お客様各位

カタログ等資料中の旧社名の扱いについて

2010年4月1日を以ってNECエレクトロニクス株式会社及び株式会社ルネサステクノロジが合併し、両社の全ての事業が当社に承継されております。従いまして、本資料中には旧社名での表記が残っておりますが、当社の資料として有効ですので、ご理解の程宜しくお願い申し上げます。

ルネサスエレクトロニクス ホームページ（http://www.renesas.com）

2010年4月1日
ルネサスエレクトロニクス株式会社

【発行】ルネサスエレクトロニクス株式会社（http://www.renesas.com）

【問い合わせ先】http://japan.renesas.com/inquiry

ご注意書き

1. 本資料に記載されている内容は本資料発行時点のものであり、予告なく変更することがあります。当社製品のご購入およびご使用にあたりましては、事前に当社営業窓口で最新の情報をご確認いただきますとともに、当社ホームページなどを通じて公開される情報に常にご注意ください。
2. 本資料に記載された当社製品および技術情報の使用に関連し発生した第三者の特許権、著作権その他の知的財産権の侵害等に関し、当社は、一切その責任を負いません。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。
3. 当社製品を改造、改変、複製等しないでください。
4. 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お客様または第三者に生じた損害に関し、当社は、一切その責任を負いません。
5. 輸出に際しては、「外国為替及び外国貿易法」その他輸出関連法令を遵守し、かかる法令の定めるところにより必要な手続を行ってください。本資料に記載されている当社製品および技術を大量破壊兵器の開発等の目的、軍事利用の目的その他軍事用途の目的で使用しないでください。また、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に使用することができません。
6. 本資料に記載されている情報は、正確を期すため慎重に作成したものですが、誤りがないことを保証するものではありません。万一、本資料に記載されている情報の誤りに起因する損害がお客様に生じた場合においても、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質水準を「標準水準」、「高品質水準」および「特定水準」に分類しております。また、各品質水準は、以下に示す用途に製品が使われることを意図しておりますので、当社製品の品質水準をご確認ください。お客様は、当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途に当社製品を使用することができません。また、お客様は、当社の文書による事前の承諾を得ることなく、意図されていない用途に当社製品を使用することができません。当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途または意図されていない用途に当社製品を使用したことによりお客様または第三者に生じた損害等に関し、当社は、一切その責任を負いません。なお、当社製品のデータ・シート、データ・ブック等の資料で特に品質水準の表示がない場合は、標準水準製品であることを表します。

   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

   高品質水準： 輸送機器（自動車、電車、船舶等）、交通用信号機器、防災・防犯装置、各種安全装置、生命維持を目的として設計されていない医療機器（厚生労働省定義の管理医療機器に相当）

   特定水準： 航空機器、航空宇宙機器、海底中継機器、原子力制御システム、生命維持のための医療機器（生命維持装置、人体に埋め込み使用するもの、治療行為（患部切り出し等）を行うもの、その他直接人命に影響を与えるもの）（厚生労働省定義の高度管理医療機器に相当）またはシステム等
8. 本資料に記載された当社製品のご使用につき、特に、最大定格、動作電源電圧範囲、放熱特性、実装条件その他諸条件につきましては、当社保証範囲内でご使用ください。当社保証範囲を超えて当社製品をご使用された場合の故障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。
9. 当社は、当社製品の品質および信頼性の向上に努めておりますが、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、当社製品は耐放射線設計については行っておりません。当社製品の故障または誤動作が生じた場合も、人身事故、火災事故、社会的損害など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、機器またはシステムとしての出荷保証をお願いいたします。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、お客様が製造された最終の機器・システムとしての安全検証をお願いいたします。
10. 当社製品の環境適合性等、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。お客様が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。
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固くお断りいたします。
12. 本資料に関する詳細についてのお問い合わせその他お気付きの点等がございましたら当社営業窓口までご照会ください。

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その総株主の議決権の過半数を直接または間接に保有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。

RENESAS

アプリケーション・ノート

CA850 Ver.2.50

Cコンパイラ　パッケージ

コーディング・テクニック

対象デバイス
V850シリーズ™

資料番号　U16076JJ1V0AN00（第1版）
発行年月　May 2002 N　CP(K)

〔メ　モ〕

目次要約

V850**シリーズ，**V853, V850/SA1, V850/SB1, V850/SB2, V850/SF1, V850/SV1, V850E/MS1, V850E/MA1, V850E/MA2, V850E/IA1, V850E/IA2**は，日本電気株式会社の商標です。**

Windows**は米国**Microsoft Corporation**の米国およびその他の国における登録商標または商標です。**

巻末にアンケート・コーナを設けております。このドキュメントに対するご意見をお気軽にお寄せください。

はじめに

対 象 者　このアプリケーション・ノートはV850シリーズの各製品の応用システムを設計，開発するユーザを対象としています。

目　　的　このアプリケーション・ノートは，V850シリーズ用Cコンパイラ CA850を使用して，最適化オプションを指定したあとにコード・サイズをさらに小さくする，または，実効速度を高速化するためのコーディング・テクニックについてユーザに理解していただくことを目的としています。

構　　成　このマニュアルは，大きく分けて次の内容で構成しています。

- 概　　要
- コード・サイズの削減
- 実行速度の高速化
- 変数の定義

読 み 方　このマニュアルの読者には，電気，論理回路，マイクロコンピュータ，C言語，アセンブラに関する一般知識を必要とします。

V850シリーズのハードウエア機能を知りたいとき
　→各製品の**ユーザーズ・マニュアル　ハードウエア編**を参照してください。

V850シリーズの命令機能を知りたいとき
　→V850**シリーズ　ユーザーズ・マニュアル　アーキテクチャ編**を参照してください。

凡　　例　データ表記の重み　：左が上位桁，右が下位桁
注　：本文中につけた注の説明
注意　：気をつけて読んでいただきたい内容
備考　：本文の補足説明
数の表記　：2進数…XXXXまたはXXXXB
10進数…XXXX
16進数…XXXXH
2のべき数を示す接頭語（アドレス空間，メモリ容量）：
K（キロ）：$2^{10} = 1024$
M（メガ）：$2^{20} = 1024^2$
G（ギガ）：$2^{30} = 1024^3$

