jr _dummy1 .BB.LABEL.1_4: ; switch_clause_bb1 jr _dummy2</pre>
44バイト	28バイト

4.1.8 インライン展開する関数はstatic関数にする

インライン展開する関数が他のソース・ファイルから参照されていない場合、static 関数にしておくと関数本体のコードが削除されコードサイズが小さくなる可能性があります。

変更前	変更後
<pre>extern int s, t; #pragma inline fwsb void fwsb() { int tmp; tmp = s; s = t; t = tmp; } void fw() { if(s == 1){ fwsb(); } }</pre>	<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>
<pre>_fwsb: .stack_fwsb = 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>	<pre>_fb: .stack_fb = 0 movhi HIGHW1(#_s), r0, r2 ld.w LOWW(#_s)[r2], r5 cmp 0x00000001, r5 bnz9 .BB.LABEL.1_2 .BB.LABEL.1_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.1_2: ; return jmp [r31]</pre>
56バイト	30バイト

4.1.9 forループをdo whileループに置き換える

1 回以上ループを実行することが分かっている for 文の場合、do while 文に置き換えることでコードサイズが小さくなる可能性があります。

変更前	変更後
<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>	<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>
<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>	<pre>_fb: .stack_fb = 0 mul r6, r7, r0 mov 0x00000000, r2 mov #_array, r5 .BB.LABEL.1_1: ; bb st.w r0, 0x00000000[r5] add 0x00000004, r5 add 0x00000001, r2 cmp r7, r2 blt9 .BB.LABEL.1_1 .BB.LABEL.1_2: ; return jmp [r31]</pre>
28バイト	26バイト

また、条件식을等価比較に置き換えることでもコードサイズが小さくなる可能性があります。

変更前	変更後
<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>	<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>
<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</pre>	<pre>_fb2: .stack_fb2 = 0 mul r6, r7, r0 mov #_array, r2 .BB.LABEL.1_1: ; bb st.w r0, 0x00000000[r2] add 0x00000004, r2 loop r7, .BB.LABEL.1_1 .BB.LABEL.1_2: ; return jmp [r31]</pre>

<pre>br9 .BB.LABEL.1_1 .BB.LABEL.1_3: ; return jmp [r31]</pre>	
28バイト	22バイト

4.1.10 条件比較の結果によって0または1を代入する場合は条件式を直接代入する

if文の条件分岐先で0または1を代入する場合は、条件式の結果を直接代入してください。

変更前	変更後
<pre>extern int flag, s; int fw(void) { if(s > 100){ flag = 1; } else{ flag = 0; } return 0; }</pre>	<pre>extern int flag, s; int fb(void) { flag = (s > 100); return 0; }</pre>
<pre>_fw: .stack_fw = 0 movhi HIGHW1(#_s), r0, r2 ld.w LOWW(#_s)[r2], r2 addi 0xFFFFFFFF9B, 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>	<pre>_fb: .stack_fb = 0 movhi HIGHW1(#_s), r0, r2 ld.w LOWW(#_s)[r2], r2 addi 0xFFFFFFFF9C, r2, r0 setf 0x0000000F, r2 movhi HIGHW1(#_flag), r0, r5 st.w r2, LOWW(#_flag)[r5] mov 0x00000000, r10 jmp [r31]</pre>
30バイト	28バイト

4.1.11 2のべき乗の除算はシフト演算に置き換える

除算の除数が2のべき乗であり、被除数が正の値と分かっている場合は、除算をシフト演算に置き換えてください。

注意 被除数の型を unsigned int 型に変更することで同様の効果を得られる場合があります。

変更前	変更後
<pre>extern int s; void fw(void) { s = s / 2; }</pre>	<pre>extern int s; void fb(void) { s = s >> 1; }</pre>
<pre>_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]</pre>	<pre>_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]</pre>
18バイト	16バイト

4.1.12 外部変数および静的変数の符号を変える

式中に、型が signed でも unsigned でもよい ep 相対参照の外部変数および静的変数が含まれる場合は signed で宣言するとコードサイズが小さくなる可能性があります。

