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。

お客様各位

資料中の「日立製作所」、「日立XX」等名称の株式会社ルネサス テクノロジへの変更について

2003年4月1日を以って三菱電機株式会社及び株式会社日立製作所のマイコン、ロジック、アナログ、ディスクリート半導体、及びDRAMを除くメモリ(フラッシュメモリ・SRAM等)を含む半導体事業は株式会社ルネサス テクノロジに承継されました。従いまして、本資料中には「日立製作所」、「株式会社日立製作所」、「日立半導体」、「日立XX」といった表記が残っておりますが、これらの表記は全て「株式会社ルネサス テクノロジ」に変更されておりますのでご理解の程お願い致します。尚、会社商標・ロゴ・コーポレートステートメント以外の内容については一切変更しておりませんので資料としての内容更新ではありません。

ルネサステクノロジ ホームページ (http://www.renesas.com)

2003年4月1日
株式会社ルネサス テクノロジ
カスタマサポート部

RENESAS

ご注意

安全設計に関するお願い

1. 弊社は品質、信頼性の向上に努めておりますが、半導体製品は故障が発生したり、誤動作する場合があります。弊社の半導体製品の故障又は誤動作によって結果として、人身事故、火災事故、社会的損害などを生じさせないような安全性を考慮した冗長設計、延焼対策設計、誤動作防止設計などの安全設計に十分ご留意ください。

本資料ご利用に際しての留意事項

1. 本資料は、お客様が用途に応じた適切なルネサス テクノロジ製品をご購入いただくための参考資料であり、本資料中に記載の技術情報についてルネサス テクノロジが所有する知的財産権その他の権利の実施、使用を許諾するものではありません。
2. 本資料に記載の製品データ、図、表、プログラム、アルゴリズムその他応用回路例の使用に起因する損害、第三者所有の権利に対する侵害に関し、ルネサス テクノロジは責任を負いません。
3. 本資料に記載の製品データ、図、表、プログラム、アルゴリズムその他全ての情報は本資料発行時点のものであり、ルネサス テクノロジは、予告なしに、本資料に記載した製品または仕様を変更することがあります。ルネサス テクノロジ半導体製品のご購入に当たりましては、事前にルネサス テクノロジ、ルネサス販売または特約店へ最新の情報をご確認頂きますとともに、ルネサス テクノロジホームページ (http://www.renesas.com)などを通じて公開される情報に常にご注意ください。
4. 本資料に記載した情報は、正確を期すため、慎重に制作したものですが万一本資料の記述誤りに起因する損害がお客様に生じた場合には、ルネサス テクノロジはその責任を負いません。
5. 本資料に記載の製品データ、図、表に示す技術的な内容、プログラム及びアルゴリズムを流用する場合は、技術内容、プログラム、アルゴリズム単位で評価するだけでなく、システム全体で十分に評価し、お客様の責任において適用可否を判断してください。ルネサス テクノロジは、適用可否に対する責任は負いません。
6. 本資料に記載された製品は、人命にかかわるような状況の下で使用される機器あるいはシステムに用いられることを目的として設計、製造されたものではありません。本資料に記載の製品を運輸、移動体用、医療用、航空宇宙用、原子力制御用、海底中継用機器あるいはシステムなど、特殊用途へのご利用をご検討の際には、ルネサス テクノロジ、ルネサス販売または特約店へご照会ください。
7. 本資料の転載、複製については、文書によるルネサス テクノロジの事前の承諾が必要です。
8. 本資料に関し詳細についてのお問い合わせ、その他お気付きの点がございましたらルネサス テクノロジ、ルネサス販売または特約店までご照会ください。

User's Manual

H8/300 シリーズ、H8/300L シリーズ

プログラミングガイド

ルネサスシングルチップマイクロコンピュータ

ルネサス エレクトロニクス
www.renesas.com

Rev.1.00　1993.03

ご注意

1. 本書に記載の製品及び技術のうち「外国為替及び外国貿易法」に基づき安全保障貿易管理関連貨物・技術に該当するものを輸出する場合，または国外に持ち出す場合は日本国政府の許可が必要です。
2. 本書に記載された情報の使用に際して，弊社もしくは第三者の特許権，著作権，商標権，その他の知的所有権等の権利に対する保証または実施権の許諾を行うものではありません。また本書に記載された情報を使用した事により第三者の知的所有権等の権利に関わる問題が生じた場合，弊社はその責を負いませんので予めご了承ください。
3. 製品及び製品仕様は予告無く変更する場合がありますので，最終的な設計，ご購入，ご使用に際しましては，事前に最新の製品規格または仕様書をお求めになりご確認ください。
4. 弊社は品質・信頼性の向上に努めておりますが，宇宙，航空，原子力，燃焼制御，運輸，交通，各種安全装置，ライフサポート関連の医療機器等のように，特別な品質・信頼性が要求され，その故障や誤動作が直接人命を脅かしたり，人体に危害を及ぼす恐れのある用途にご使用をお考えのお客様は，事前に弊社営業担当迄ご相談をお願い致します。
5. 設計に際しては，特に最大定格，動作電源電圧範囲，放熱特性，実装条件及びその他諸条件につきましては，弊社保証範囲内でご使用いただきますようお願い致します。
   保証値を越えてご使用された場合の故障及び事故につきましては，弊社はその責を負いません。
   また保証値内のご使用であっても半導体製品について通常予測される故障発生率，故障モードをご考慮の上，弊社製品の動作が原因でご使用機器が人身事故，火災事故，その他の拡大損害を生じないようにフェールセーフ等のシステム上の対策を講じて頂きますようお願い致します。
6. 本製品は耐放射線設計をしておりません。
7. 本書の一部または全部を弊社の文書による承認なしに転載または複製することを堅くお断り致します。
8. 本書をはじめ弊社半導体についてのお問い合わせ，ご相談は弊社営業担当迄お願い致します。

はじめに

H8/300 シリーズ及びH8/300Lシリーズは、日立オリジナルアーキテクチャを採用したH8/300 CPU をコアとしています。H8/300 CPU は、8ビット×16本（または16ビット×8 本）の汎用レジスタと高速動作を指向した簡潔で最適化された命令セットおよび、制御機器などの組み込み用に最適なビット操作命令を豊富に備えており、応用プログラムを容易に作成することができます。

本書は、H8/300シリーズ及びH8/300Lシリーズを初めてご使用になる方に効率良くプログラム開発をして頂く方法を解説しています。また付録として命令セット一覧を添付しましたので、開発時のハンドブックとしてもご活用頂けます。

命令セットおよびアセンブラの詳細については、別冊「H8/300シリーズプログラミングマニュアル」、「H8/300Lシリーズプログラミングマアニュアル」および「H8/300シリーズクロスアセンブラユーザーズマニュアル」を用意しておりますのでご参照下さい。またハードウェアやソフトウェアについてまとめたアプリケーションノートも用意しておりますので、併せてご活用下さい。

なお、ハードウェアの詳細については、各製品別のハードウェアマニュアルをご参照下さい。

目次

1．ソフトウエア開発手順 ――――――――――――――――― 1

2．ＣＰＵの概要 ――――――――――――――――――――― 5
2．1　レジスタ構成 ――――――――――――――――― 5
2．1．1　汎用レジスタ ――――――――――――― 5
2．1．2　コントロールレジスタ ――――――――― 6
2．2　データ構成 ――――――――――――――――― 7
2．3　メモリ空間 ――――――――――――――――― 8
2．4　命令 ――――――――――――――――――― 8
2．5　アドレッシングモード ――――――――――― 9

3．Ｈ８／３００ＣＰＵの特長 ――――――――――――― 11
3．1　メモリマップ ――――――――――――――― 11
3．1．1　８ビット絶対アドレッシングモード ――――― 12
3．1．2　メモリ間接アドレッシングモード ――――― 13
3．2　ビット操作命令 ――――――――――――― 14
3．3　算術演算命令 ――――――――――――― 15
3．4　ブロックデータ転送命令 ――――――――― 16

4．Ｈ８／３００の効果的なプログラミング手法 ――――― 17
4．1　モジュールの構造 ――――――――――― 17
4．2　定数の処理 ――――――――――――― 17
4．3　サブルーチンコール時の引数 ――――――― 18

5．Ｈ８／３００の特長的なプログラミング技法 ――――― 19
5．1　ＭＯＶ命令によるテスト ――――――――― 20
5．2　汎用レジスタ上の定数設定 ――――――― 21
5．3　汎用レジスタのクリア ――――――――― 22
5．4　ビットデータの転送 ――――――――― 23
5．5　ユーザビットの使用方法 ――――――― 24
5．6　ビットテストアンドブランチ ――――――― 25
5．7　ビットチェックアンドブランチ ――――――― 26
5．8　ＲＡＭおよびレジスタのクリア ――――――― 27
5．9　短縮命令の使用（１） ――――――――― 28
5．１０　短縮命令の使用（２） ――――――――― 29
5．１１　メモリ間接アドレッシングの使用 ――――― 30
5．１２　４ステートのＮＯ　ＯＰＥＲＡＴＩＯＮ ――― 31

6．プログラム例 ――――――――――――――――― 32
6．1　ブロック転送 ――――――――――――― 33
6．2　二次元配列 ――――――――――――― 41
6．3　符号付き３２ビット２進数の加算 ――――――― 47

付録
1．命令セット一覧 ――――――――――――――― 54

第１章　ソフトウェア開発手順

以下にH8/300シリーズ及びH8/300Lシリーズのソフトウェア開発手順を示します。なお特に断わらない限り、H8/300シリーズ及びH8/300Lシリーズの両シリーズを合わせてH8/300シリーズで表記します。

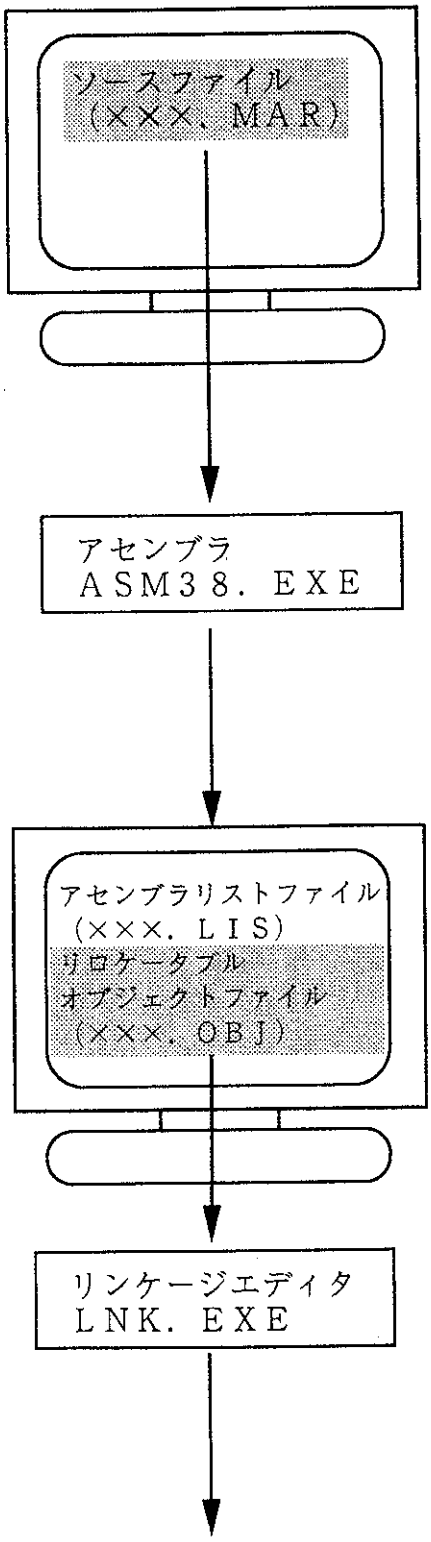
（１）ソースプログラム作成
　リスト例１のようなソースファイルを作成します。

（２）アセンブラの起動
　アセンブラを起動してソースファイルを翻訳します。リンケージエディタ、ライブラリアン、シュミレータデバッガに入力できる形式のオブジェクトファイル（リスト例２）を生成します。

アセンブラの機能概要
（１）マクロ命令の展開
（２）構造化アセンブリの解析
（３）ファイルのインクルード
（４）条件付きアセンブリの判定

（３）リンケージエディタの起動
　リンケージエディタを起動してオブジェクトファイルからロードモジュールを生成します。

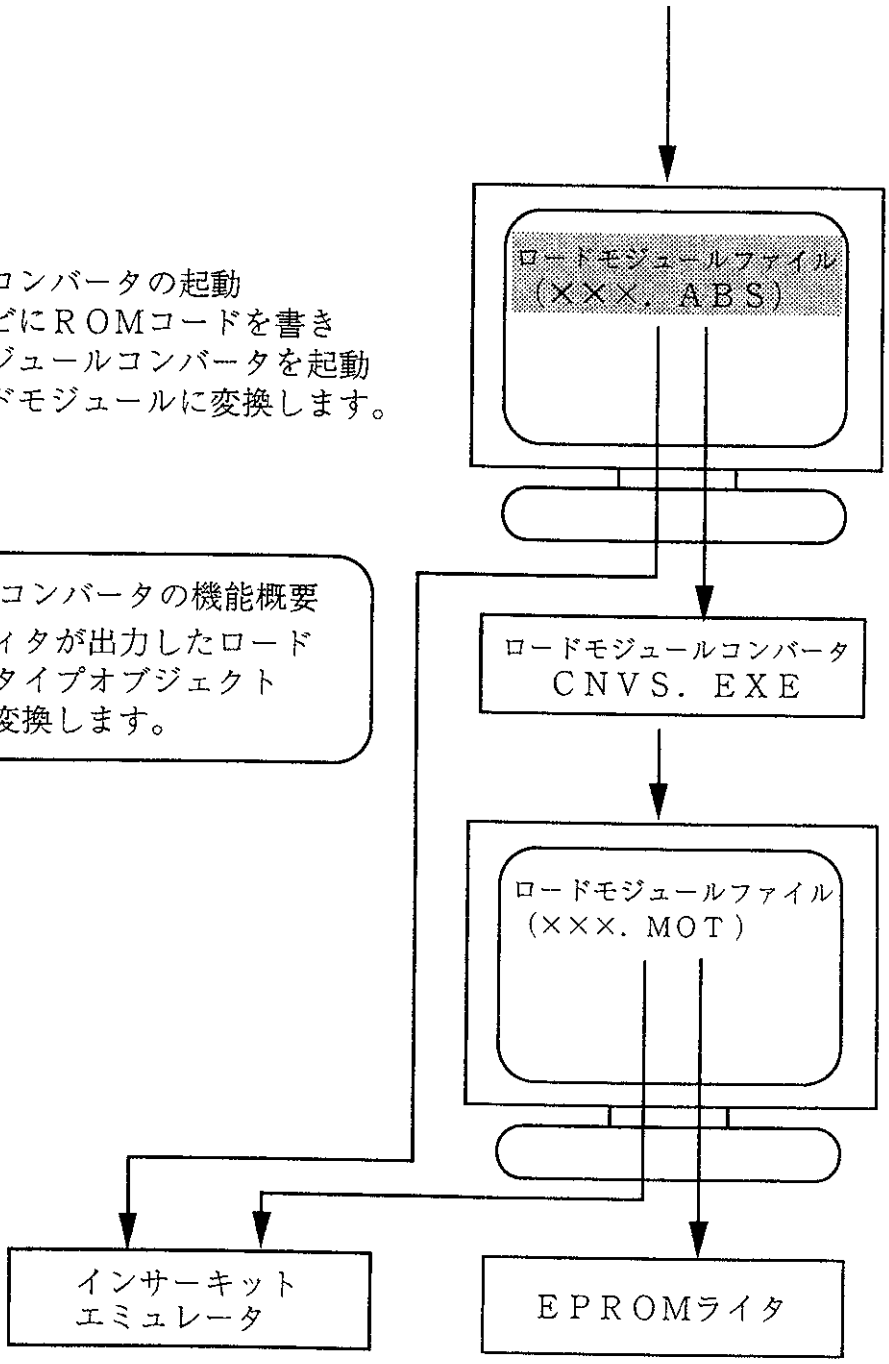
リンケージエディタの機能概要
(1) モジュールの結合機能
(2) アドレスの解決機能
(3) ロードモジュールファイルの再入力機能
(4) マルチリンケージ機能
(5) デバッグ援助機能