関連資料　**このマニュアルを使用する場合は，次の資料もあわせてご覧ください。**
関連資料は暫定版の場合がありますが，この資料では「暫定」の表示をしておりません。
あらかじめご了承ください。

V850**シリーズに関する資料（ユーザーズ・マニュアル）**

資　料　名		資料番号	
		和文	英文
IE-703002-MC（V853™, V850/SA1™, V850/SB1™, V850/SB2™, V850/SF1™, V850/SV1™**用インサーキット・エミュレータ）**		U11595J	U11595E
IE-V850E-MC（V850E/IA1™, V850E/IA2™**用インサーキット･エミュレータ），** IE-V850E-MC-A（V850E/MA1™, V850E/MA2™**用インサーキット・エミュレータ）**		U14487J	U14487E
IE-703003-MC-EM1（V853**用インサーキット・エミュレータ・オプション・ボード）**		U11596J	U11596E
IE-703017-MC-EM1（V850/SA1**用インサーキット・エミュレータ・オプション・ボード）**		U12898J	U12898E
IE-703037-MC-EM1（V850/SB1, V850/SB2**用インサーキット･エミュレータ･オプション・ボード）**		U14151J	U14151E
IE-703040-MC-EM1（V850/SV1**用インサーキット・エミュレータ・オプション・ボード）**		U14337J	U14337E
IE-703079-MC-EM1（V850/SF1**用インサーキット・エミュレータ・オプション・ボード）**		U15447J	U15447E
IE-703102-MC（V850E/MS1™**用インサーキット・エミュレータ）**		U13875J	U13875E
IE-703102-MC-EM1, IE-703102-MC-EM1-A（V850E/MS1**用インサーキット・エミュレータ・オプション・ボード）**		U13876J	U13876E
IE-703107-MC-EM1（V850E/MA1**用インサーキット・エミュレータ・オプション・ボード）**		U14481J	U14481E
IE-703116-MC-EM1（V850E/IA1**用インサーキット・エミュレータ・オプション・ボード）**		U14700J	U14700E
CA850 Ver.2.50　**Cコンパイラ・パッケージ**	**操作編**	U16053J	**作成予定**
	C言語編	U16054J	U16054E
	PM plus**編**	U16055J	**作成予定**
	アセンブリ言語編	U16042J	U16042E
ID850 Ver.2.40　**統合ディバッガ**	**操作編**　Windows®**ベース**	U15181J	U15181E
SM850 Ver.2.40　**システム・シミュレータ**	**操作編**　Windows**ベース**	U15182J	U15182E
SM850 Ver.2.00**以上　システム・シミュレータ**	**外部部品ユーザ・オープン・インタフェース仕様編**	U14873J	U14873E
RX850 Ver.3.13**以上　リアルタイム**OS	**基礎編**	U13430J	U13430E
	インストレーション編	U13410J	U13410E
	テクニカル編	U13431J	U13431E
RX850 Pro Ver.3.13　**リアルタイム**OS	**基礎編**	U13773J	U13773E
	インストレーション編	U13774J	U13774E
	テクニカル編	U13772J	U13772E
RX-NET　**ネットワーク・ライブラリ**（TCP/IP）		U15083J	－
RX-NET　**ネットワーク・ライブラリ**（PPP）		U15303J	－
RX-NET　**ネットワーク・ライブラリ**（DNS）		U15304J	－
RX-NET　**ネットワーク・ライブラリ**（DHCP）		U15382J	－
RX-NET　**ネットワーク・ライブラリ**（SMTP）		U15505J	－
RX-NET　**ネットワーク・ライブラリ**（POP）		U15539J	－
RD850　Ver.3.01　**タスク・ディバッガ**		U13737J	U13737E
RD850 Pro Ver.3.01　**タスク・ディバッガ**		U13916J	U13916E
AZ850 Ver.3.10　**システム・パフォーマンス・アナライザ**		U14410J	U14410E
PG-FP4　**フラッシュ・メモリ・プログラマ**		U15260J	U15260E
CA850 Ver.2.50　**Cコンパイラ・パッケージ（アプリケーション・ノート）**	**コーディング・テクニック**	**このマニュアル**	**作成予定**

目　　次

第1章　概　　要

このアプリケーション・ノートは，V850シリーズ用Cコンパイラ CA850を使用して，最適化オプションを指定したあとにコード・サイズをさらに小さくする，または，実効速度を高速化するためのコーディング・テクニックについて説明します。

CA850の詳細については，CA850**関連のユーザーズ・マニュアル**を参照してください。

なお，各項目の例に記述された削減量はその例に関するものであり，他に適用した場合の削減量は個々のケースにより若干異なります。

また，ソース変更を行う際には，次のような点に注意する必要があります。

ソース変更によりレジスタの使用状況が変わるため，意図しない箇所において，それまで最適化されずに残っていたレジスタ転送が削除されたり，逆に最適化が効かなくなって冗長なレジスタ転送が残ったりする可能性があります。

テンポラリ変数を追加することにより，新たなレジスタ変数用レジスタが使用されるようになり，それにともない関数の入口／出口でそのレジスタの退避／復帰が追加されることがあります。この場合，退避／復帰のコードの分コード・サイズが増加します。

このアプリケーション・ノートで記載してある例の出力アセンブラは，サイズ優先最適化（-Os）を指定してコンパイルしたものです。サイズ優先最適化以外の最適化を指定した場合には，結果が異なりますので，注意してください。

第2章　コード・サイズの削減

ここでは，コード・サイズを小さくする，コーディング・テクニックを紹介します。
また，コード・サイズを小さくすることにより，実効速度も高速化できる場合もあります。

2. 1　switch文の代わりにif～else文を使う

CA850はswitch文に対して，次の2つの条件が同時にみたされる場合に，テーブル分岐形式のコードを生成します。

- caseラベルの個数が4個以上
- ラベルの値の下限と上限の差がcaseの数の3倍まで

この場合，caseの数がおよそ16個（ただし，この数はswitchの式の形式やラベルの値の分布によって異なります）以下ならば，等価なif～else文に変更し，比較命令と分岐命令の並びにした方が，コード・サイズが小さくなります。

なお，switchの式が外部変数参照や複雑な式の場合は，いったんテンポラリ変数に値を代入してifの式ではそのテンポラリ変数を参照するように変更する必要があります。