変更前	変更後
<pre>/* -Osmap */ unsigned char cw[256]; int fw() { return cw[0] + cw[16]; } </pre>	<pre>/* -Osmap */ signed char cb[256]; int fb() { return cb[0] + cb[16]; } </pre>
<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>	<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>
22バイト	20バイト

4.1.13 構造体を活用する

関連するデータを同一関数の中で何度も参照している場合、構造体を用いると相対アクセスを利用したコードが生成され易くなり、効率向上が期待できます。また、引数として渡す場合も効率が向上します。相対アクセスにはアクセス範囲に制限があるため、頻繁にアクセスするデータは構造体の先頭に集めると効果的です。

注意 -Omap または -Osmap オプションで同様の効果を得られる場合があります。

変更前	変更後
<pre>int a, b, c; void fw() { a = 1; b = 2; c = 3; } </pre>	<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>
<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>	<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>
32バイト	26バイト

4.1.14 2ビット以上のビットフィールドはchar型に変更する

ビットフィールドのメンバのサイズが2ビット以上の場合は、ビットフィールドは使用せずに char 型に変更してください。特定のビットからデータを取り出す処理が必要となるため、コードサイズが増加する可能性があります。

注意 使用する RAM のメモリ領域サイズは増加します。

変更前	変更後
<pre> struct { unsigned char b0:1; unsigned char b1:2; } dw; unsigned char dummy; int fw(void) { if(dw.b1){ dummy++; } return 0; } </pre>	<pre> struct { unsigned char b0:1; unsigned char b1; } db; unsigned char dummy; int fb(void) { if(db.b1){ dummy++; } return 0; } </pre>
<pre> _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 </pre>	<pre> _fb: .stack_fb = 0 movhi HIGHW1(#_db+0x00000001), r0, r2 ld.bu LOWW(#_db+0x00000001)[r2], r2 cmp 0x00000000, r2 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 </pre>
34バイト	32バイト

4.1.15 自動変数として使用できる大域変数は自動変数として定義する

自動変数として使用できるものは、外部変数ではなく自動変数として定義してください。外部変数は関数呼び出しやポインタ操作によって値が変化してしまう可能性があるため最適化の効率が悪くなります。

変更前	変更後
<pre> int tmp; void fw(int* a, int* b) { tmp = *a; *a = *b; *b = tmp; } </pre>	<pre> void fb(int* a, int* b) { int tmp; tmp = *a; *a = *b; *b = tmp; } </pre>
<pre> _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] </pre>	<pre> _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] </pre>
30バイト	18バイト

4.1.16 定数参照の際には絶対値の小さい値を割り当てる

マクロ定義や列挙型で定数値を使用する場合、絶対値の小さい値を割り当てるとコードサイズが小さくなる可能性があります。

変更前	変更後
<pre>#define CODEW (567) extern int data; void fw() { data= CODEW; }</pre>	<pre>#define CODEB (15) extern int data; void fb() { data= CODEB; }</pre>
<pre>_fw: .stack_fw = 0 movhi HIGHW1(#_data), r0, r2 movea 0x00000237, r0, r5 st.w r5, LOWW(#_data)[r2] jmp [r31]</pre>	<pre>_fb: .stack_fb = 0 movhi HIGHW1(#_data), r0, r2 mov 0x0000000F, r5 st.w r5, LOWW(#_data)[r2] jmp [r31]</pre>
14バイト	12バイト

4.1.17 関数をモジュール化する

呼び出し関係のある関数同士を同じソース・ファイルに定義すると、コードサイズが小さくなる可能性があります。

変更前	変更後
<pre>extern void fwsb(); void fw() { fwsb(); }</pre>	<pre>void fbsub() { ... } void fb() { fbsub(); }</pre>
<pre>_fw: .stack_fw = 0 jr fwsb</pre>	<pre>_fb: .stack_fb = 0 br9 fbsub</pre>
4バイト	2バイト