・・・この部分のファイルを次のプログラムに入力します。

（4）ロードモジュールコンバータの起動
　ＰＲＯＭライタなどにＲＯＭコードを書き込む時にはロードモジュールコンバータを起動してＳタイプのロードモジュールに変換します。

ロードモジュールコンバータの機能概要
リンケージエディタが出力したロードモジュールをＳタイプオブジェクトフォーマットに変換します。



リスト例１（ソースファイル）

```
; MBSET MACRO DEFINE
        .MACRO MBSET BIT
        BSET.B  #¥BIT&H'0000FFFF,@(¥BIT/H'FFFF)&H'0000FFFF
        .ENDM

; BIT POSITION DEFINE
SR0     .EQU    H'FFB90000      ; ADDRESS=H'FFB9 BIT=H'0000
SR1     .EQU    H'FFB90001      ; ADDRESS=H'FFB9 BIT=H'0001

        MOV.B   #0,R2L

        .INCLUDE        "TEST1.MAR"

        .IF (R1L <EQ> R0L)
        MBSET  SR0
        .ELSE
        MBSET  SR1
        .ENDI

        .END
```

マクロ命令

BEST命令のオペライドを１つにするマクロ命令。

オペライド８桁の内、上位４桁が指定番地を示し

最下位の桁がビット位置を示す。

インクルード文

別に作成したファイル“TEST1.MAR”を挿入する。

アセンブリ言語の構造化記述

R1LとR0Lを比較して、一致していればMBSET SR0を

実行し一致していなければMBSET SR1を実行する。

```
;*****************************
;*      TEST PROGRAM 1       *
;*****************************
;
        MOV.B   R2L,@H'FFB9
```

挿入されるソースファイル

“TEST1.MAR”

リスト例2（アセンブラリストファイル）

行番　アドレス　オブジェクトコード

```
  1                                      1    ; MBSET MACRO DEFINE
  2                                      2             .MACRO MBSET BIT
  3                                      3             BSET.B  #¥BIT&H'0000FFFF,@(¥BIT/H'FFFF)&H'0000FFFF
  4                                      4             .ENDM
  5                                      5
  6                                      6    ; BIT POSITION DEFINE
  7          FFB90000                    7    SR0      .EQU    H'FFB90000       ; ADDRESS=H'FFB9 BIT=H'0000
  8          FFB90001                    8    SR1      .EQU    H'FFB90001       ; ADDRESS=H'FFB9 BIT=H'0001
  9                                      9
 10   0000 FA00                         10             MOV.B   #0,R2L
 11                                     11
 12                                     12             .INCLUDE        "B:TEST1.MAR"
 13                                      1 I1 ;*****************************
 14                                      2 I1 ;*      TEST PROGRAM 1       *
 15                                      3 I1 ;*****************************
 16                                      4 I1 ;
 17   0002 3AB9                          5 I1           MOV.B   R2L,@H'FFB9
 18                                      6 I1 ;
 19                                     13
 20                                     14             .IF (R1L <EQ> R0L)
 21   0004 1C98                          S                       CMP     R1L,R0L
 22   0006 4606                          S                       BNE     _$I00000
 23                                     15             MBSET   SR0
 24   0008 7FB97000                      M                       BSET.B  #SR0&H'0000FFFF,@(SR0/H'FFFF)&H'0000FFFF
 25                                     16             .ELSE
 26   000C 4004                          S                       BRA     _$I00001
 27        0000000E                     S                 _$I00000: .EQU  $
 28                                     17             MBSET   SR1
 29   000E 7FB97010                      M                       BSET.B  #SR1&H'0000FFFF,@(SR1/H'FFFF)&H'0000FFFF
 30                                     18             .ENDI
 31        00000012                     S                 _$I00001: .EQU  $
 32                                     19
 33                                     20             .END
*****TOTAL ERRORS       0
*****TOTAL WARNINGS     0
```

挿入されたファイル “TEST1.MAR”

MBSET SR0をマクロ展開したもの

MBSET SR1をマクロ展開したもの

第２章　CPUの概要

H8/300 CPUは、8ビット×16本(または16ビット×8本)の汎用レジスタを持ち、高速動作を指向した簡潔で最適化された命令セットを備えています。以下にＣＰＵの概要について説明します。

2.1　レジスタ構成

CPUは、図２．１に示すように、8ビット構成の汎用レジスタ16本(R0H/R0L～R7H/R7L)と、コントロールレジスタとして16ビット構成のプログラムカウンタ(PC)および8ビット構成のコンデションコードレジスタ(CCR)を備えています。

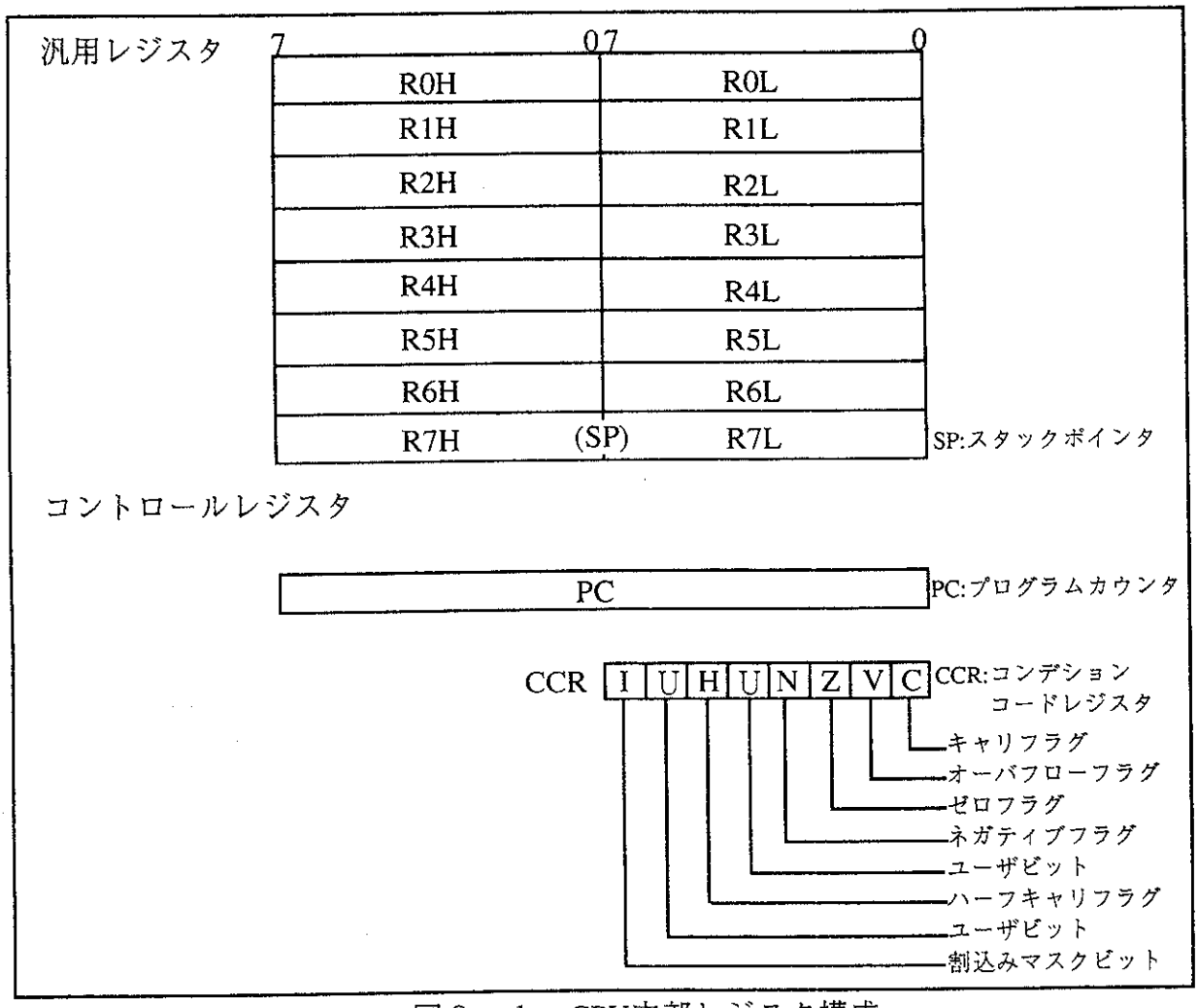


図２．１　CPU内部レジスタ構成

2.1.1　汎用レジスタ

8ビット構成の16本の汎用レジスタは全て同じ機能を持っており、データレジスタ、アドレスレジスタの区別なく使用できます。データレジスタとしては8ビットレジスタとしてだけでなく上位(H)と下位(L)を合わせて16ビットレジスタとして使用することもできます。アドレスレジスタとしては、16ビットレジスタとして使用します。

レジスタR7には、スタックポインタ(SP)としての機能が割り当てられており、例外処理やサブルーチンコール時などに暗黙的に使用されます。

2.1.2 コントロールレジスタ

<プログラムカウンタ(PC)>

CPUが次に実行する命令のアドレスを示す16ビットのカウンタです。

<コンディションコードレジスタ(CCR)>

CPUの内部状態がセットされる8ビット構成のレジスタです。キャリ(C)オーバフロー(V)、ゼロ(Z)、ネガティブ(N)、ハーフキャリ(H)の各フラグ、ユーザ(U)および割込みマスクビット(I)で構成されます。CCRは、CCR操作／転送命令で操作できます。

ビット7(I):割込みマスクビット

このビットが"1"の状態で割込みがマスクされます。例外処理の実行が開始されたとき自動的に"1"にセットされます。

ビット6(U):ユーザビット

ユーザ用ビットです。ソフトウェア(CCR操作／転送命令)でリード／ライトできます。

ビット5(H):ハーフキャリフラグ

ADD.B、ADDX.B、SUB.B、CMP.B命令の実行によって、ビット3にキャリまたはボローが生じたときに"1"にセットされ、生じなかったときは"0"にクリアされます。DAA、DAS命令では暗黙的に使用されます。また、ADD.W、SUB.W、CMP.W命令では、ビット11にキャリまたはボローが生じたときに"1"にセットされ、生じなかったときは"0"にクリアされます。

ビット4(U):ユーザビット

ユーザ用ビットです。ソフトウェア(CCR操作／転送命令)でリード／ライトできます。

ビット3(N):ネガティブフラグ

データの最上位ビットを符号ビットとみなし、そのビットの値を格納します。

ビット2(Z):ゼロフラグ

データがゼロのときに"1"にセットされ、ゼロ以外の値のときは"0"にクリアされます。

ビット1(V):オーバフローフラグ

算術演算命令の実行によって、オーバフローが生じたときに"1"にセットされ、生じなっかったときは"0"クリアされます。

ビット0(C):キャリフラグ

演算の実行によって、キャリが生じたときに"1"にセットされ、生じなかったときは"0"にクリアされます。キャリには、減算結果のボロー、シフト・ローテートによるキャリがあります。また、キャリフラグには、ビットアキュムレータ機能があり、ビット演算やビット転送命令で使用します。

2.2 データ構成

H8/300CPUは、1ビット、4ビットBCD、8ビット(バイト)、および16ビット(ワード)長のデータを扱うことができます。バイトデータは基本的に全ての命令で使用できます。1ビットデータはビット処理命令で、ワードデータは転送命令や演算命令の一部で使用できます。また、4ビットBCDデータは10進補正命令で使用できます。汎用レジスタとメモリのデータフォーマットを図２．２、図２．３に示します。