ただし，V850Eの場合にはswitch命令を出力しますので，switch文の方がコード・サイズが小さくなります。

注意　ソース記述を変更しなくても，-Xcaseオプションでswitch文の展開コードをファイル単位で指定できます。

次にプログラム例を示します。

備考　xはauto変数であるとします。

変更前	変更後

`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);`

【V850での出力アセンブラ】

変更前		変更後	
プログラム	サイズ〔byte〕	プログラム	サイズ〔byte〕
`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		
合計コード・サイズ	94 bytes	**合計コード・サイズ**	76 bytes

【V850Eでの出力アセンブラ】

変更前		変更後	
プログラム	サイズ〔byte〕	プログラム	サイズ〔byte〕
`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
合計コード・サイズ	70 bytes	合計コード・サイズ	76 bytes

2. 2　switch文やif～else文の分岐先での同一の外部変数への代入はテンポラリ変数を介して行う

switch文やif～else文の分岐先のおのおので同じ外部変数へ違う値を代入する場合は，おのおのの箇所ではいったんテンポラリ変数へ代入し，再び合流したあとにテンポラリ変数から元の外部変数へ代入を行うことにより，コード・サイズが削減される可能性があります。

これは，外部変数はレジスタに割り付けられることが少ないため，外部変数への代入はメモリへのストア命令（4bytes）となりますが，テンポラリ変数への代入は多くの場合レジスタ転送（2bytes）になるためです。

次にプログラム例を示します。

備考　sは外部変数であるとします。

変更前	変更後

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:
	;

【V850での出力アセンブラ】

変更前			変更後		
プログラム		サイズ (byte)	プログラム		サイズ (byte)
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:					
合計コード・サイズ		70 bytes	合計コード・サイズ		58 bytes

【V850Eでの出力アセンブラ】

変更前			変更後		
プログラム		サイズ (byte)	プログラム		サイズ (byte)
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:					
合計コード・サイズ		66 bytes	合計コード・サイズ		54 bytes

2. 3　if～else文の分岐先が同じ変数への代入文のみの場合は一方をif文の前に移動する

if～else文の分岐先のおのおのが同じ変数へ異なる値を代入するような文のみを含む場合，一方をif文の前に移動することにより，elseのブロックが削除されてifのブロックからelseのブロックへの後へのジャンプ命令が減るため，コード・サイズが削減される可能性があります。

次にプログラム例を示します。

備考　sは外部変数であるとします。

変更前	変更後

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

【V850での出力アセンブラ】

変更前			変更後		
プログラム		サイズ (byte)	プログラム		サイズ (byte)
	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:					
合計コード・サイズ		20 bytes	合計コード・サイズ		18 bytes

【V850Eでの出力アセンブラ】

変更前			変更後		
プログラム		サイズ (byte)	プログラム		サイズ (byte)
	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:					
合計コード・サイズ		20 bytes	合計コード・サイズ		18 bytes

2. 4　外部変数へのアクセスをテンポラリ変数のアクセスへ置き換える

外部変数アクセスはロード／ストアともに4bytes必要なので，2. 2のような代入以外の場合にも，外部変数の値をいったんテンポラリ変数に代入してそのテンポラリ変数を使いまわすと，メモリ・アクセスがレジスタ・アクセスに変わりコード・サイズが減る可能性があります。

次にプログラム例を示します。

備考　sは外部変数であるとします。

変更前	変更後

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;

【V850での出力アセンブラ】

変更前			変更後		
プログラム		サイズ (byte)	プログラム		サイズ (byte)
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:					
合計コード・サイズ		70 bytes	合計コード・サイズ		62 bytes

【V850Eでの出力アセンブラ】

変更前		変更後	
プログラム	サイズ (byte)	プログラム	サイズ (byte)
`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:`			
合計コード・サイズ	64 bytes	合計コード・サイズ	56 bytes

2.5　分岐後の同一の文は分岐前に移動する

分岐後のおのおのにおいて，同一の代入文や関数呼び出しが存在する場合には，分岐前に移動可能であれば移動してください。

その文の評価結果が参照されているならば，いったんテンポラリ変数へ代入しそのテンポラリを参照するようにしてください。

次にプログラム例を示します。

備考　**sは外部変数であるとします。**

変更前	変更後

`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--;`
	`  }`
	`}`

【V850での出力アセンブラ】

変更前			変更後		
プログラム		サイズ (byte)	プログラム		サイズ (byte)
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:					
合計コード・サイズ		62 bytes	合計コード・サイズ		54 bytes

【V850Eでの出力アセンブラ】

変更前			変更後		
プログラム		サイズ (byte)	プログラム		サイズ (byte)
	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:					
合計コード・サイズ		62 bytes	合計コード・サイズ		54 bytes

2. 6　合流前の同一の文は合流後に移動する

分岐後のおのおのにおいて，同一の代入文や関数呼び出しが存在する場合に，2. 5のように分岐前に移動することが不可能であって，合流後に移動することは可能であるならば，合流後に移動してください。

次にプログラム例を示します。

備考　s, tは外部変数であるとします。

変更前	変更後

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

【V850での出力アセンブラ】

変更前			変更後		
プログラム		サイズ (byte)	プログラム		サイズ (byte)
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:					
合計コード・サイズ		52 bytes	合計コード・サイズ		48 bytes

【V850Eでの出力アセンブラ】

変更前			変更後		
プログラム		サイズ (byte)	プログラム		サイズ (byte)
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:					
合計コード・サイズ		50 bytes	合計コード・サイズ		46 bytes

2.7　分岐後の同一関数の引き数が異なる呼び出しはテンポラリ変数を使用し合流後にまとめる

分岐後のおのおので同一の関数を異なる引き数を用いて呼び出しを行っている場合，関数呼び出しを合流後に移動可能であるならば移動してください。

このとき，もともとの呼び出しの箇所ではその異なる引き数をテンポラリ変数へ代入し，呼び出しにおいてはそのテンポラリ変数を引き数として用いてください。

次にプログラム例を示します。

備考　sは外部変数であるとします。

変更前	変更後

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

【V850での出力アセンブラ】

変更前		変更後	
プログラム	サイズ (byte)	プログラム	サイズ (byte)
`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:`			
合計コード・サイズ	30 bytes	**合計コード・サイズ**	24 bytes