4.1.18 switch文の代わりにテーブルを活用する

switch 文の代わりにテーブルを使用することでコードサイズが小さくなる可能性があります。

変更前	変更後
<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>	<pre>const char chbuf[] = { 'a', 'x', 'b' }; int fb(int i) { if ((unsigned int)i < 3) { return (chbuf[i]); } return (0); }</pre>
<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.switch movea 0x00000061, r0, r10 br9 .BB.LABEL.1_6 .BB.LABEL.1_3: ; switch_clause_bb3 mov 0x00000000, r10 br9 .BB.LABEL.1_6 .BB.LABEL.1_4: ; switch_clause_bb1 movea 0x00000078, r0, r10 br9 .BB.LABEL.1_6 .BB.LABEL.1_5: ; switch_clause_bb2 movea 0x00000062, r0, r10 .BB.LABEL.1_6: ; switch_break_bb jmp [r31]</pre>	<pre>_fb: .stack_fb = 0 cmp 0x00000002, r6 bh9 .BB.LABEL.1_2 .BB.LABEL.1_1: ; if_then_bb mov #_chbuf, r2 add r6, r2 ld.b 0x00000000[r2], r10 jmp [r31] .BB.LABEL.1_2: ; bb9 mov 0x00000000, r10 jmp [r31] .section .const, const _chbuf: .db 0x61,0x78,0x62</pre>
40バイト	25バイト

4.1.19 割り込み関数の仕様を見直す

-Xreg_mode=22, -Xreserve_r2を指定すると割り込み関数内で退避・復帰するレジスタの数が減り、コードサイズを小さくできる場合があります。

変更前	変更後
<pre> /* -Xreg_mode=32 */ #pragma interrupt fw void fw() { sub(); } </pre>	<pre> /* -Xreg_mode=22, -Xreserve_r2 */ #pragma interrupt fb void fb() { sub(); } </pre>
<pre> _fw: .stack_fw = 88 movea 0xFFFFFAC, 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 0x00000006, r9, 0x00000000 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>	<pre> _fb: .stack_fb20 = 64 movea 0xFFFFF4C, 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 0x00000006, r9, 0x00000000 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>
188バイト	152バイト

また、#pragma interrupt の割り込み仕様を見直すことで、退避・復帰するレジスタの数が減り、コードサイズを小さくできます。

変更前	変更後
<pre>#pragma interrupt fw void fw() { sub(); }</pre>	<pre>#pragma interrupt fb (enable=false, callt=false, fpu=false) void fb() { sub(); }</pre>
<pre>_fw: .stack_fw = 88 movea 0xFFFFFAC, 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 0x00000006, r9, 0x00000000 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>	<pre>_fb: .stack_fb20 = 72 movea 0xFFFFFBC, 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</pre>
188バイト	132バイト

4.1.20 ライブラリ化する

複数のプロジェクトで共有する処理は、**関数や変数を小分けにしてライブラリにする**と、それぞれのプロジェクトごとに必要な関数・変数だけリンクしますので、サイズを削減できます。

4.2 実行速度の改善

変更前後の例は、-Ospeed オプション指定時のものです。

4.2.1 ループ展開する

ループする回数を削減すると、ループによる分岐命令のためのオーバーヘッドが減少します。

注意:コンパイラも同様の最適化を実施しますが、ソース・プログラムの記述内容や-Ounroll オプションの指定により効果が変わります。

変更前	変更後
<pre>extern int array[]; void fw(void) { int i; int *p; p = array; for(i = 16; i > 0; i--){ *p++ = 0; } }</pre>	<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>
<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>	<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>
34バイト, 36クロック	72バイト, 27クロック

4.2.2 除数は定数を使用する

定数による除算は、除算以外の演算に展開する最適化を実施しています。したがって、できるだけ定数の除算を使用してください。

注意 RH850 の divq 命令は、除数と被除数の有効ビット数の差が小さいほど高速になります。

変更前	変更後
<pre>int fw(int x, int y) { return x/y; }</pre>	<pre>int fb(int x) { return x/3; }</pre>
<pre>_fw: .stack _fw = 0 mov r6, r10 divq r7, r10, r0 jmp [r31]</pre>	<pre>_fb: .stack _fb = 0 mov 0x55555556, r2 mul r2, r6, r6 mov r6, r2 shr 0x0000001F, r2 mov r6, r10 add r2, r10 jmp [r31]</pre>
8バイト, 15~30クロック	20バイト, 17クロック

4.2.3 ループ制御変数の型を変更する

ループ制御変数を符号付き4バイト整数型(signed int/signed long)に変更すると、ループ展開最適化が適用され易くなり、実行速度が速くなる可能性があります。