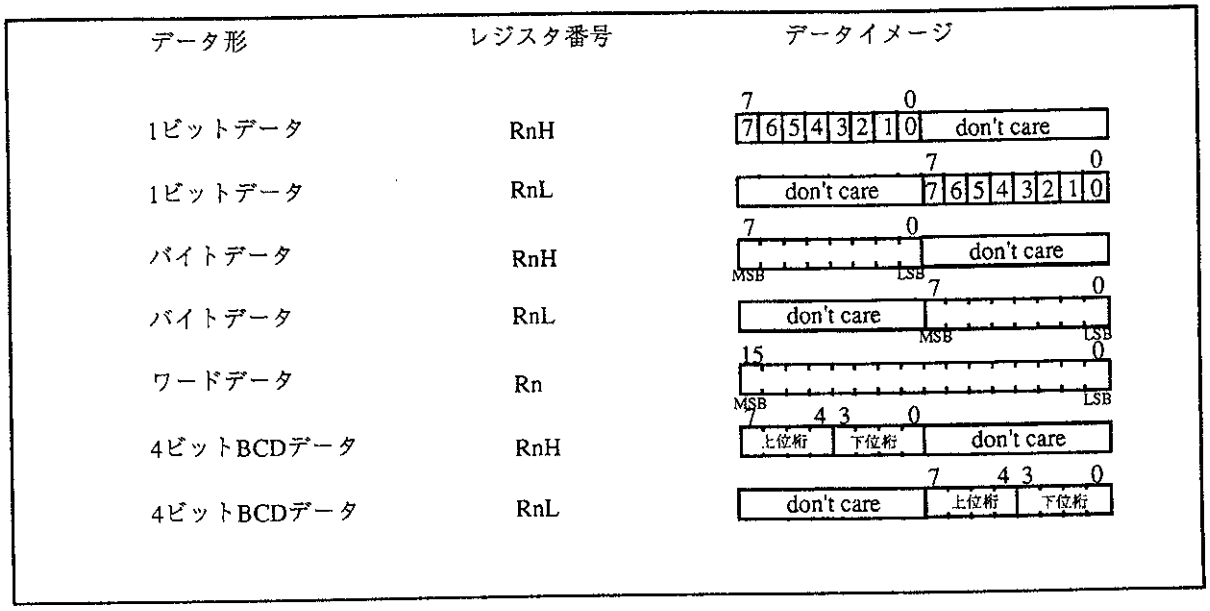
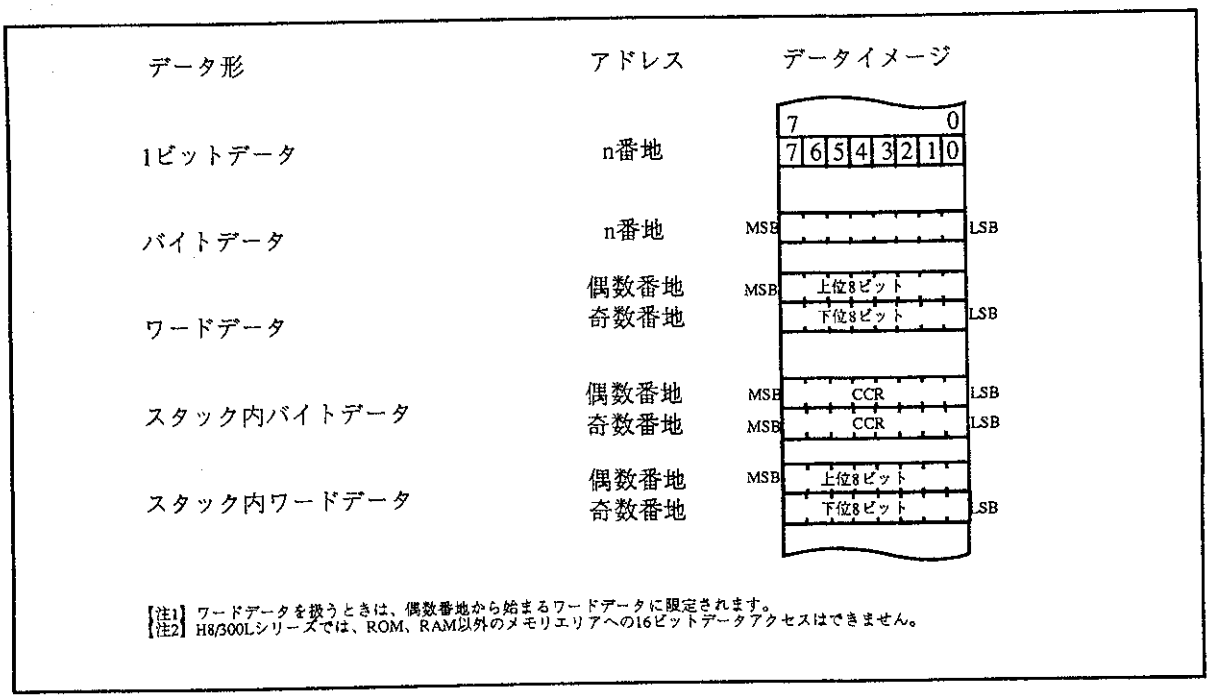


図２．２　汎用レジスタのデータフォーマット



【注1】ワードデータを扱うときは、偶数番地から始まるワードデータに限定されます。
【注2】H8/300Lシリーズでは、ROM、RAM以外のメモリエリアへの16ビットデータアクセスはできません。

図２．３　メモリのデータフォーマット

2.3 メモリ空間

H8/300 CPUがサポートするメモリ空間は、プログラムコードとデータ領域を合わせて64Kバイトです。メモリマップは、H8/300シリーズの各LSI、および各LSIの動作モードによって異なります。詳細は、当該LSIのハードウェアマニュアルを参照してください。

2.4 命令

H8/300CPUの命令は、合計59種類あり、全ての命令の命令長は2バイトまたは4バイトで構成されています。各命令の持つ機能によって、表2．1のように分類されます。各命令については「付録1．命令セット一覧」またはH8/300シリーズプログラミングマニュアルを参照してください。

表2．1　命令の分類

機　能	命　令	種　類
データ転送命令	MOV,POP,PUSH,MOVFPE*1 ,MOVTPE*1	5
算術演算命令	ADD , SUB, ADDX, SUBX, INC, DEC, ADDS, SUBS, DAA, DAS, MULXU, DIVXU, CMP, NEG	14
論理演算命令	AND, OR, XOR, NOT	4
シフト命令	SHAL, SHAR, SHLL, SHLR, ROTL, ROTR, ROTXL,ROTXR	8
ビット操作命令	BSET, BCLR, BNOT, BTST, BAND, BIAND, BOR, BIOR,BXOR,BIXOR, BLD,BILD, BST, BIST	14
分岐命令	Bcc*2, JMP, BSR, JSR, RTS	5
システム制御命令	RTE, SLEEP, LDC, STC, ANDC, ORC , XORC, NOP	8
ブロック転送命令	EEPMOV	1

【注1】H8/300Lシリーズ及びH8/300シリーズの一部のLSIでは、MOVFPE、MOVTPE命令は使用できません。

【注2】Bccは条件分岐命令の総称です。

2.5 アドレッシングモード

CPUは表２．２に示すように8種類のアドレッシングモードをサポートしています。
また、表２．３に実効アドレスの計算方法を示します。

表２．２　アドレッシングモード一覧

No.	アドレッシングモード	記号
1	レジスタ直接	Rn
2	レジスタ間接	@Rn
3	ディスプレースメント付レジスタ間接	@(d:16,Rn)
4	プリデクリメントレジスタ間接 ポストインクリメントレジスタ間接	@-Rn @Rn+
5	イミディエイト	#xx:8 #xx:16
6	絶対アドレス	@aa:8 @aa:16
7	PC相対	@(d:8,PC)
8	メモリ間接	@@aa:8

【注】　データ転送命令では、1から6までのアドレッシングモードが使用できます。

表２．３　実効アドレス(EA:Effective Address)の計算方法

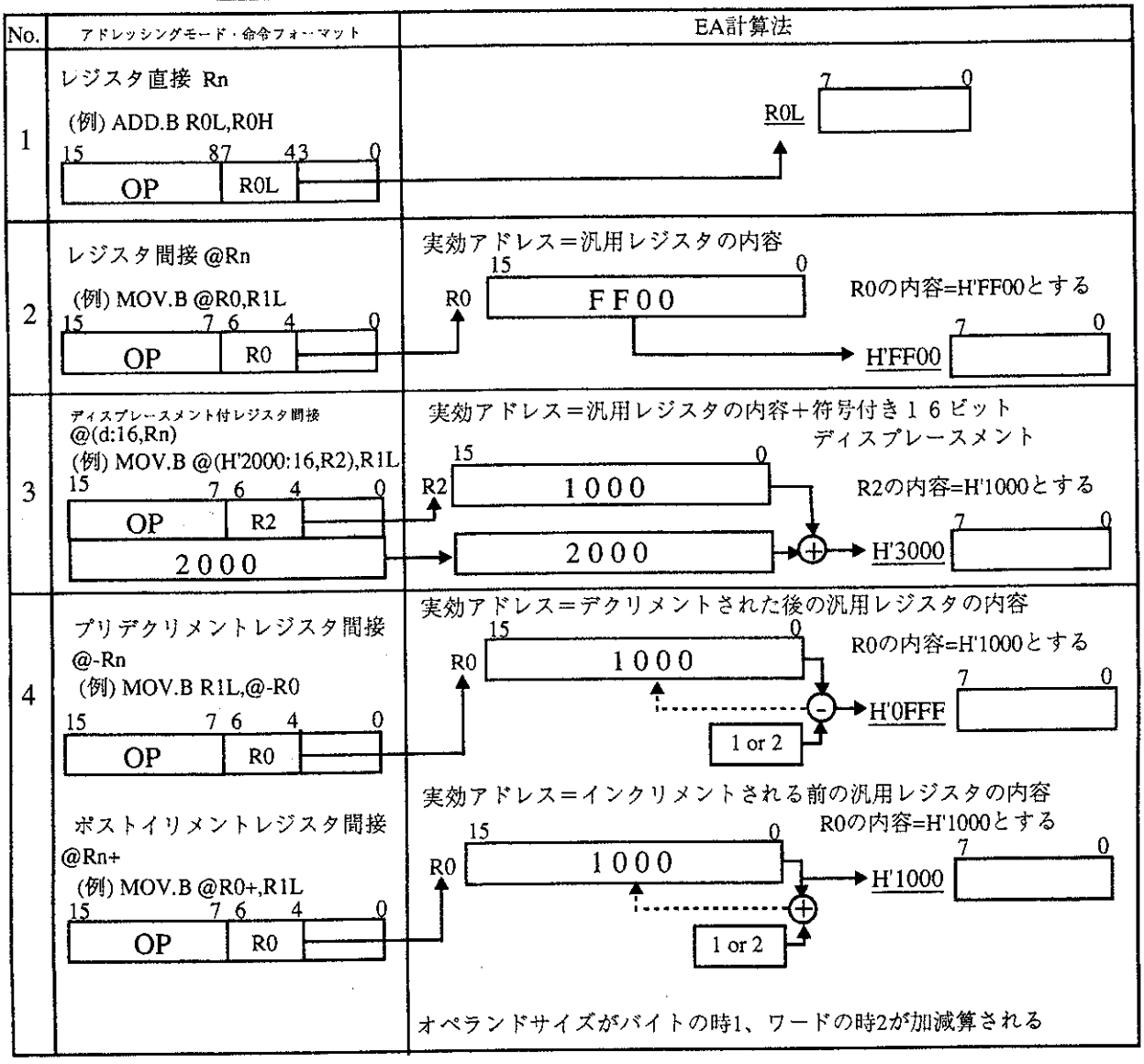
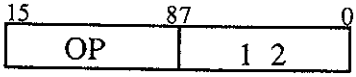
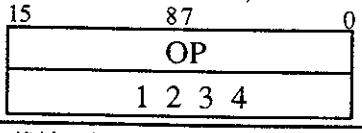
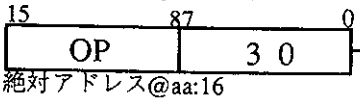
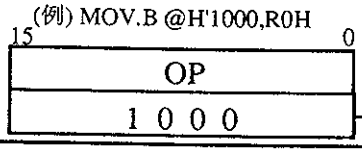
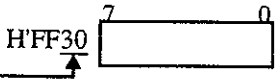
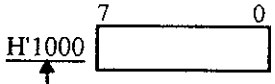
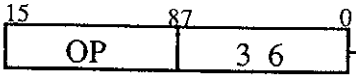
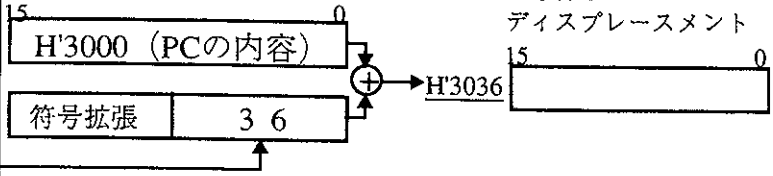
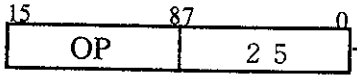
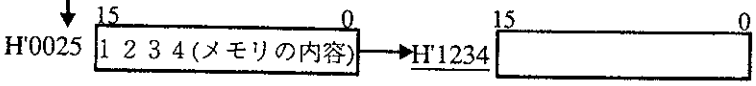


表２．３　実効アドレス(EA:Effective Address)の計算方法

<table>
<tr><th>No.</th><th>アドレッシングモード・命令フォーマット</th><th>EA計算法</th></tr>
<tr><td rowspan="2">5</td><td>イミディエイト#xx:8
(例) MOV.B #H'12,R1L
 
</td><td>オペランドは1バイトデータ</td></tr>
<tr><td>イミディエイト#xx:16
(例) MOV.W #H'1234,R1
 
</td><td>オペランドは2バイトデータ</td></tr>
<tr><td>6</td><td>絶対アドレス@aa:8
(例) MOV.B @H'FF30,R0L
 

絶対アドレス@aa:16
(例) MOV.B @H'1000,R0H
 
</td><td>実効アドレス＝上位アドレスH’ＦＦとオペランドで指定された
下位８ビットのアドレス
 

実効アドレス＝オペランドで指定された１６ビットのアドレス
 
</td></tr>
<tr><td>7</td><td>プログラムカウンタ相対
@(d:8,PC)
(例) BRA H'36
 
</td><td>実効アドレス＝プログラムカウンタの内容＋符号付き８ビット
ディスプレースメント
 
</td></tr>
<tr><td>8</td><td>メモリ間接 @@aa:8
(例) JSR @@25
 
</td><td>実効アドレス＝(上位アドレスH’００とオペランドで指定された下位
８ビットのアドレス）の番地のメモリの内容
 
</td></tr>
</table>

reg:汎用レジスタ、#IMM:イミディエイトデータ、d:ディスプレースメント、aa:絶対アドレス

第３章　H8/300ＣＰＵの特長

この章ではH8/300ＣＰＵの命令およびアドレッシングモードで特長的なものについて説明します。

3.1　メモリマップ

図３．１にH8/300シリーズのメモリマップを示します。H8/300シリーズには８ビット絶対アドレスエリアと８ビットメモリ間接アドレスエリアという２種類の特殊エリアを持っています。８ビット絶対アドレスエリアではデータ転送命令が２バイトで実行できます。８ビットメモリ間接アドレスエリアではメモリ間接アドレッシングでの実効アドレスを格納します。この２種類の特殊エリアを有効的に活用することが最適化プログラミングの重要なポイントとなります。