【V850Eでの出力アセンブラ】

変更前		変更後	
プログラム	サイズ (byte)	プログラム	サイズ (byte)
`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:`			
合計コード・サイズ	30 bytes	**合計コード・サイズ**	24 bytes

2. 8　複雑なif文を論理的に等しいものに置き換える

if～elseの組み合わせにおいて，複数のケースで同じ処理を実行する場合，別の条件を用いてその“複数のケース”を一つにまとめられるのであれば，まとめて冗長な部分を削除してください。

次にプログラム例を示します。

備考　xの初期値が0で，sおよびtの値は0または1のいずれかである，という条件がそろっている場合，この例のように変形することが可能です。s, t, u, vは外部変数であるとします。

変更前	変更後

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

【V850での出力アセンブラ】

変更前		変更後	
プログラム	サイズ (byte)	プログラム	サイズ (byte)
`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:`			
合計コード・サイズ	70 bytes	合計コード・サイズ	34 bytes

【V850Eでの出力アセンブラ】

変更前		変更後	
プログラム	サイズ (byte)	プログラム	サイズ (byte)
`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:`			
合計コード・サイズ	70 bytes	合計コード・サイズ	34 bytes

2. 9　for／whileループをgotoループに変形する

CA850は，forやwhileのように最初に条件判断の式のあるループに対して，次のイメージ図のように条件判断の式を2回生成します。

このようなループの変形は，コンパイラの最初のフェーズであるフロント・エンド（構文解析部）によって行われますが，これはその後の最適化によって最初の条件判断が削除されることが多く，実行速度向上の点ではこのように変形するのが有利であるためです。

ところが，削除されなかった場合コード・サイズの面では冗長となります。

【イメージ図】

構文	イメージ

for (**文**1; **式**2; **文**3) {
　　ループ・ボディ
}
	文1;
	if (**式**2) {
	　　do {
	　　　　ループ・ボディ
	　　　　文3;
	　　} while (**式**2) ;
	}

while (**式**1) {
　　ループ・ボディ
}
	if (**式**1) {
	　　do {
	　　　　ループ・ボディ
	　　} while (**式**1) ;
	}

したがって，1回目の条件判断の式が最適化により削除されない場合には，次のようにgotoで構成されるループに変形することで，条件判断の式を1個に減らすことができます。

次のイメージ図のようにしてください。

【イメージ図】

forループ	**文**1; loop_bgn: 　　if (! **式**2) goto loop_end; 　　**ループ・ボディ** 　　**文**3; 　　goto loop_bgn; loop_end:
whileループ	loop_bgn: 　　if (! **式**1) goto loop_end; 　　**ループ・ボディ** 　　goto loop_bgn; loop_end:

次にプログラム例を示します。

備考　s, array[ ]**は外部変数であるとします。**

変更前	変更後

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:
	　;

【V850での出力アセンブラ】

変更前		変更後	
プログラム	サイズ (byte)	プログラム	サイズ (byte)
`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:`			
合計コード・サイズ	38 bytes	合計コード・サイズ	32 bytes

【V850Eでの出力アセンブラ】

変更前		変更後	
プログラム	サイズ (byte)	プログラム	サイズ (byte)
`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:`			
合計コード・サイズ	38 bytes	合計コード・サイズ	32 bytes

2. 10　ループを展開する

実行回数が少なく，かつループ・ボディが小さい場合には，展開した方が小さくなる場合があります。この場合，さらに実行速度も向上します。

次にプログラム例を示します。

備考　array[ ]は外部変数であるとします。

変更前	変更後

`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;`

【V850での出力アセンブラ】

変更前		変更後	
プログラム	サイズ (byte)	プログラム	サイズ (byte)
`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		
合計コード・サイズ	22 bytes	合計コード・サイズ	16 bytes

【V850Eでの出力アセンブラ】

変更前		変更後	
プログラム	サイズ (byte)	プログラム	サイズ (byte)
`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		
合計コード・サイズ	22 bytes	合計コード・サイズ	16 bytes

2. 11　変数の生存範囲を小さくする

スタック変数に値が代入されてからその値が実際に参照されるまでに間がある場合，その間レジスタが占有されて他の変数がレジスタに割り付けられる機会が減ります。コンパイラの最適化により値の代入が後ろへ移動することが多いですが，間に関数呼び出しを含む場合は最適化されないこともあります。

このような場合には，実際に参照する直前に代入を行うように変更することにより，他の変数のレジスタ割り付けの機会が増えてメモリ・アクセスが減り，コード・サイズが小さくなります。

次にプログラム例を示します。

備考　s1, s2, s3, array[ ]は外部変数であるとします。

変更前	変更後

`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++;`
`  }`
`  }`
`}`
	`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++;`
	`  }`
	`  }`
	`}`

【V850での出力アセンブラ】

変更前			変更後		
プログラム		サイズ (byte)	プログラム		サイズ (byte)
`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			
合計コード・サイズ		66 bytes	合計コード・サイズ		62 bytes

【V850Eでの出力アセンブラ】

変更前			変更後		
プログラム		サイズ (byte)	プログラム		サイズ (byte)
	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			
合計コード・サイズ		64 bytes	合計コード・サイズ		60 bytes

2. 12　“帰納変数の削除”を行う

次の例のように，ループの制御を行う変数を帰納変数（誘導変数）といいますが，ループの制御を他の変数を用いて行うように変更することで帰納変数を削除する最適化が“帰納変数の削除”です。

この最適化はCA850でも実装されていますが，適用される条件が限られるため，すべての場合を最適化することはできません。

したがって，次のようにプログラムを修正することにより，この最適化を“手動”で行ってください。

次にプログラム例を示します。

備考　x, *tableは外部変数であるとします。

変更前	変更後

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);
	  }
	}

【V850での出力アセンブラ】

変更前			変更後		
プログラム		サイズ (byte)	プログラム		サイズ (byte)
	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			
合計コード・サイズ		68 bytes	合計コード・サイズ		62 bytes

【V850Eでの出力アセンブラ】

変更前			変更後		
プログラム		サイズ (byte)	プログラム		サイズ (byte)
	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			
合計コード・サイズ		60 bytes	合計コード・サイズ		54 bytes