変更前	変更後
<pre>extern int ub; extern char a[16]; void fw() { unsigned char i; for(i=0;i<ub;i++) { a[i]=0; } }</pre>	<pre>extern int ub; extern char a[16]; void fb() { int i; for(i=0;i<ub;i++) { a[i]=0; } }</pre>
<pre>_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]</pre>	<pre>_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 ble9 .BB.LABEL.1_5 .BB.LABEL.1_2: ; preheader.ul mov 0xFFFFFFFF, r5 mov r2, r6 and r2, r5 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]</pre>
42バイト, 24~159クロック	80バイト, 23~62クロック

4.3 データサイズの削減

データサイズの削減は、ROM から RAM へのマップで生成される初期値データ領域(*.data.R 等)のサイズ削減、また初期値データ転送処理、ゼロクリア処理の実行時間短縮にも効果があります。

4.3.1 データを整列する

RH850のメモリアクセス命令は、アクセスするアドレスが整列されている必要があります。そのためコンパイラでは変数を整列(順番を変えずにパディング領域を挿入)して配置します。

整列条件は、char 型は 1 バイト境界、short 型は 2 バイト境界、int 型は 4 バイト境界になります。変数を定義する際にはデータ長の長いものから定義してください。

変更前	変更後
<pre>char aw; int bw; char cw; int dw; char ew;</pre>	<pre>int bb; int db; char ab; char cb; char eb;</pre>
<pre>_aw: .ds (1) .align 4 _bw: .ds (4) _cw: .ds (1) .align 4 _dw: .ds (4) _ew: .ds (1)</pre>	<pre>_bb: .ds (4) .align 4 _db: .ds (4) _ab: .ds (1) _cb: .ds (1) _eb: .ds (1)</pre>
17バイト	11バイト

4.3.2 構造体メンバを整列する

構造体のメンバも変数と同様に整列して配置します。メンバをデータ長の長いものから配置することで変数のサイズが小さくなります。

注意: 構造体パッキング機能を使用すると、メンバを整列せずに詰めて配置します。この場合、変数のサイズは小さくなりますが、構造体変数へアクセスするコードのサイズが増加します。

変更前	変更後
<pre>struct { char aw; int bw; char cw; int dw; char ew; } fw;</pre>	<pre>struct { int bb; int db; char ab; char cb; char eb; } fb;</pre>
<pre>_fw: .ds (20)</pre>	<pre>_fb: .ds (12)</pre>
20バイト	12バイト

4.3.3 初期値を持つ変数をconst変数あるいはbss属性セクションに変更する

初期値を持つ外部変数と静的変数は、プログラム開始時に初期値データをROMからRAMへコピーします。このため、メモリ上に本来のサイズの2倍の領域を占有します。またコピー処理の実行時間もかかります。

値を変更しない変数はconst変数にしてROMに配置してください。

また、bss属性セクションはスタートアップルーチンで0で初期化します。**初期値が0である外部変数と静的変数は初期値を指定せずにbss属性セクションに配置**することで、変数の初期値データを削減できます。

4.3.4 アライン・ホールを減らす

各データ・セクションの先頭アドレスは4バイト境界に整列されている必要があります。このため、変数定義が複数のファイルに分かれている場合、リンク時にファイル間のアライン・ホールが生じる場合があります。1つのファイルに定義をまとめることで、ファイル間のアライン・ホールを解消することが可能です。

変更前	変更後
<pre>==w1.c== int v; char w; ==w2.c== char y; char z;</pre>	<pre>int v; char w; char y; char z;</pre>
10バイト	7バイト

1つのファイル内でも、#pragma section 指令により変数の配置先セクションを別々にすると、アライン・ホールが生じる場合があります。1つのセクションにまとめることでアライン・ホールを解消することが可能です。

変更前	変更後
<pre>#pragma section fw1 char fw; #pragma section fw2 char fw2;</pre>	<pre>#pragma section fb char fb1; char fb2;</pre>
5バイト	4バイト