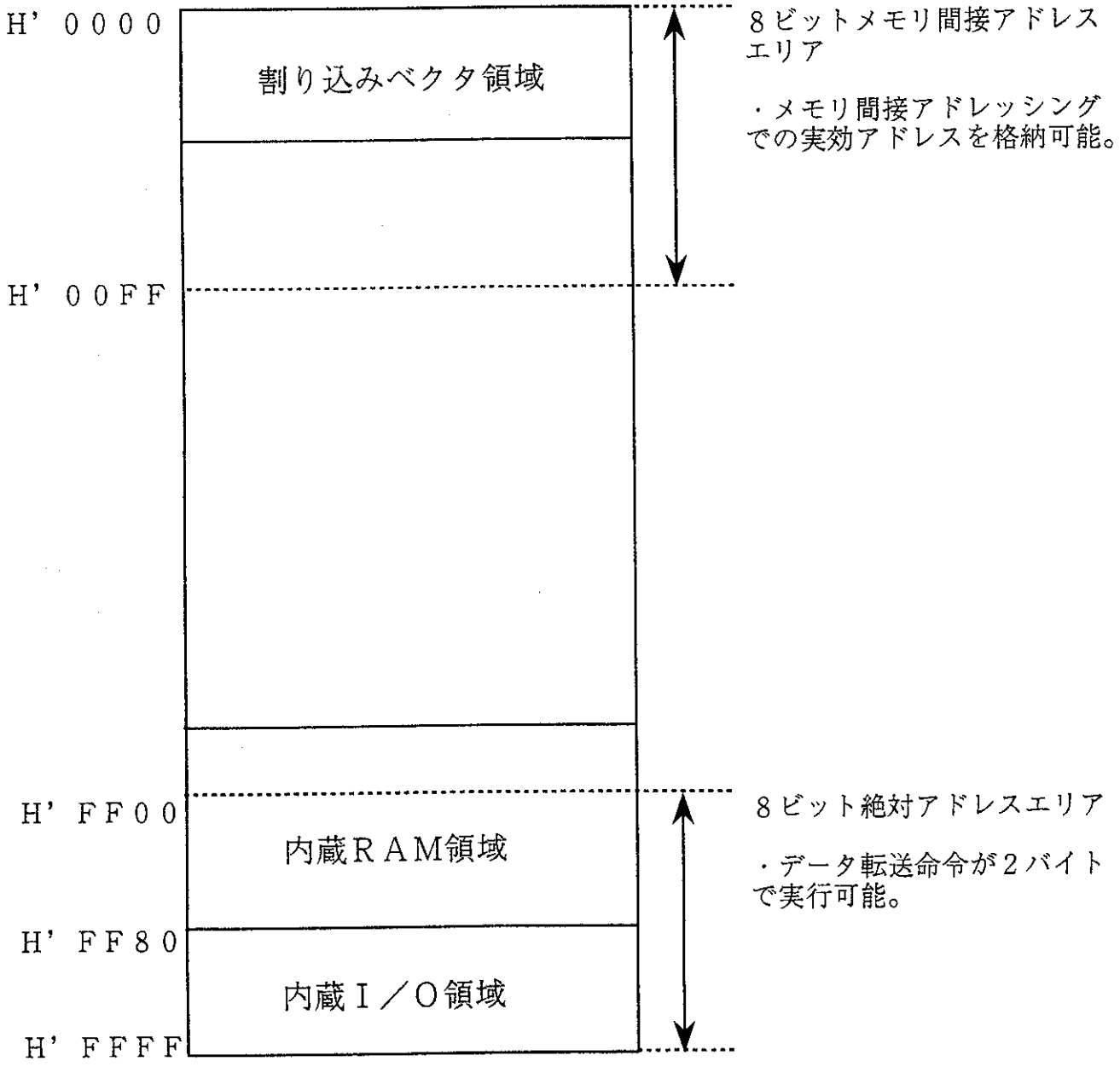


図３．１　メモリマップ

3.1.1　８ビット絶対アドレッシングモード

絶対アドレッシングモードは命令コード中に含まれる絶対アドレスでメモリ上の実効アドレスを指定します。表３．１に示すように絶対アドレッシングには８ビット絶対アドレス（@ａａ：８）、１６ビット絶対アドレス（@ａａ：１６）の２種類があります。８ビット絶対アドレスはバイト単位のデータ転送命令（ＭＯＶ．Ｂ）、ビット操作命令で使用できます。１６ビット絶対アドレスはデータ転送命令（ＭＯＶ．Ｂ、ＭＯＶ．Ｗ）、無条件分岐命令（ＪＭＰ）およびサブルーチン分岐命令（ＪＳＲ）の各命令で使用できます。８ビット絶対アドレスの場合、上位８ビットは全て’１’（Ｈ’ＦＦ）になります。従ってアクセス範囲はＨ’ＦＦ００～Ｈ’ＦＦＦＦ番地となります。

バイト単位のデータ転送命令（ＭＯＶ．Ｂ）を使用する場合には８ビット絶対アドレスを使用した方が１６ビット絶対アドレスよりも使用バイト数、実行ステート数が少なくなります。

表３．１　ＭＯＶ．Ｂ命令の構成

ＭＯＶ．Ｂ　Ｒｓ，＜ＥＡｄ＞

アドレッシングモード	オペランド形式	命令の構成				使用バイト数	実行ステート数
		第１バイト	第２バイト	第３バイト	第４バイト		
８ビット絶対アドレス	@ａａ：８	３ | rs	ａｂｓ			２	４
１６ビット絶対アドレス	@ａａ：１６	６ | Ａ	８ | rs	ａｂｓ		４	６

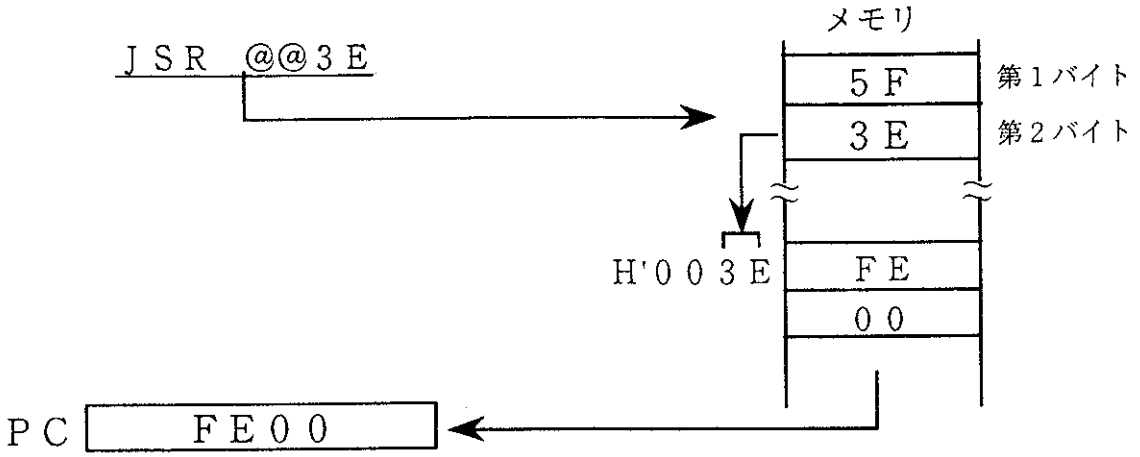
abs：絶対アドレス
Rs：ソース側の汎用レジスタ
rs：レジスタ番号
<EAd>：ディスティネーション
　　オペランド

3.1.2　メモリ間接アドレッシング

無条件分岐命令（ＪＭＰ）およびサブルーチン分岐命令（ＪＳＲ）で使用します。命令コードの第２バイトに含まれる８ビット絶対アドレスでメモリ上の実効アドレスを指定し、この内容を分岐アドレスとして分岐します。この実効アドレスの上位８ビットは全て’０’（Ｈ’００）となりますので、分岐アドレスを格納できるのはＨ’００００～Ｈ’００ＦＦ番地です。

このアドレッシングモードはプログラム開発段階でＲＯＭサイズを小さくする時に用いられます。１６ビット絶対アドレッシングで書かれた命令をメモリ間接アドレッシングに置き換えると、２バイト節約することができます。以下に使用例とＪＳＲの命令の構成を示します。

使用例（Ｈ’ＦＥ００番地を先頭番地とするサブルーチンをコールする。）



ＪＳＲ　＜ＥＡ＞　　表３．２　ＪＳＲの命令の構成

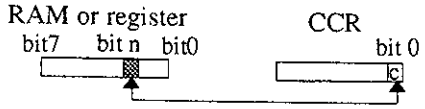
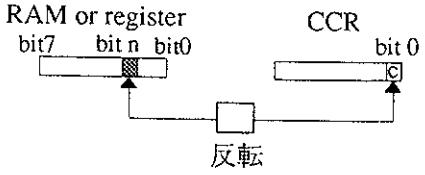
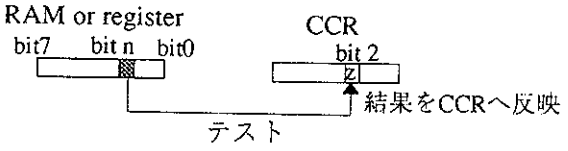
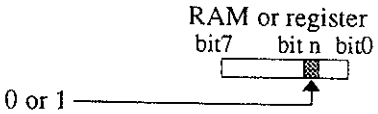
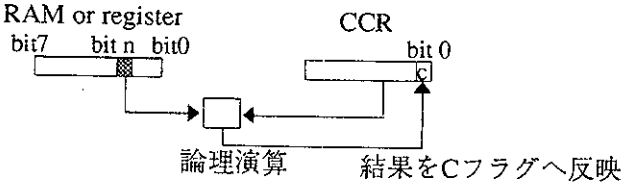
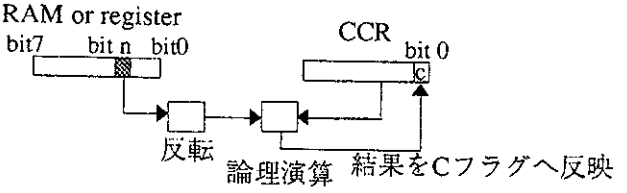
アドレッシングモード	オペランド形式	命令の構成				使用バイト数	実行ステート数
		第１バイト	第２バイト	第３バイト	第４バイト		
メモリ間接	@@ａａ：８	５ ｜ Ｆ	ａｂｓ			２	８
１６ビット絶対アドレス	@ａａ：１６	５ ｜ Ｅ	０ ｜ ０	ａｂｓ		４	８

abs：絶対アドレス
<EA>：ディスティネーション
オペランド

3.2　ビット操作命令

H8/300ＣＰＵは表３．３に示すように豊富なビット操作命令を持っています。従来の８ビットマイコンが持っているビットテスト命令やビットセット／クリア命令の他にビット転送命令を４種類、ビット演算命令を７種類持っており、複雑なビット操作処理を容易に実現することができます。

表３．３　ビット操作処理説明

No.	処理	適用命令	動作説明
1	ビット転送	ＢＬＤ ＢＩＬＤ ＢＳＴ ＢＩＳＴ	(a)BLD等の場合  (b)BILD等の場合 
2	ビットテスト	ＢＴＳＴ	
3	ビットセット／ ビットクリア	ＢＳＥＴ ＢＣＬＲ	
4	論理演算	ＢＮＯＴ ＢＡＮＤ ＢＩＡＮＤ ＢＯＲ ＢＩＯＲ ＢＸＯＲ ＢＩＸＯＲ	(a)BAND等の場合  (b)BIAND等の場合 

3.3　算術演算命令

H8/300ＣＰＵの汎用レジスタは１６ビットレジスタとしても使用できます。従って、１６ビットの加減算、８ビット×８ビットの乗算及び１６ビット÷８ビットの除算を１命令で高速に実行できます。表３．４に算術演算命令の処理性能を示します。

表３．４　算術演算処理性能

算術演算	適用命令	命令長	実行ステート数
加算　16-bit +16-bit	ＡＤＤ， ＡＤＤＳ	２バイト	２ステート
減算　16-bit -16-bit	ＳＵＢ， ＳＵＢＳ	2	2
乗算　8-bit x 8-bit	ＭＵＬＸＵ	2	１４
除算　16-bit ÷8-bit	ＤＩＶＸＵ	2	１４

3.4　ブロックデータ転送命令

H8/300ＣＰＵはブロックデータ転送命令（ＥＥＰＭＯＶ）を持っており、１命令で最大２５５バイトの連続データを転送することができます。この命令実行中に割り込み要求が発生してもＣＰＵは受け付けません。

使用例１．H'0200で始まる100バイトのデータをH'FF00で始まるエリアへ転送します。表３．５にＥＥＰＭＯＶを使用した場合と使用しない場合のブロック転送を比較します。

表３．５　ブロック転送の比較

		ＥＥＰＭＯＶ使用時	ＥＥＰＭＯＶ未使用時
プログラム		MOV.W #H'0200, R5 MOV.W #H'FF00, R6 MOV.B #100, R4L EEPMOV	MOV.W #H'0200, R5 MOV.W #H'FF64, R6 MOV.W #H'FF00, R3 BR1 MOV.W @R5+, R4 MOV.W R4, @-R6 CMP.W R3, R6 BNE BR1
結果	ステップ数	４ステップ	7ステップ
	ＲＯＭ使用量	14バイト	20バイト
	実行ステート数	418ステート	918ステート

使用例２．転送データを２回に分けて転送する。２回目の転送時転送元アドレス（Ｒ５）と転送先アドレス（Ｒ６）は設定されており、転送バイト数（Ｒ４Ｌ）の設定のみでＥＥＰＭＯＶを発行できます。

表３．６　ブロック転送のプログラム例

プログラム		MOV.W #H'0200, R5 MOV.W #H'FF00, R6 MOV.B #50, R4L EEPMOV MOV.B #50, R4L EEPMOV
結果	ステップ数	６ステップ
	ＲＯＭ使用量	20バイト
	実行ステート数	430ステート