2. 13　（unsigned）short, char型の変数は（unsigned）int型にする

（unsigned）ANSI-Cの仕様により，（unsigned）short, （unsigned）char型の変数は演算時にint型またはunsigned int型に拡張されるため，これらの変数を使用したプログラムに対しては（特にこれらの変数がレジスタに割り付けられた場合には）型変換命令が多く生成されます。

（unsigned）int型にすればこの型変換が不要となるため，コード・サイズが削減されます。

特に，比較的レジスタに割り付けられやすいスタック変数は，できるだけ（unsigned）int型を用いた方がよいです。

次にプログラム例を示します。

備考　array[ ], *pは外部変数であるとします。

変更前	変更後

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);
	}

【V850での出力アセンブラ】

変更前			変更後		
プログラム		サイズ (byte)	プログラム		サイズ (byte)
	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			
合計コード・サイズ		38 bytes	合計コード・サイズ		34 bytes

【V850Eでの出力アセンブラ】

変更前			変更後		
プログラム		サイズ (byte)	プログラム		サイズ (byte)
	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			
合計コード・サイズ		36 bytes	合計コード・サイズ		34 bytes

2. 14　代入文の次で代入された値が参照されている場合，文を一つにまとめる

代入文の次でその代入された値が参照されている場合，参照されている箇所を代入文で置換して一つにまとめることにより，余分なレジスタ転送が削除されてコード・サイズが削減される可能性があります。

ただし，多くの場合はコンパイラの最適化によって冗長なレジスタ転送が削除されているため，コード・サイズは変わりません。

次にプログラム例を示します。

備考　s, tは外部変数であるとします。

変更前	変更後

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

この例はコンパイラの最適化によって同じコード・サイズになります。

2. 15　if～else文の分岐先が分岐条件の結果を返すreturn文である場合if～else文を削除する

if～else文の分岐先のおのおのがreturn文のみを含み，その戻り値が分岐条件の結果そのものであるならば，分岐条件の式の値を返すようして，if～else文を削除してください。

次にプログラム例を示します。

備考　s1, s2は外部変数であるとします。

変更前	変更後

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

【V850での出力アセンブラ】

変更前		変更後	
プログラム	サイズ (byte)	プログラム	サイズ (byte)
`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:`			
合計コード・サイズ	18 bytes	合計コード・サイズ	14 bytes

【V850Eでの出力アセンブラ】

変更前		変更後	
プログラム	サイズ (byte)	プログラム	サイズ (byte)
`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:`			
合計コード・サイズ	18 bytes	合計コード・サイズ	14 bytes

2. 16　比較演算のオペランドの一方が16, -17の定数の場合は，条件を変更して15, -16にする

比較命令cmpは，一方のオペランドが-16～15の即値以外の場合にはアセンブラにより命令展開が行われて複数命令になります。

したがって，条件を変更することによりオペランドの値を-16～15の範囲に抑えられれば展開が抑止され，コード・サイズが小さくなります。

具体的には，for, ifなどの条件式において比較に用いられる値を変更してください。

次にプログラム例を示します。

備考　sは外部変数であるとします。

変更前	変更後

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

この例はコンパイラの最適化によって同じコード・サイズになります。

2. 17　変数を初期化する

関数内でauto変数が初期化されずに使用されている場合，その変数がレジスタに割り付けられずにメモリのまま残るため，コード・サイズが大きくなることがあります。

例では，switchのいずれのケースにも該当しない場合に変数aが初期化されずにreturn文で参照されることになります。実際には必ずいずれかのケースに該当するとしても，レジスタ割り付けにおいてプログラムを解析する際には分からないため，初期化されない場合があると見なされます。このような場合にはCA850のレジスタ割り付けでは割り付けられません。

そこで，初期化を追加することにより，レジスタに割り付けられるようにするとコード・サイズが削減されます。

次にプログラム例を示します。

備考　aはauto変数であるとします。

変更前	変更後

`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;`

【V850での出力アセンブラ】

変更前		変更後	
プログラム	サイズ (byte)	プログラム	サイズ (byte)
`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		
合計コード・サイズ	26 bytes	合計コード・サイズ	14 bytes

【V850Eでの出力アセンブラ】

変更前		変更後	
プログラム	サイズ (byte)	プログラム	サイズ (byte)
`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		
合計コード・サイズ	26 bytes	合計コード・サイズ	14 bytes

2. 18　戻り値のない関数はvoidと宣言する

戻り値のない関数はvoid宣言してください。

冗長なレジスタの転送命令が削除されます。

次にプログラム例を示します。

変更前	変更後

`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;`
	`  }`
	`}`

【V850での出力アセンブラ】

変更前		変更後	
プログラム	サイズ (byte)	プログラム	サイズ (byte)
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		
合計コード・サイズ	26 bytes	合計コード・サイズ	24 bytes

【V850Eでの出力アセンブラ】

変更前		変更後	
プログラム	サイズ (byte)	プログラム	サイズ (byte)
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		
合計コード・サイズ	26 bytes	合計コード・サイズ	24 bytes

2. 19　switch文の共通なcaseの処理はまとめる

switch文のcaseの処理が同じものの記述をまとめるとコード・サイズが削減される可能性があります。

次にプログラム例を示します。

備考　xは外部変数であるとします。

変更前	変更後

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;
	}

【V850での出力アセンブラ】

変更前		変更後	
プログラム	サイズ (byte)	プログラム	サイズ (byte)
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:			
合計コード・サイズ	72 bytes	合計コード・サイズ	54 bytes

【V850Eでの出力アセンブラ】

変更前		変更後	
プログラム	サイズ (byte)	プログラム	サイズ (byte)
`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:`			
合計コード・サイズ	48 bytes	合計コード・サイズ	30 bytes

2. 20　値が同じreturn文はまとめる

同じ値を返すreturn文の記述をまとめると，コード・サイズが削減される可能性があります。

次にプログラム例を示します。

備考　s, t, uは外部変数であるとします。

変更前	変更後

`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;`

【V850での出力アセンブラ】

変更前		変更後	
プログラム	サイズ (byte)	プログラム	サイズ (byte)
`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:`			
合計コード・サイズ	44 bytes	合計コード・サイズ	32 bytes

【V850Eでの出力アセンブラ】

変更前		変更後	
プログラム	サイズ (byte)	プログラム	サイズ (byte)
`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:`			
合計コード・サイズ	44 bytes	合計コード・サイズ	32 bytes