第5章 改定履歴

Rev.	発行日	改定内容	
		ページ	ポイント
1.00	2020.7.7	-	初版発行

すべての商標および登録商標は、それぞれの所有者に帰属します。

ご注意書き

1. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器・システムの設計において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因して生じた損害（お客様または第三者いずれに生じた損害も含みます。以下同じです。）に関し、当社は、一切その責任を負いません。
 2. 当社製品、本資料に記載された製品データ、図、表、プログラム、アルゴリズム、応用回路例等の情報の使用に起因して発生した第三者の特許権、著作権その他の知的財産権に対する侵害またはこれらに関する紛争について、当社は、何らの保証を行うものではなく、また責任を負うものではありません。
 3. 当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
 4. 当社製品を、全部または一部を問わず、改造、改変、複製、リバースエンジニアリング、その他、不適切に使用しないでください。かかる改造、改変、複製、リバースエンジニアリング等により生じた損害に関し、当社は、一切その責任を負いません。
 5. 当社は、当社製品の品質水準を「標準水準」および「高品質水準」に分類しており、各品質水準は、以下に示す用途に製品が使用されることを意図しております。
標準水準： コンピュータ、OA 機器、通信機器、計測機器、AV 機器、家電、工作機械、パーソナル機器、産業用ロボット等
高品質水準： 輸送機器（自動車、電車、船舶等）、交通制御（信号）、大規模通信機器、金融端末基幹システム、各種安全制御装置等
当社製品は、データシート等により高信頼性、Harsh environment 向け製品と定義しているものを除き、直接生命・身体に危害を及ぼす可能性のある機器・システム（生命維持装置、人体に埋め込み使用するもの等）、もしくは多大な物的損害を発生させるおそれのある機器・システム（宇宙機器と、海底中継器、原子力制御システム、航空機制御システム、プラント基幹システム、軍事機器等）に使用されることを意図しておらず、これらの用途に使用することは想定していません。たとえ、当社が想定していない用途に当社製品を使用したことにより損害が生じて、当社は一切その責任を負いません。
 6. 当社製品をご使用の際は、最新の製品情報（データシート、ユーザーズマニュアル、アプリケーションノート、信頼性ハンドブックに記載の「半導体デバイスの使用上の一般的な注意事項」等）をご確認の上、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他指定条件の範囲内でご使用ください。指定条件の範囲を超えて当社製品をご使用された場合の故障、誤動作の不具合および事故につきましては、当社は、一切その責任を負いません。
 7. 当社は、当社製品の品質および信頼性の向上に努めていますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は、データシート等において高信頼性、Harsh environment 向け製品と定義しているものを除き、耐放射線設計を行っておりません。仮に当社製品の故障または誤動作が生じた場合であっても、人身事故、火災事故その他社会的損害等を生じさせないよう、お客様の責任において、冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、お客様の機器・システムとしての出荷保証を行ってください。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様の機器・システムとしての安全検証をお客様の責任で行ってください。
 8. 当社製品の環境適合性等の詳細につきましては、製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。かかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
 9. 当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器・システムに使用することはできません。当社製品および技術を輸出、販売または移転等する場合は、「外国為替及び外国貿易法」その他日本国および適用される外国の輸出管理関連法規を遵守し、それらの定めるところに従い必要な手続きを行ってください。
 10. お客様が当社製品を第三者に転売等される場合には、事前に当該第三者に対して、本ご注意書き記載の諸条件を通知する責任を負うものといたします。
 11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを禁じます。
 12. 本資料に記載されている内容または当社製品についてご不明な点がございましたら、当社の営業担当者までお問合せください。
- 注 1. 本資料において使用されている「当社」とは、ルネサス エレクトロニクス株式会社およびルネサス エレクトロニクス株式会社が直接的、間接的に支配する会社をいいます。
- 注 2. 本資料において使用されている「当社製品」とは、注 1 において定義された当社の開発、製造製品をいいます。

(Rev.4.0-1 2017.11)

本社所在地

〒135-0061 東京都江東区豊洲 3-2-24（豊洲フォレシア）

www.renesas.com

お問合せ窓口

弊社の製品や技術、ドキュメントの最新情報、最寄の営業お問合せ窓口に関する情報などは、弊社ウェブサイトをご覧ください。

www.renesas.com/contact/

商標について

ルネサスおよびルネサスロゴはルネサス エレクトロニクス株式会社の商標です。すべての商標および登録商標は、それぞれの所有者に帰属します。