第４章　Ｈ８／３００の効果的なプログミング手法

本章では豊富な汎用レジスタを持ったＣＰＵに適したプログラミング手法について説明します。

4.1　モジュールの構造

H8/300ＣＰＵは汎用レジスタ間での演算を原則として命令体系が作られています。したがって、モジュール毎のプログラムは図４．１のように、

（１）演算に使用するＲＡＭ上のデータを全て汎用レジスタにロードする。
（２）汎用レジスタ間の演算を行なう。
（３）汎用レジスタからＲＡＭにデータを格納する。

という構造になります。データが多い場合には使用頻度の高いものから順に汎用レジスタにロードします。この構造にすると、不必要なデータの転送を避けることができます。つまり、同じデータを何回もロードしたり、格納したりすることはＲＯＭの使用量の増加となります。

例　（n個のデータの演算を行い、その結果をメモリへ帰します）

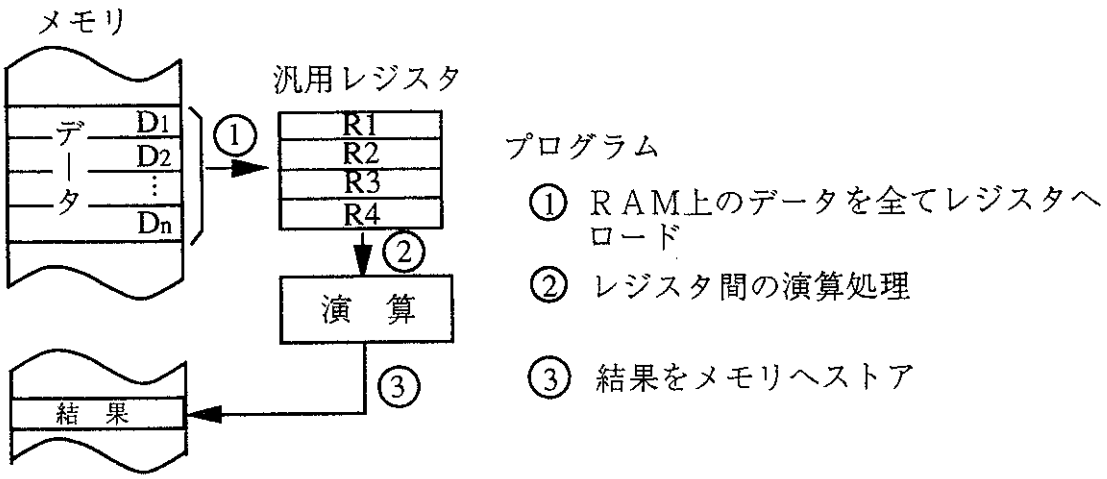


図４．１　モジュールの構造

4.2　定数の処理

頻繁に使われる定数は汎用レジスタ上に常駐させることでＲＯＭ使用量を低減できます。たとえばプログラム中にＲＡＭの内容をクリアする（Ｈ’００にする）動作を多く使うプログラムではＲ０ＬにＨ’００を常駐させます。すると、従来ＲＡＭクリアには
（１）Ｈ’００を汎用レジスタにロードする。
（２）汎用レジスタの内容をＲＡＭに転送する。
と言うように２ステップ必要でしたが、Ｈ’００をＲ０Ｌに常駐させると上記（１）を省くことができます。

使用例（ＷＯＲＫをクリアする。）

```
MOV.B #H'00, R0L          ──────►  MOV.B R0L, @WORK
MOV.B R0L, @WORK
```

図４．２　定数処理のプログラム例

4.3　サブルーチンコール時の引数

サブルーチン分岐命令（ＪＳＲ）等でサブルーチンコールをする場合サブルーチンに引き渡される引数は、１種類とは限りません。また１種類だとしても、サブルーチンの先頭でそのデータを処理するとは限りません。演算処理用レジスタを１本しか持たないＣＰＵでは引数をＲＡＭに格納してサブルーチンコールを行なう方法が一般的ですが、H8/300シリーズでは16本の汎用レジスタに、引数を置いたままサブルーチンをコールすることができます。従って、豊富なレジスタを使用して、サブルーチンの引数を引き渡すことにより、RAM・レジスタ間のロード/ストアを省略できます。

使用例（引数が２個の場合のサブルーチンコール）

```
        MOV.B   #H'01, R0L
        MOV.B   R0L, @RAM1
        MOV.B   #H'02, R0H
        MOV.B   R0H, @RAM2
        JSR     @SUB
        ------
SUB     MOV.B   @RAM1, R0L
        MOV.B   @RAM2, R0H
        SUB.B   R0L, R0H
        ------
```

→

```
        MOV.B   #H'01, R0L
        MOV.B   #H'02, R0H
        JSR     @SUB
        ------
SUB     SUB.B   R0L, R0H
        ------
```

図４．３　サブルーチンコールのプログラム例

第５章　Ｈ８／３００の特長的なプログラミング技法

この章ではＨ８／３００の特長を利用したプログラミング技法について説明します。

ＮＯ	内容
1	ＭＯＶ命令によるテスト
2	汎用レジスタ上の定数設定
3	汎用レジスタのクリア
4	ビットデータの転送
5	ユーザビットの使用方法
6	ビットテストアンドブランチ
7	ビットチェックアンドブランチ
8	ＲＡＭおよびレジスタのクリア
9	短縮命令の使用（１）
１０	短縮命令の使用（２）
１１	メモリ間接アドレッシングの使用
１２	4ステートのNO OPERATION

5．1　MOV命令によるテスト

適用命令	
○	転送命令
	演算命令
	ビット操作命令
	シフト命令
	分岐命令
	制御命令
	その他

説明

MOV命令を実行するとCCRのN、Zフラグが変化します。この直後に分岐命令を置いてその結果を利用します。

使用例

RAMエリアのデータ@DATAが0の場合分岐します。

(現行プログラム)

```
MOV.B @DATA,R1L
CMP.B #0,R1L
BEQ   BR1
```

(変更後のプログラム)

```
MOV.B @DATA,R1L
BEQ   BR1
```

ステップ数	バイト数	ステップ数	バイト数
3 Step	6 Byte	2 Step	4 Byte

補足

MOV．Wでも同様です。　また、Nフラグも変化するので’BMI’を使用して分岐することもできます。

5．2　汎用レジスタ上の定数設定

適用命令	
○	転送命令
	演算命令
	ビット操作命令
	シフト命令
	分岐命令
	制御命令
	その他

説明

汎用レジスタの上位側（Ｒ０Ｈ等）に定数H'００を設定しておくと、ディスプレイスメント付レジスタ間接アドレッシングを使用する時にステップ数が節約できます。

使用例

ＴＯＰ番地からＭＯＤＥ番目のデータを汎用レジスタにロードします。

（現行プログラム）

```
MOV.B  #00,R0H
MOV.B  #MODE,R0L
MOV.B  @(TOP,R0),R1L
```

（Ｒ０Ｈ＝Ｈ’００と設定した場合）

```
MOV.B  #MODE,R0L
MOV.B  @(TOP,R0),R1L
```

ステップ数	バイト数	ステップ数	バイト数
3 Ｓｔｅｐ	8 Ｂｙｔｅ	2 Ｓｔｅｐ	6 Ｂｙｔｅ

補足

このアドレッシングモードにおいて、アドレス演算を行なう汎用レジスタは１６ビットデータのみ扱うことができます。

５．３　汎用レジスタのクリア

適用命令	
○	転送命令
	演算命令
	ビット操作命令
	シフト命令
	分岐命令
	制御命令
	その他

説明

汎用レジスタ１６ビットを０クリアする時は自分自身との減算を行います。

使用例

Ｒ０をクリアします。

（現行プログラム）

```
MOV.W #0,R0
```

（変更後のプログラム）

```
SUB.W R0,R0
```

ステップ数	バイト数	ステップ数	バイト数
1 Ｓｔｅｐ	4 Ｂｙｔｅ	1 Ｓｔｅｐ	2 Ｂｙｔｅ

補足

この時ＣＣＲは変化します。

５．４　ビットデータの転送

適用命令	
	転送命令
	演算命令
○	ビット操作命令
	シフト命令
	分岐命令
	制御命令
	その他

説明

任意のビットデータを転送する場合はＣフラグを利用した
ビット操作命令を使用します。

使用例

@ＷＫ１のｂｉｔ２のデータを@ＷＫ２のｂｉｔ６に転送します。

（現行プログラム）

```
      BTST  #2,@WK1
      BEQ   L1
      BSET  #6,@WK2
      BRA   L2
L1:   BCLR  #6,@WK2
L2:
```

（変更後のプログラム）

```
      BLD   #2,@WK1
      BST   #6,@WK2
```

ステップ数	バイト数	ステップ数	バイト数
5 Ｓｔｅｐ	16 Ｂｙｔｅ	2 Ｓｔｅｐ	8 Ｂｙｔｅ

補足

5．5　ユーザビットの使用方法

適用命令	
	転送命令
	演算命令
○	ビット操作命令
	シフト命令
	分岐命令
	制御命令
	その他

説明

ＣＣＲ内にあるユーザビット（２ビット）はＲＡＭエリアのビットと較べて以下の特長があります。

（１）ビットのセット、クリアが２バイトで可能です。

（２）例外処理時にこのビットはスタックされます。つまり、各例外処理ルーチン内でワークエリアとして使用できます。

使用例

ＣＣＲのビット６をクリアします。

（ＲＡＭ上のビットの場合）

```
BCLR  #6,@RAM
```

（ユーザビットの場合）

```
ANDC  #BF,CCR
```

ステップ数	バイト数	ステップ数	バイト数
1 Ｓｔｅｐ	4 Ｂｙｔｅ	1 Ｓｔｅｐ	2 Ｂｙｔｅ

補足

ビットをセットする時は’ＯＲＣ’を使用します。

例．ＯＲＣ　＃Ｈ’４０，ＣＣＲ

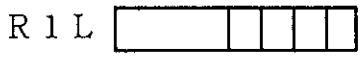
5．6　ビットテストアンドブランチ

適用命令	
	転送命令
	演算命令
○	ビット操作命令
	シフト命令
○	分岐命令
	制御命令
	その他

説明

複数のビット（4 bit以下）の状態により分岐を行う場合、判断するビットデータをレジスタＲｎの下位４ビットに設定しておき、Ｒｎを ＣＣＲへロードし分岐命令で分岐します。

使用例



Ｒ１Ｌの下位４ビットをテストして各々ブランチ先にブランチします。

（現行プログラム）

```
BTST  #0,R1L
BNE   BR1
BTST  #1,R1L
BNE   BR2
BTST  #2,R1L
BNE   BR3
BTST  #3,R1L
BNE   BR4
BRA   BR5
```

（変更後のプログラム）

```
LDC  R1L,CCR
BCS  BR1
BVS  BR2
BEQ  BR3
BMI  BR4
BRA  BR5
```

ステップ数	バイト数	ステップ数	バイト数
9 Ｓｔｅｐ	１８ Ｂｙｔｅ	6 Ｓｔｅｐ	１２ Ｂｙｔｅ

補足

ＣＣＲのＩビットにあたるbit 7は、常にＩビットと同じ状態にしておく必要があります。

５．７　ビットチェック＆ブランチ

適用命令
転送命令
演算命令
○ ビット操作命令
シフト命令
○ 分岐命令
制御命令
その他

説明

bitA	bitB	結果
0	0	BR1
0	1	BR2
1	0	BR2
1	1	BR1

任意の２ビットをテストした結果が左表のような関係にあるとき、ＢＸＯＲ命令を使用します。

使用例

@ＷＫ１のｂｉｔ２と@ＷＫ２のｂｉｔ６が上表のような関係にある場合の処理をします。

（現行プログラム）

```
      BTST  #2,@WK1
      BNE   BR3
      BTST  #6,@WK2
      BNE   BR2
      BRA   BR1
BR3:  BTST  #6,@WK2
      BNE   BR1
BR2:
```

（変更後のプログラム）

```
      BLD   #2,@WK1
      BXOR  #6,@WK2
      BCC   BR1
BR2:
```

ステップ数	バイト数	ステップ数	バイト数
7 Ｓｔｅｐ	２０ Ｂｙｔｅ	3 Ｓｔｅｐ	１０ Ｂｙｔｅ

補足

ＢＡＮＤ，ＢＯＲも同様に使用できます。

5．8　RAMおよびレジスタのクリア

適用命令	
○	転送命令
	演算命令
	ビット操作命令
	シフト命令
	分岐命令
	制御命令
	その他

説明

RAMおよびレジスタをクリアする処理が多数ある場合には汎用レジスタRnに定数H'0000をセットしておき、そのレジスタの内容をＲＡＭまたはレジスタにライトします。

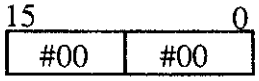
使用例

あるメモリ@DATA1(16bit)をクリアします。

（現行プログラム）

```
MOV.W #00,R1
MOV.W R1,@DATA1
```

（変更後のプログラム）



Ｒ０は常に＃００をセットしておきます。

```
MOV.W R0,@DATA1
```

ステップ数	バイト数	ステップ数	バイト数
2 Ｓｔｅｐ	8 Ｂｙｔｅ	1 Ｓｔｅｐ	4 Ｂｙｔｅ

補足

5．9　短縮命令の使用（1）

適用命令	
	転送命令
	演算命令
	ビット操作命令
	シフト命令
	分岐命令
	制御命令
○	その他

説明

メモリアドレスH’ＦＦ００からH’ＦＦＦＦに対して
絶対アドレッシングを使用する時は短縮命令を使用できます。

使用例　@ＤＡＴＡを絶対アドレッシングします。

（現行プログラム）

@ＤＡＴＡ＝H’ＦＥ００と設定

```
MOV.B @DATA,R0L
```

（変更後のプログラム）

@ＤＡＴＡ＝H’ＦＦ００と設定

```
MOV.B @DATA,R0L
```

ステップ数	バイト数	ステップ数	バイト数
1 Ｓｔｅｐ	4 Ｂｙｔｅ	1 Ｓｔｅｐ	2 Ｂｙｔｅ

補足

5．10　短縮命令の使用（2）

適用命令	
	転送命令
	演算命令
○	ビット操作命令
	シフト命令
	分岐命令
	制御命令
	その他

説明

ビットデータが多いＲＡＭはH’ＦＦ００以降に置き、短縮命令を使用します。

使用例

FLGの下位２ビットをセットします。

（現行プログラム）

@ＦＬＧ＝H’ＦＥ００の時

```
MOV.W #H'FE00,R0
BSET  #FL0,@R0
BSET  #FL1,@R0
```

ステップ数	バイト数
3 Ｓｔｅｐ	12 Ｂｙｔｅ

（変更後のプログラム）

@ＦＬＧ＝H’ＦＦ００の時

```
BSET  #FL0,@FLG
BSET  #FL1,@FLG
```

ステップ数	バイト数
2 Ｓｔｅｐ	8 Ｂｙｔｅ

補足

H’ＦＦ００番地以降のＲＡＭに１ビットという割付ができれば’ＢＴＳＴ’の代わりに’ＭＯＶ’を使用して２バイトでビット判定ができます。また、最上位ビットのビットテストもＮフラグをチェックすることで同様に’ＭＯＶ’で判定することができます。