2. 21　インライン展開する関数はstatic関数にする

インライン展開を指定した関数が，他のファイルからは参照されていない場合に，static関数にすることにより関数本体のコードが最適化により削除されます。これにより，コード・サイズが削減される可能性があります。

ただし，プログラム[注]中にアセンブラ記述がある場合には，この最適化は行われないため関数の本体のコードは出力されます。この場合には，アセンブラ記述のある関数を別のファイルにしてください。

注　コンパイルするときのプログラム（インクルード・ファイルを含みます）です。

例　static関数と通常の関数の場合

説明のため変更前と変更後で関数名を変更していますので，注意してください。

変更前	変更後

`#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();`
	`  }`
	`}`

関数func1subのコードは出力されますが，関数func2subのコードは出力されません。

例　アセンブラ記述がある／ない場合

説明のため変更前と変更後で関数名を変更していますので，注意してください。

変更前	変更後

`#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");`
`}`
	`#pragma inline func4sub`
	
	`int s,t;`
	
	`static void func4sub()`
	`{`
	`  int tmp;`
	
	`  tmp = s;`
	`  s = t;`
	`  t = tmp;`
	`}`
	
	`void func4()`
	`{`
	`  if(s == 1){`
	`    func4sub();`
	`  }`
	`}`

変更前のプログラムにはアセンブラ記述がありますので，関数func3sub**のコードは出力されます。しかし，変更後のプログラムにはアセンブラ記述がありませんので，関数**func4sub**のコードは出力されません。**

第3章　実行速度の高速化

この章では，本来ならばコンパイラにまかせる処理を，人間が記述した方が実効速度の高速化ができる例を記述しました。

3. 1　配列の連続アクセスはポインタを使用する

ループ内での配列の連続アクセスは，ポインタで行ってください。ポインタを使用しない場合，配列の添え字から実アドレスを求める処理が，毎回出力される可能性があります。

次にプログラム例を示します。

備考　sum, array[ ]は外部変数であるとします。

変更前	変更後

`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++;`
	`}`

【V850 での出力アセンブラ】

変更前		変更後	
プログラム	サイズ (byte)	プログラム	サイズ (byte)
`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		
合計コード・サイズ	34 bytes	合計コード・サイズ	32 bytes

【V850E での出力アセンブラ】

変更前		変更後	
プログラム	サイズ (byte)	プログラム	サイズ (byte)
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		
合計コード・サイズ	34 bytes	合計コード・サイズ	32 bytes

3. 2　外部変数へのアクセスをテンポラリの変数のアクセスに置き換える

ループ内では，できるだけ外部変数を使用しないようにしてください。

アドレス計算やメモリ・アクセス（ロード／ストア命令）が毎回出力される可能性がありますので，テンポラリの変数のアクセスに置き換えてください。

次にプログラム例を示します。

備考　sum, array[ ]**は外部変数であるとします。**

変更前	変更後

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;

【V850 での出力アセンブラ】

変更前			変更後		
プログラム		サイズ (byte)	プログラム		サイズ (byte)
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			
合計コード・サイズ		32 bytes	合計コード・サイズ		28 bytes

【V850E での出力アセンブラ】

変更前			変更後		
プログラム		サイズ (byte)	プログラム		サイズ (byte)
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			
合計コード・サイズ		32 bytes	合計コード・サイズ		28 bytes

3. 3　ループの終了条件に変数式は使用しない

ループの終了条件には，できるだけ変数式を使用しないでテンポラリ変数を使用して記述してください。
終了式の比較のために演算が毎回出力される可能性があります。

次にプログラム例を示します。

備考　array[ ][ ]は外部変数であるとします。

変更前	変更後

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;
	}

この例はコンパイラの最適化によって同じコードになります。

3.4　ループの終了条件に0との比較を使用する

ループの終了条件に0との比較式を使用すると，ループ1回ごとの終了条件の演算が速くなる可能性があります。また，使用するレジスタ数が減る可能性もあります。

次にプログラム例を示します。

備考　array[ ][ ]は外部変数であるとします。

変更前	変更後

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;
	}

【V850 での出力アセンブラ】

変更前			変更後		
プログラム		サイズ (byte)	プログラム		サイズ (byte)
	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:		
合計コード・サイズ		38 bytes	合計コード・サイズ		38 bytes

備考　この例では，合計コード・サイズに変わりがありません。

【V850E での出力アセンブラ】

変更前			変更後		
プログラム		サイズ (byte)	プログラム		サイズ (byte)
	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:		
合計コード・サイズ		36 bytes	合計コード・サイズ		36 bytes

備考　この例では，合計コード・サイズに変わりがありません。

3.5　ループの展開をする

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ヘッドが減少します。

次にプログラム例を示します。

備考　array[ ]は外部変数であるとします。

変更前	変更後

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;
	}

【V850 での出力アセンブラ】

変更前		変更後	
プログラム	サイズ (byte)	プログラム	サイズ (byte)
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
合計コード・サイズ	20 bytes	**合計コード・サイズ**	62 bytes

【V850E での出力アセンブラ】

変更前		変更後	
プログラム	サイズ (byte)	プログラム	サイズ (byte)
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
合計コード・サイズ	20 bytes	**合計コード・サイズ**	62 bytes

3. 6　ポインタの最適化をする

次にプログラム例を示します。

備考　array[ ]**は外部変数で，**N**は**10**であるとします。**

変更前

```
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;
}
```

変更後

```
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;
}
```

【V850 **での出力アセンブラ】**

変更前			変更後		
プログラム		サイズ (byte)	プログラム		サイズ (byte)
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			
合計コード・サイズ		62 bytes	**合計コード・サイズ**		50 bytes

【V850E **での出力アセンブラ】**

変更前			変更後		
プログラム		サイズ (byte)	プログラム		サイズ (byte)
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			
合計コード・サイズ		62 bytes	**合計コード・サイズ**		50 bytes

3. 7　条件比較の結果によって0または1を出力するにはsetf命令を出力させる

if～else文では分岐命令が出力されます。そのため，RISC CPUの最大の特長であるパイプラインが乱れることになります。変数の代入には，if～else文でなく，（条件比較の結果によって0または1を出力する）setf命令が出力されるような記述が可能です。