５．１１　メモリ間接アドレッシングの使用

適用命令	
	転送命令
	演算命令
	ビット操作命令
	シフト命令
○	分岐命令
	制御命令
	その他

説明

ＪＳＲ、ＪＭＰ命令を用いる場合ジャンプテーブルをH’００００からH’００ＦＦ番地に用意しておきメモリ間接アドレッシングで上記命令を実行します。

使用例

ＡＡＡ番地にジャンプします。（但しH’００ＦＥ番地およびH’００ＦＦ番地にＡＡＡが格納されていると仮定します。）

（現行プログラム）

```
JMP  @AAA
```

（変更後のプログラム）

```
JMP  @@H'FE
```

ステップ数	バイト数	ステップ数	バイト数
1 Ｓｔｅｐ	4 Ｂｙｔｅ	1 Ｓｔｅｐ	2 Ｂｙｔｅ

補足

５．１２　４ステートのNO OPERATION

適用命令	
	転送命令
	演算命令
	ビット操作命令
	シフト命令
	分岐命令
	制御命令
○	その他

説明

H8/300シリーズにはＮＯＰ（NO OPERATION）命令があり、使用ＲＯＭ２バイト、処理時間２ステートで実行します。処理時間４ステートのＮＯＰが必要な場合には、分岐命令ＢＲＡを用いた「ＢＲＡ　＄＋Ｈ’０２」で実現できます。（ここで　’＄’は現在ＰＣが示している番地を表します。）

使用例

８ステートの待ち時間を作ります。

（現行プログラム）

```
NOP
NOP
NOP
NOP
```

（変更後のプログラム）

```
BRA $+H'02
BRA $+H'02
```

ステップ数	バイト数	ステップ数	バイト数
4 Ｓｔｅｐ	8 Ｂｙｔｅ	2 Ｓｔｅｐ	4 Ｂｙｔｅ

補足

ＢＲＮでも同様。

第６章　プログラム例

この章では第５章で説明したプログラミング技法の応用例を示します。

1. ブロック転送
2. 二次元配列
3. 符号付き32ビット2進数の加算

6.1.ブロック転送	ﾗﾍﾞﾙ名	MOVE2

機　　能

(1)　データメモリエリアにあるブロックデータを別のメモリエリアに転送します。
(2)　転送元、転送先のメモリエリアは任意に設定できます。
(3)　転送元と転送先のデータメモリエリアが重なっても転送可能です。
(4)　EEPMOV命令（ブロック転送命令）のアプリケーションソフト例です。

引　　数

内　　容		格納場所	ﾃﾞｰﾀ長（ﾊﾞｲﾄ）
入　力	転送バイトの数	R4L	1
	転送元の先頭アドレス	R5	2
	転送先の先頭アドレス	R6	2
出　力	エラーの有無	Cフラグ（CCR）	

内部レジスタ変化およびフラグ変化

R0		R1
×	×	·
R2		R3
×		×
R4		R5
×	×	×
R6		R7
×		·

I	U	H	U
·	·	×	·
N	Z	V	C
×	×	×	↕

·：不変
×：不定
↕：結果

仕　　様

ﾌﾟﾛｸﾞﾗﾑﾒﾓﾘ（ﾊﾞｲﾄ）
58
ﾃﾞｰﾀﾒﾓﾘ（ﾊﾞｲﾄ）
0
ｽﾀｯｸ（ﾊﾞｲﾄ）
0
ｸﾛｯｸｻｲｸﾙ数
1083
ﾘｴﾝﾄﾗﾝﾄ
可
ﾛｹｰｼｮﾝ
可
途中割込み
可

注　意　事　項

仕様のクロックサイクル数は、255バイトのブロックデータを転送した時の値です。

説　　明

(1)　機能詳細

(a)　引数の詳細は以下のとおりです。

R4L：入力引数として、ブロックデータの転送バイト数を設定します。

R5　：入力引数として、転送元のデータメモリエリアの先頭アドレスを設定します。

R6　：入力引数として、転送先のデータメモリエリアの先頭アドレスを設定します。

Cフラグ（CCR）　：出力引数として、ソフトウェアMOVE2のデータ長、アドレス設定のエラーの有無を示します。

Cフラグ＝0　：データ転送が終了したことを示します。

Cフラグ＝1　：入力引数の設定が間違っていることを示します。

(b)　図1-1にソフトウェアMOVE2の実行例を示します。

入力引数を①のように設定すると、②のように転送元（H´FD80～H´FD89）のデータを転送先（H´FE80～H´FE89）へブロック転送します。

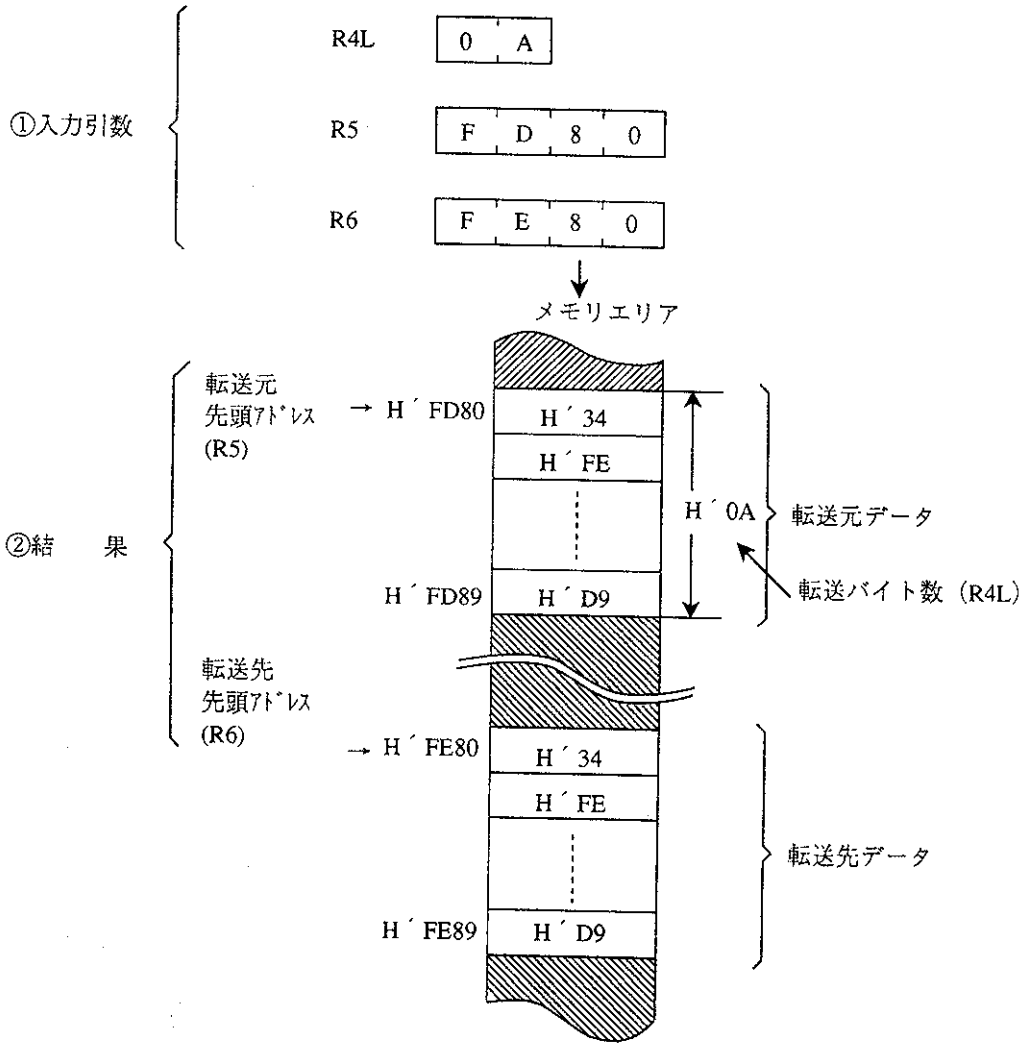


図1-1　ソフトウェアMOVE2の実行例

説　　明

(2)　使用上での注意

(a) R4Lはバイトですので、H´01≦R4L　≦H´FFの範囲のデータを設定してください。

(b) 図1-2のように転送先または転送先のデータメモリエリアがメモリエリア上の最終アドレスから（H´FFFF）先頭アドレス（H´0000）にまたがるような設定は避けてください。
この場合、正常に作動しません。

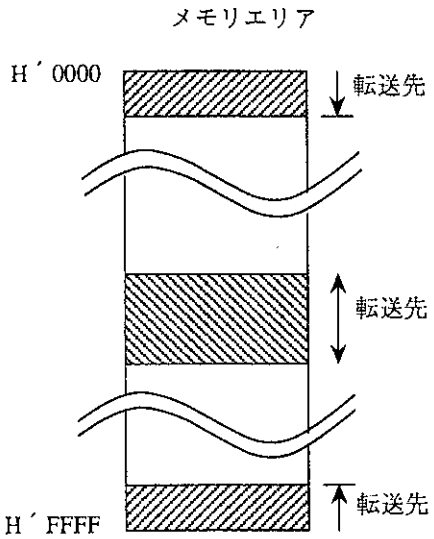


図1-2　データメモリエリアが上位アドレスから下位アドレスに配置されている場合

(3)　データメモリの説明

ソフトウェアMOVE2ではデータメモリを使用していません。

説　　明

(4)　使用例

転送元の先頭アドレス、転送先の先頭アドレスおよび転送バイト数を設定し、ソフトウェアMOVE2をサブルーチンコールします。

```
WORK1     .DATA.B  10      ……  ユーザプログラムで転送バイト数を設定するデータメモリエリアを確保（1byte:内容H´0A）します。
          .ALIGN   2       ……  データメモリエリア（WORK1）を偶数番地に設定します。
WORK2     .DATA.W  0       ……  ユーザプログラムで転送元の先頭アドレスを設定するデータメモリエリアを確保（2byte:内容H´0000）します。
WORK3     .DATA.W  0       ……  ユーザプログラム転送先の先頭アドレスを設定するデータメモリエリアを確保（2byte:内容H´0000）します。
               :
          MOV.B    @WORK1,R4L   ……  ユーザプログラムで設定した転送バイト数を入力引数に格納します。
          MOV.W    @WORK2,R5    ……  ユーザプログラムで設定して転送元のアドレスを格納します。
          MOV.W    @WORK3,R6    ……  ユーザプログラムで設定した転送先のアドレスを格納します。
          JSR      @MOVE2       ……  ソフトウェアMOVE2をサブルーチンコールします。
               :
```

(5)　動作原理

(a)　R5は転送元のアドレスを示すポインタ、R6は転送先のアドレスを示すポインタとして使います。

(b)　R4Lは転送バイト数を示すカウンタとして使います。1バイトのデータを転送するごとにデクリメントし、R4Lが“0”になったら終了します。

(c)　入力引数で転送データのバイト数が“0”もしくは転送元の先頭アドレスと転送先の先頭アドレスが同一の場合、Cフラグを“1”に設定（エラー表示）して終了します。

説　　明

(d)　図1-3のように転送先のデータメモリエリアの先頭アドレスBが転送元データメモリエリアの先頭アドレスAと最終アドレスA+n-1の間にある場合。

(A＜B＜A+n-1)

16ビット絶対アドレスアドレッシングモードで、転送元の上位アドレスのデータから、順にデータの転送を実行します。

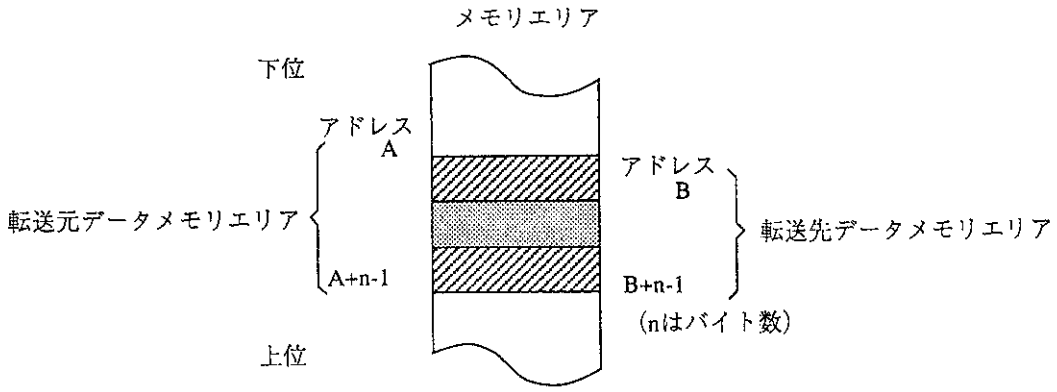
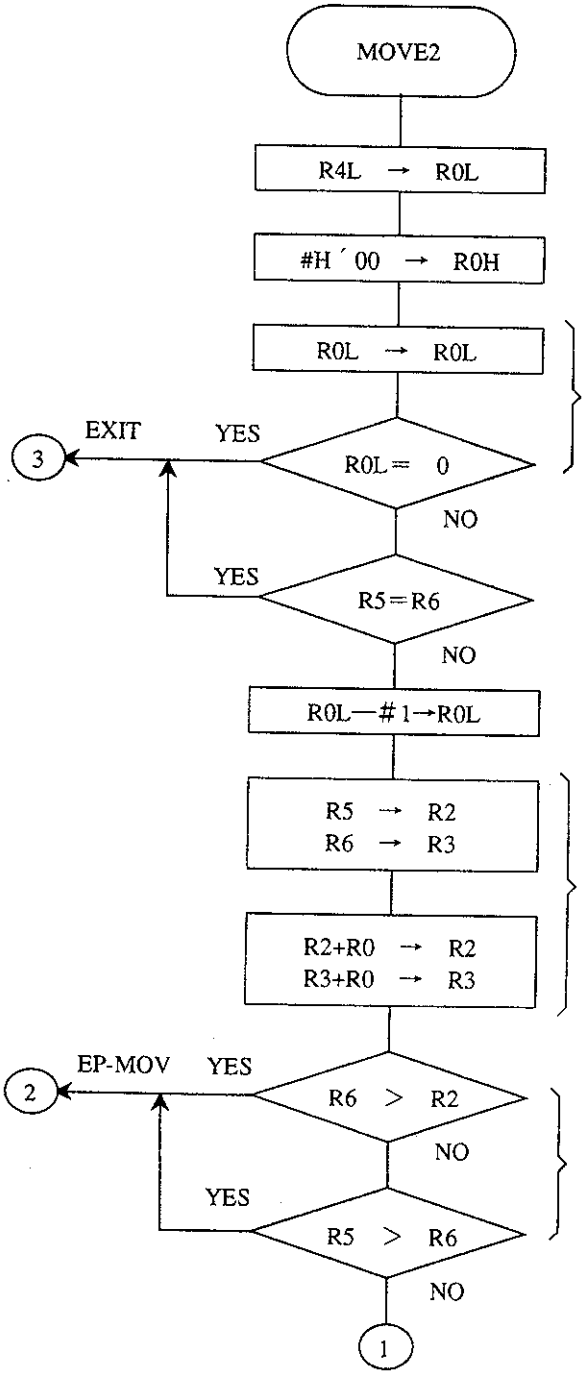