次にプログラム例を示します。

備考　s, flagは外部変数であるとします。

変数sが100より大きければflag = 1, 100以下ならflag = 0にします。

変更前	変更後

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

【V850 での出力アセンブラ】

変更前			変更後		
プログラム		サイズ (byte)	プログラム		サイズ (byte)
`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:`					
合計コード・サイズ		22 bytes	合計コード・サイズ		18 bytes

【V850E での出力アセンブラ】

変更前			変更後		
プログラム		サイズ (byte)	プログラム		サイズ (byte)
`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:`					
合計コード・サイズ		22 bytes	合計コード・サイズ		18 bytes

3. 8　引き数は4個以内にする

関数コールでは，一般的に引き数を付けて呼び出します。

引き数の最初の4つまではレジスタr6, r7, r8, r9に引き数の値をコピーして関数をコールします。コールされた側は，最初の4つの引き数としてこれらのレジスタの値を使用して処理をします。5つ目からの引き数はスタックに積まれて関数コールが行われます。

コールされた側は5つ目からの引き数をスタックから取得します。引き数が4つから5つに増えただけでレジスタで済んでいた処理にメモリ・アクセスが加わるので関数コール時のオーバヘッドが大きくなります。引き数は4つ以内にすることを念頭に置いて引き数の受け渡しを設計してください。

3. 9　ローカル変数（auto変数）は10個以下，できれば6～7個にする

ローカル変数（auto変数）はレジスタに割り当てられます。

V800コンパイラでは1つの関数内で，作業用レジスタとして10本，レジスタ変数用として10本，合計20本のレジスタを変数用として使用できます（32レジスタ・モード時）。

1つの関数内の処理が時間のかかる処理であれば，ローカル変数をたくさん使うべきです。20本すべてのレジスタをローカル変数用として使用してもかまいません。

あまり時間のかからない関数では，作業用レジスタの10本だけを使用するようにしてください。

レジスタ変数用のレジスタをプッシュ／ポップする時間のオーバヘッドを考慮すると，レジスタ変数の使用は避けた方がよいです。レジスタ変数を使用する／しないは，コンパイラが勝手に決めます。

ローカル変数は6～7個くらいにとどめておくことを推奨します。

残りの3～4本のレジスタは，本当の作業用レジスタとして使用されます。

3. 10　式はあらかじめ整理しておく

式を書いてもコンパイラは正常に式を展開をしてくれますが，式の書き方によっては演算の数を減らすことが可能です。

次にプログラム例を示します。

備考　x, yは外部変数であるとします。

変更前（演算が7個）	変更後（演算が6個）
`y = 7*x*x*x+5*x*x+x;`	`y = x*(7*x*x+5*x+1);`

【V850 での出力アセンブラ】

変更前			変更後		
プログラム		サイズ (byte)	プログラム		サイズ (byte)
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			
合計コード・サイズ		42 bytes	合計コード・サイズ		36 bytes

【V850E での出力アセンブラ】

変更前			変更後		
プログラム		サイズ (byte)	プログラム		サイズ (byte)
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
合計コード・サイズ		32 bytes	合計コード・サイズ		32 bytes

備考　V850Eでは，コンパイラの最適化によって同じコードになります。

3. 11　二のべき乗の乗除算はシフト演算に置き換える

計算式で2のべき乗（2, 4, 8, 16, 32…）での乗除算は，シフト命令に置き換えた方が高速化できます。

通常はＣコンパイラにまかせてもよいのですが，明らかに2のべき乗になる計算は，コーディング時にシフト命令に置き換えてください。

ただし，この方法では負の値を扱ったとき不具合が生じます。正の値を扱うときのみとしてください。正の値を扱うunsigned型ならば問題はありません。

次にプログラム例を示します。

備考　s, t, uは外部変数（unsigned int）であるとします。

変更前	変更後

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

この例はコンパイラの最適化によって同じコードになります。

第4章　変数の定義

変数を定義する際に，注意する点をまとめてあります。

4.1　データの整列

デバイスのアーキテクチャで，メモリ上のデータは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ます）して配置します。

基本の整列条件は，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。

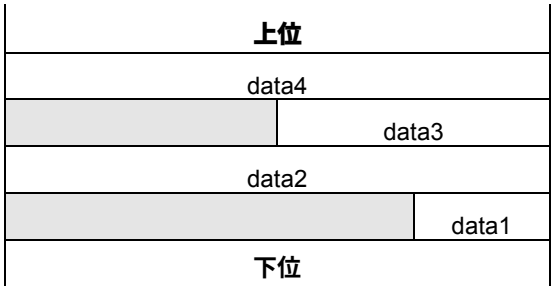
変数を定義する際にはデータ長の長いものから定義してください。

次にプログラム例を示します。

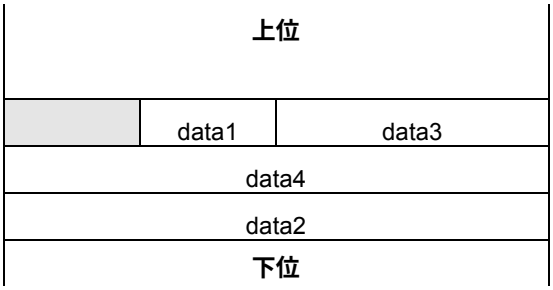
変更前	変更後
struct{ char　data1; long　data2; short　data3; long　data4; }data;	struct{ long　data2; long　data4; short　data3; char　data1; }data;

メモリ配置は次のようになります。

変更前（16 bytes）



変更後（12 bytes）



注意　　はパディング領域です。

4.2　volatile指定

ポートなどの外部I/Oや，割り込み処理中で使用されている外部変数を記述する際は，volatile指定をしてください。volatile指定をしないと，Cコンパイラの最適化により，予想外の動きをすることがあります。

次にプログラム例を示します。

備考　aは外部変数であるとします。

変更前

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

struct bitf {                                                    /* ビットフィールド */
  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    /* 0x100000番地  0bit目 */

void func()
{
         PORT1 = 0xFF;     /* PORT1への書き込み */
         a = PORT1;        /* PORT1からの読み出し */
         PORTb = 1;        /* PORTbのセット */
}
```

変更後

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

struct bitf {                                                    /* ビットフィールド */
  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            /* 0x100000番地  0bit目 */

void func()
{
  PORT1 = 0xFF;    /* PORT1への書き込み */
  a = PORT1;       /* PORT1からの読み出し */
  PORTb = 1;       /* PORTbのセット */
}
```

【V850での出力アセンブラ】

変更前		変更後	
プログラム	サイズ〔byte〕	プログラム	サイズ〔byte〕
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
合計コード・サイズ	20 bytes	合計コード・サイズ	24 bytes

【V850Eでの出力アセンブラ】

変更前		変更後	
プログラム	サイズ〔byte〕	プログラム	サイズ〔byte〕
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
合計コード・サイズ	22 bytes	合計コード・サイズ	24 bytes

4.3　リード・オンリーの変数

リード・オンリー（読み出し専用）の変数は，const変数としてください。

これにより，変数定義がROMに配置されることになり，RAMの変数領域のサイズが小さくなります。

4.4　ファイル間のアラインを減らす

変数定義が，複数のファイルで定義がある場合，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・ホールをなくすことが可能です。

次にプログラム例を示します。

変更前	変更後

-- 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;

【リンク・マップ】

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

変更後				
.sdata		0x00ffe00c	0x00000007	
	.sdata	0x00ffe00c	0x00000007	sub2.o

4.5　フラグをまとめる

数ビット以内のフラグなどがまとめられるときは，ビット・フィールドにまとめます。

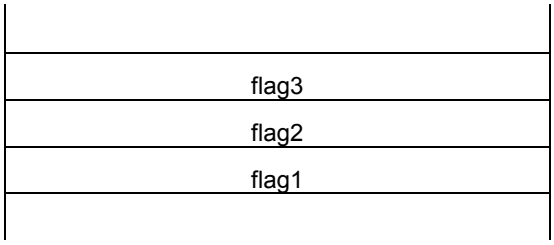
次にプログラム例を示します。

変更前	変更後

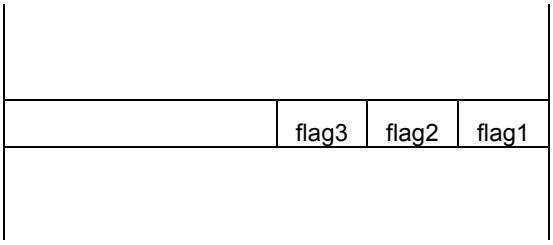
unsigned char flag1;
unsigned char flag2;
unsigned char flag3;
 	
flag1 = 1;
flag2 = 1;
flag3 = 1;
	struct bitf {
	  unsigned char flag1:1;
	  unsigned char flag2:1;
	  unsigned char flag3:1;
	} flags;
	
	flags.flag1 = 1;
	flags.flag2 = 1;
	flags.flag3 = 1;

メモリ配置は次のようになります。

変更前



変更後



4.6　関数のネスト・レベルを少なくする

アルゴリズムの変更により，関数のネストを少なくすると，スタックの使用量が少なくなる可能性があります。インライン展開してもあまり変わらないと考えられるので，アルゴリズムの変更が必要になります。

付　録　総合索引

50音で始まる語句の索引

アルファベットで始まる語句の索引

〔メ　モ〕

お問い合わせ先

【技術的なお問い合わせ先】

ＮＥＣ半導体テクニカルホットライン
（電話：午前 9:00～12:00，午後 1:00～5:00）

電　話　：044−435−9494
FAX　：044−435−9608
E-mail　：info@lsi.nec.co.jp

【営業関係お問い合わせ先】

〔システムLSI〕

システムLSI第一営業事業部

東　京　(03)3798−6106, 6107, 6108, 6155
大　阪　(06)6945−3178, 3200, 3208
名古屋　(052)222−2375
仙　台　(022)267−8740
水　戸　(029)226−1702
広　島　(082)242−5504
鳥　取　(0857)27−5313
松　山　(089)945−4149

システムLSI第二営業事業部

東　京　(03)3798−6110, 6111, 6112, 6151, 6156
名古屋　(052)222−2170, 2190
松　本　(0263)35−1662
前　橋　(027)243−6060
立　川　(042)526−5981
静　岡　(054)254−4794
金　沢　(076)232−7303
福　岡　(092)261−2806

〔汎用デバイス〕

汎用デバイス営業事業部

東　京　(03)3798−6671, 6801
大　阪　(06)6945−3202
名古屋　(052)222−2375, 2170, 2175
仙　台　(022)267−8740
長　野　(0263)35−1662
群　馬　(027)243−6060
水　戸　(029)226−1702
静　岡　(054)254−4794
北　陸　(076)232−7303
鳥　取　(0857)27−5313
九　州　(092)261−2806

【資料の請求先】

上記営業関係お問い合わせ先またはNEC特約店へお申しつけください。

【NECエレクトロンデバイス　ホームページ】

NECエレクトロンデバイスの情報がインターネットでご覧になれます。　URL（アドレス）　http://www.ic.nec.co.jp/

アンケート記入のお願い

お手数ですが，このドキュメントに対するご意見をお寄せください。今後のドキュメント作成の参考にさせていただきます。

［ドキュメント名］CA850 Ver.2.50　アプリケーション・ノート　コーディング・テクニック
（U16076JJ1V0AN00（第1版））

［お名前など］（さしつかえのない範囲で）

御社名（学校名，その他）　（　　）
ご住所　（　　）
お電話番号　（　　）
お仕事の内容　（　　）
お名前　（　　）

1．ご評価（各欄に○をご記入ください）

項　　目	大変良い	良　い	普　通	悪　い	大変悪い
全体の構成					
説明内容					
用語解説					
調べやすさ					
デザイン，字の大きさなど					
その他（　　）					
（　　）					

2．わかりやすい所（第　章，第　章，第　章，第　章，その他　　）
理由［　　］

3．わかりにくい所（第　章，第　章，第　章，第　章，その他　　）
理由［　　］

4．ご意見，ご要望

5．このドキュメントをお届けしたのは
NEC販売員，特約店販売員，その他（　　）

ご協力ありがとうございました。
下記あてにFAXで送信いただくか，最寄りの販売員にコピーをお渡しください。

日本電気(株) NEC エレクトロンデバイス
半導体テクニカルホットライン
FAX：（044）435-9608

2000.6

キリトリ