図1-3　データメモリエリアが重なる場合

(e)　(d) 以外の場合は、EEPMOV命令を用いて下位アドレスから順にデータの転送を実行します。

フローチャート



…… 転送先バイト数（R4L）をR0Lにコピーします。

…… R0Hをクリアします。

…… 転送バイト数が“0”であれば分岐します。

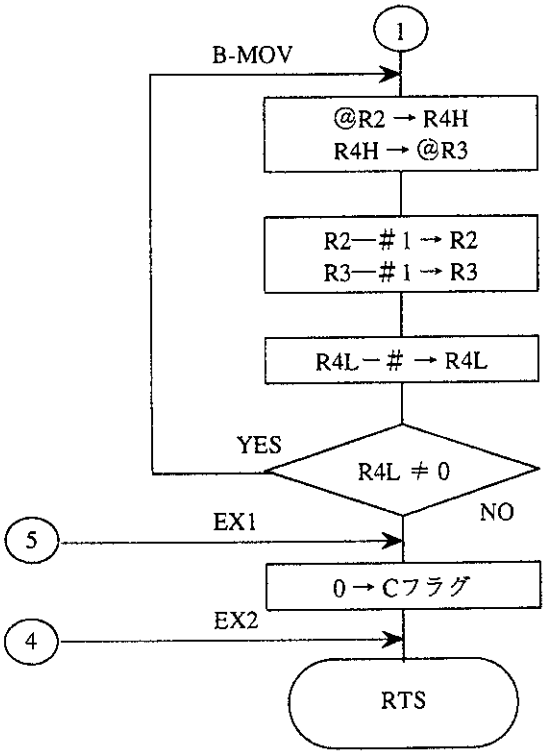
…… 転送元のアドレスポインタ（R5）と転送先のアドレスポインタ（R6）が同じであれば分岐します。

…… R0Lをデクリメントします。

…… R2に転送元データの最終アドレスを、R3に転送先データの最終アドレスを設定します。

…… 転送先の先頭アドレス（R6）が
R2＜R6＜R5
であれば分岐します。

フローチャート



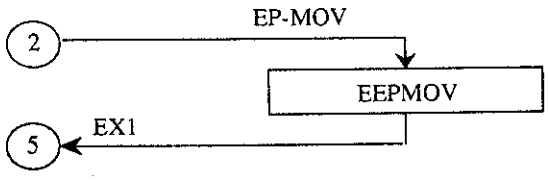
..... 転送先から転送先へ1バイトのデータを転送します。

..... R2,R3を各々デクリメントします。

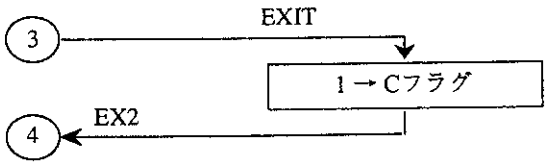
..... R4Lをデクリメントします。

..... 転送が終了したか判定します。

..... エラーの有無を示すビット（Cフラグ）を“0”にクリアします。



..... ブロック転送命令を実行します。



..... Cフラグを“1”にセットします。

プログラムリスト

```
*** H8/300 ASSEMBLER                    VER 1.0B **    08/18/92 09:46:07
PROGRAM NAME =

    1                    ;*****************************************************************
    2                    ;*
    3                    ;*    00 - NAME                    :BROCK DATA TRANSFER (MOVE2)
    4                    ;*
    5                    ;*****************************************************************
    6                    ;*
    7                    ;*    ENTRY              :R4L     (Byte counter)
    8                    ;*                        R5      (Source data start address)
    9                    ;*                        R6      (Destination data start address)
   10                    ;*
   11                    ;*    RETURN             :C bit of CCR (C=0;TRUE , C=1;FALSE)
   12                    ;*
   13                    ;*****************************************************************
   14                    ;
   15 MOVE2_co C 0000         .SECTION           MOVE2_code,CODE,ALIGN=2
   16                         .EXPORT            MOVE2
   17                    ;
   18 MOVE2_co C       00000000 MOVE2            .EQU     $          ;Entry point
   19 MOVE2_co C 0000 0CC8      MOV.B   R4L,R0L
   20 MOVE2_co C 0002 F000      MOV.B   #H'00,R0H
   21 MOVE2_co C 0004 0C88      MOV.B   R0L,R0L
   22 MOVE2_co C 0006 472E      BEQ     EXIT               ;If byte counter="0" then exit
   23 MOVE2_co C 0008 1D56      CMP.W   R5,R6
   24 MOVE2_co C 000A 472A      BEQ     EXIT               ;If R5=R6 then exit
   25 MOVE2_co C 000C 1A08      DEC.B   R0L
   26 MOVE2_co C 000E 0D52      MOV.W   R5,R2
   27 MOVE2_co C 0010 0D63      MOV.W   R6,R3
   28 MOVE2_co C 0012 0902      ADD.W   R0,R2              ;Set end address of source data
   29 MOVE2_co C 0014 0903      ADD.W   R0,R3              ;Set end address of destination data
   30 MOVE2_co C 0016 1D26      CMP.W   R2,R6
   31 MOVE2_co C 0018 4214      BHI     EP_MOV             ;Branch if R6>R2
   32 MOVE2_co C 001A 1D65      CMP.W   R6,R5
   33 MOVE2_co C 001C 4210      BHI     EP_MOV             ;Branch if R5>R6
   34 MOVE2_co C 001E           B_MOV
   35 MOVE2_co C 001E 6824      MOV.B   @R2,R4H            ;Load source data to R4H
   36 MOVE2_co C 0020 68B4      MOV.B   R4H,@R3            ;Store R4H to destination
   37 MOVE2_co C 0022 1B02      SUBS.W  #1,R2              ;Decrement source data pointer
   38 MOVE2_co C 0024 1B03      SUBS.W  #1,R3              ;Decrement destination data pointer
   39 MOVE2_co C 0026 1A0C      DEC.B   R4L
   40 MOVE2_co C 0028 46F4      BNE     B_MOV              ;Branch if R4L=0
   41 MOVE2_co C 002A           EX1
   42 MOVE2_co C 002A 06FE      ANDC.B  #H'FE,CCR          ;Clear C flag of CCR
   43 MOVE2_co C 002C           EX2
   44 MOVE2_co C 002C 5470      RTS
   45 MOVE2_co C 002E           EP_MOV
   46 MOVE2_co C 002E 7B5C598F  EEPMOV
   47 MOVE2_co C 0032 06FE      ANDC.B  #H'FE,CCR          ;Clear C flag of CCR
   48 MOVE2_co C 0034 40F4      BRA     EX1
   49 MOVE2_co C 0036           EXIT
   50 MOVE2_co C 0036 0401      ORC.B   #H'01,CCR          ;Set c flag for false
   51 MOVE2_co C 0038 40F2      BRA     EX2
   52                    ;
   53                         .END
*****TOTAL ERRORS        0
*****TOTAL WARNINGS      0
```

「MOV命令によるテスト」（No.1）を使用

6.2.二次元配列	ﾗﾍﾞﾙ名	ARRAY

機　　能

(1)　検索されたデータを、２次元配列（以下、配列と略します）から検索し、データが一致した時のアドレス、配列の要素（x, y）を設定します。
(2)　対象となるデータは、1バイトの符号無し整数です。
(3)　配列の要素は1バイトの符号無し整数です。
(4)　配列の大きさは255バイト×255バイトの範囲で設定できます。

引　　数

内　　容		格納場所	ﾃﾞｰﾀ長（ﾊﾞｲﾄ）
入　力	検索データ	R0L	1
	配列の先頭アドレス	R4	2
	配列の行の大きさ＝x	R2L	1
	配列の行の大きさ＝y	R3L	1
出　力	一致データのアドレス	R4	2
	一致データの配列要素x	R5H	1
	一致データの配列要素y	R5L	1
	一致データの有無	Cフラグ（CCR）	1

内部レジスタ変化およびフラグ変化

R0		R1	
×	·	·	
R2		R3	
×	×	×	×
R4		R5	
↕		↕	↕
R6		R7	
×		·	

I	U	H	U
·	·	×	·
N	Z	V	C
×	×	×	↕

·：不変
×：不定
↕：結果

仕　　様

ﾌﾟﾛｸﾞﾗﾑﾒﾓﾘ（ﾊﾞｲﾄ）
46
ﾃﾞｰﾀﾒﾓﾘ（ﾊﾞｲﾄ）
0
ｽﾀｯｸ（ﾊﾞｲﾄ）
0
ｸﾛｯｸｻｲｸﾙ数
1986
ﾘｴﾝﾄﾗﾝﾄ
可
ﾘﾛｹｰｼｮﾝ
可
途中割込み
可

注　意　事　項

仕様のクロックサイクル数は図2-1の例を実行した時の値です。
配列の要素どちらかが“0”であれば直ちに終了し、Cフラグをクリアします。

説　　明

(1)　機能詳細

(a)　引数の詳細は以下のとおりです。

(i) 入力引数として次のように設定します。

R0L：検索するデータ
R4　：配列の先頭アドレス
R2L：配列の行の大きさ（x）
R3L：配列の列の大きさ（y）

(ii) 出力引数として次のように設定されます。

R4　：一致データのアドレス
R5H：一致データの配列要素　x
R5L：一致データの配列要素　y
Cフラグ（CCR）　：ソフトウェアARRAY終了時の状態

Cフラグ＝1：検索データと一致するデータが配列上にあったことを示します。
Cフラグ＝0：検索データと一致するデータが配列上になかったことを示します。

(b) 図2-1にソフトウェアARRAYの実行例を示します。
①のように入力引数を設定すると、図2-2の配列（16×16）を参照し、②のように
一致データのアドレスがR4、配列要素xがR5H、配列要素yがR5Lに設定されます。

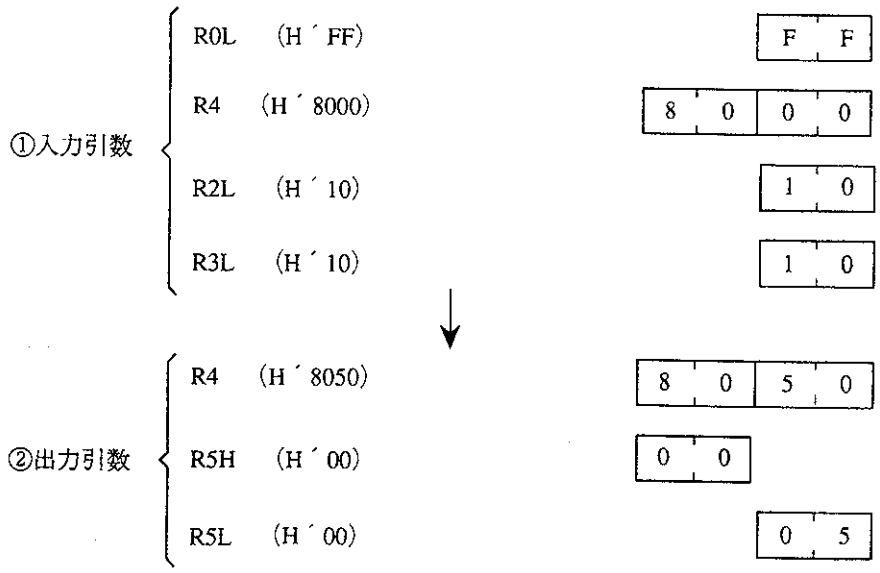


図2-1　ソフトウェアARRAYの実行例

説　　明

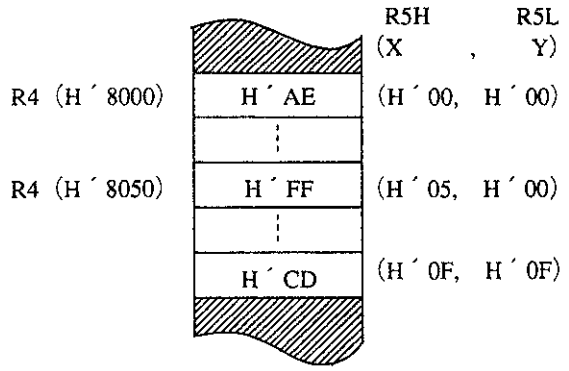


図2-2　配列の空間

(C)　ソフトウェアARRAYを実行する際、あらかじめ図2-3のような配列が必要となります。図2-3に配列の詳細について説明します。

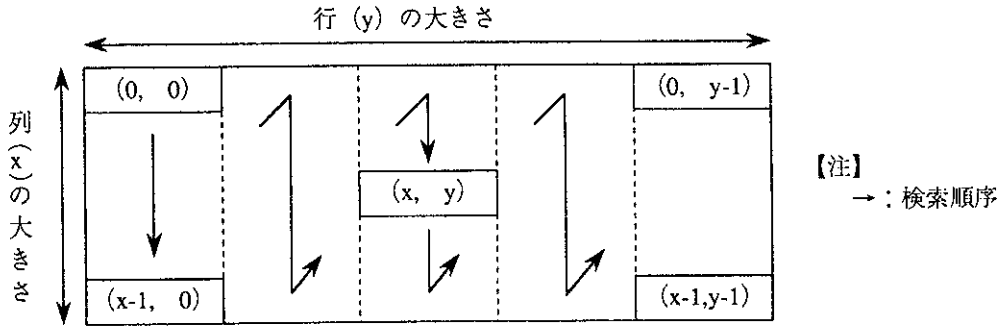


図2-3　2次元配列

(i) 行 (x) と列 (y) で配列の大きさを表します。
(ii) 配列の要素は (x,y) ＝ (行の要素、列の要素) で表わし、(0, 0) から (x-1, y-1) の範囲内で表わされます。
(iii) (0, 0) を配列の要素の先頭アドバイスとし、図2-3のような順で検索を行います。

(2) 使用上の注意

配列の大きさxと列の大きさyには“0”を設定しないでください。
“0”を設定するとデータ検索をせずにCフラグ (CCR) をクリアし終了します。

(3) データメモリの説明

ソフトウェアARRAYでは、データメモリを使用していません。

説　　明

(4)　使用例

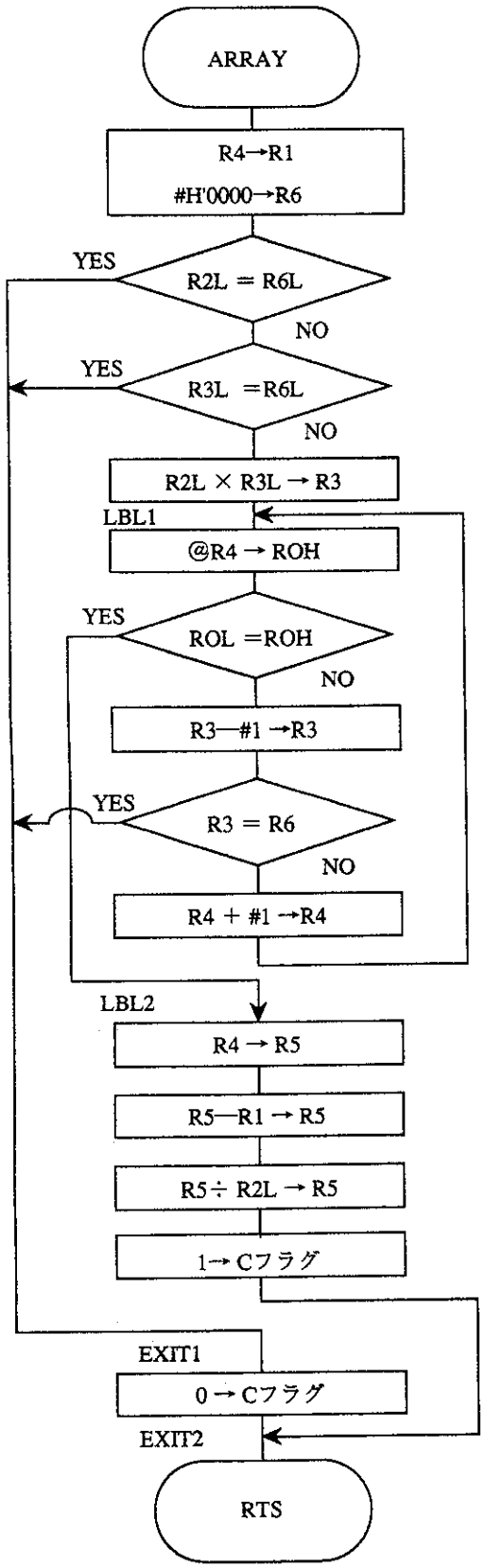
検索データと配列の先頭アドレス、行の大きさ、列の大きさを設定し、ソフトウェアARRAYをサブルーチンコールします。

```
I-WORK1     .RES.W  1       配列の先頭アドレスのデータメモリエリア
I-WORK2     .RES.B  1       配列の大きさ（x）のデータメモリエリア
I-WORK3     .RES.B  1       配列の大きさ（y）のデータメモリエリア
I-WORK4     .RES.B  1       検索データのデータメモリエリア
                :
O-WORK1     .RES.W  1       一致したﾃﾞｰﾀのｱﾄﾞﾚｽのﾃﾞｰﾀﾒﾓﾘｴﾘｱ
O-WORK2     .RES.B  1       一致したときの配列の要素（x）のﾃﾞｰﾀﾒﾓﾘｴﾘｱ
O-WORK3     .RES.B  1       一致したときの配列の要素（y）のﾃﾞｰﾀﾒﾓﾘｴﾘｱ
                :
                MOV.B   @1-WORK4,  R0L      ……検索データを設定します。
                MOV.W   @1-WORK1,  R4       ……配列の先頭アドレスを設定します。
                MOV.B   @1-WORK2,  R2H      ……配列の大きさ（x）を設定します。
                MOV.B   @1-WORK3,  R2L      ……配列の大きさ（y）を設定します。

                JSR     @ARRAY              ……ソフトウェアARRAYをサブルーチンコールします。

                MOV.W   R4,  @0-WORK1       ……一致したデータのアドレスを格納します。
                MOV.B   R2H, @0-WORK2       ……一致したときの配列の要素（x）を格納します。
                MOV.B   R2L, @0-WORK3       ……一致したときの配列の要素（y）を格納します。
                :
```

を確保します。

を確保します。

フローチャート



R4をR1に設定しR6をクリアします。

配列の行または列の大きさが“0”であれば終了します。

配列の大きさを求めます。

・配列データをROHに順次格納し、検索データと一致したら分岐します。
・また、配列データに検索データが存在しなかったら終了します。

配列の要素を除算によって求めます。

一致データが配列データ中に存在したことを設定します。
（Cフラグ＝“1”）

一致データが配列データ中に存在しなかったこと、または配列の大きさが“0”であったことを設定します。
（Cフラグ＝“0”）

プログラムリスト

```
***  H8/300 ASSEMBLER  Ver. 2.1A  ***   08/31/93 20:55:45
PROGRAM NAME =

                                    1 ;*********************************************************************
                                    2 ;*                                                                   *
                                    3 ;*      00 - NAME                :2-DIMENSIONAL ARRAY (ARRAY)        *
                                    4 ;*                                                                   *
                                    5 ;*********************************************************************
                                    6 ;*
                                    7 ;*      ENTRY            :R0L    (REFERENCE DATA)                   *
                                    8 ;*                        R2L    (NUMBER OF COLUM [X])              *
                                    9 ;*                        R3L    (NUMBER OF ROW   [Y])              *
                                   10 ;*                        R4     (ARRAY START ADDR)                 *
                                   11 ;*                                                                  *
                                   12 ;*      RETURNS          :R5H    (ARRAY ELEMENT OF COLUM [x])       *
                                   13 ;*                        R5L    (ARRAY ELEMENT OF LOW   [y])       *
                                   14 ;*                        R4     (MATCH DATA ADDR)                  *
                                   15 ;*                        C flag OF CCR (C=1;TRUE , C=0;FALSE)      *
                                   16 ;*                                                                  *
                                   17 ;*********************************************************************
                                   18 ;*
      0000                         19          .SECTION        ARRAY_code,CODE,ALIGN=2
                                   20          .EXPORT         ARRAY
                                   21 ;
            00000000               22 ARRAY           .EQU     $               ;Entry point
0000  0D41                         23          MOV.W   R4,R1
0002  1966                         24          SUB.W   R6,R6           ;Clear R6
0004  1CAE                         25          CMP.B   R2L,R6L
0006  4720                         26          BEQ     EXIT1           ;Branch if Z=1 then exit
0008  1CBE                         27          CMP.B   R3L,R6L
000A  471C                         28          BEQ     EXIT1           ;Branch if Z=1 then exit
000C  50A3                         29          MULXU   R2L,R3          ;Get total number of array(R3)
000E                               30 LBL1
000E  6840                         31          MOV.B   @R4,R0H         ;Load array data
0010  1C80                         32          CMP.B   R0L,R0H
0012  470A                         33          BEQ     LBL2            ;Branch if data find
0014  1B03                         34          SUBS.W  #1,R3           ;Decrement R3
0016  1D36                         35          CMP.W   R3,R6
0018  4710                         36          BEQ     EXIT2           ;Branch if false
001A  0B04                         37          ADDS.W  #1,R4           ;Increment data pointer
001C  40F0                         38          BRA     LBL1            ;Branch always
001E                               39 LBL2
001E  0D45                         40          MOV.W   R4,R5
0020  1915                         41          SUB.W   R1,R5
0022  51A5                         42          DIVXU   R2L,R5          ;Get count number of find data
0024  0401                         43          ORC.B   #H'01,CCR       ;Get array element [x,y]
0026  4002                         44          BRA     EXIT2           ;Set C flag of CCR
0028                               45 EXIT1                            ;Branch always
0028  06FE                         46          ANDC.B  #H'FE,CCR       ;Clear C flag of CCR
002A                               47 EXIT2
002A  5470                         48          RTS
                                   49 ;
                                   50          .END
*****TOTAL ERRORS       0
*****TOTAL WARNINGS     0
```

「汎用レジスタのクリア」（No.3）を使用

6.3符号付き32ビット2進数の加算	ﾗﾍﾞﾙ名	SADD

機　　能

(1)　符号付き32ビット2進数の加算を行ない、加算結果を汎用レジスタに設定します。
(2)　引数はすべて符号付き整数データを扱います。
(3)　データはすべて汎用レジスタ上で操作します。

引　　数

内　　容		格納場所	ﾃﾞｰﾀ長（ﾊﾞｲﾄ）
入　力	被加数	R0 ， R1	4
	加　数	R2 ， R3	4
出　力	加算結果	R0 ， R1	4
	桁上りの有無	Vフラグ（CCR）	1

内部レジスタ変化およびフラグ変化

R0	R1
↕	↕
R2	R3
・	・
R4	R5
・	・
R6	R7
×	・

I	U	H	U
・	×	×	×
N	Z	V	C
×	×	↕	×

・：不変
×：不定
↕：結果

仕　　様

ﾌﾟﾛｸﾞﾗﾑﾒﾓﾘ（ﾊﾞｲﾄ）
20
ﾃﾞｰﾀﾒﾓﾘ（ﾊﾞｲﾄ）
0
ｽﾀｯｸ（ﾊﾞｲﾄ）
0
ｸﾛｯｸｻｲｸﾙ数
44
ﾘｴﾝﾄﾗｲﾄ
可
ﾛｹｰｼｮﾝ
可
途中割込み
可

注　意　事　項

説　　明

(1)　機能詳細

(a)　引数の詳細は以下のとおりです。

(i) 入力引数は次のように設定します。

R0,R1　：符号付き32ビット2進数の被加数

R2,R3　：符号付き32ビット2進数の加数

(ii) 出力引数は次のように設定します。

R0,R1　：加算結果（符号付き32ビット2進数）

Vフラグ（CCR）　：演算の結果の桁上りの有無を示します。

Vフラグ＝1　：桁上りが生じたことを示します。

Vフラグ＝0　：桁上りが生じなかったことを示します。

(b)　図3-1にソフトウェアSADDの実行例を示します。

①のように入力引数を設定すると、②のように、加算結果をR0,R1に設定します。

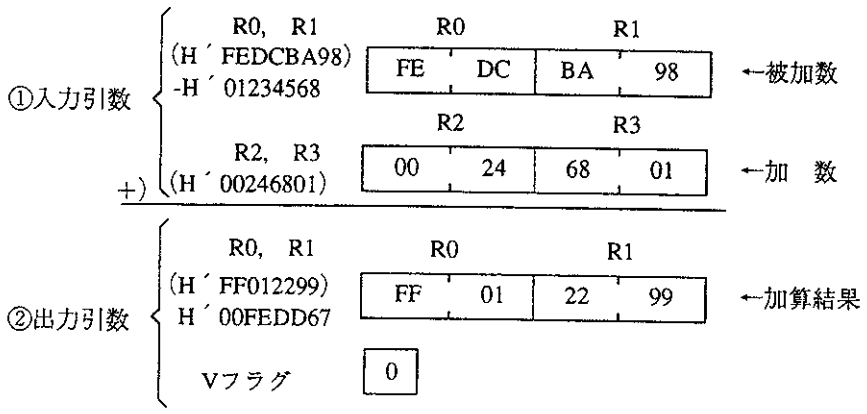


図3-1　ソフトウェアSADDの実行例

(2)　使用上の注意

(a)　ソフトウェアSADD実行後、R0,R1には加算結果が設定されるため、被加数は破壊されます。

実行後も被加数を必要とする場合は、あらかじめメモリ上に退避してください。

(3)　データメモリの説明

ソフトウェアSADDではデータメモリを使用していません。

説　　明

(4)　使用例

被加数および加数を設定し、ソフトウェアSADDをサブルーチンコールします。

```
WORK1    · RES.W   2          ……  ユーザープログラムで符号付き32ビット2進数の
WORK2    · RES.W   2          ……  被加　数 }  を設定するデータメモリエリアを
                                   加　数 }
WORK3    · RES.W   2          ……  加算結果 }  確保します。
                :
                :
         MOV.W   @WORK1,  RO      ……  ユーザープログラムで設定した符号付き32ビット
         MOV.W   @WORK1+2,  R1        2進数の被加数を入力引数に設定します。
         MOV.W   @WORK2,  R2      ……  ユーザープログラムで設定した符号付き32ビット
         MOV.W   @WORK2+2,  R3        2進数の加数を入力引数に設定します。
       | JSR       @SADD |      ……  ソフトウェアSADDをサブルーチンコールします。
         BVS       OVER
         MOV.W   R0,     @WORK3   ……  出力引数に設定された加算結果をユーザープログ
         MOV.W   R1,     @WORK3+2     ラムのデータメモリに格納します。
                :
OVER   | 桁上りの場合の処理ルーチン |
                :
```

説　　明

(5)　動作原理

(a) 符号付き32ビット2進数の加算は、加算命令（ADD.W命令、ADDX.B命令）を用いて処理します。

(b) 加算手順は以下のようになります。

（i）被加数をR0、R1に、加数をR2、R3に設定します。

(ii) CCRのユーザービット（ビット6, 4）とオーバーフローフラグ（ビット2）をクリアします。

(iii) 被加数が負の数であれば、符号ビットとしてCCRのユーザービット（ビット6）に“1”を設定します。
同様に加数が負の数であれば、符号ビットとしてCCRのユーザービット（ビット4）に“1”を設定します。

(iv) 被加数と加数を以下のように加算します。

R1+R3 → R1

R0L + R2l + C → R0L

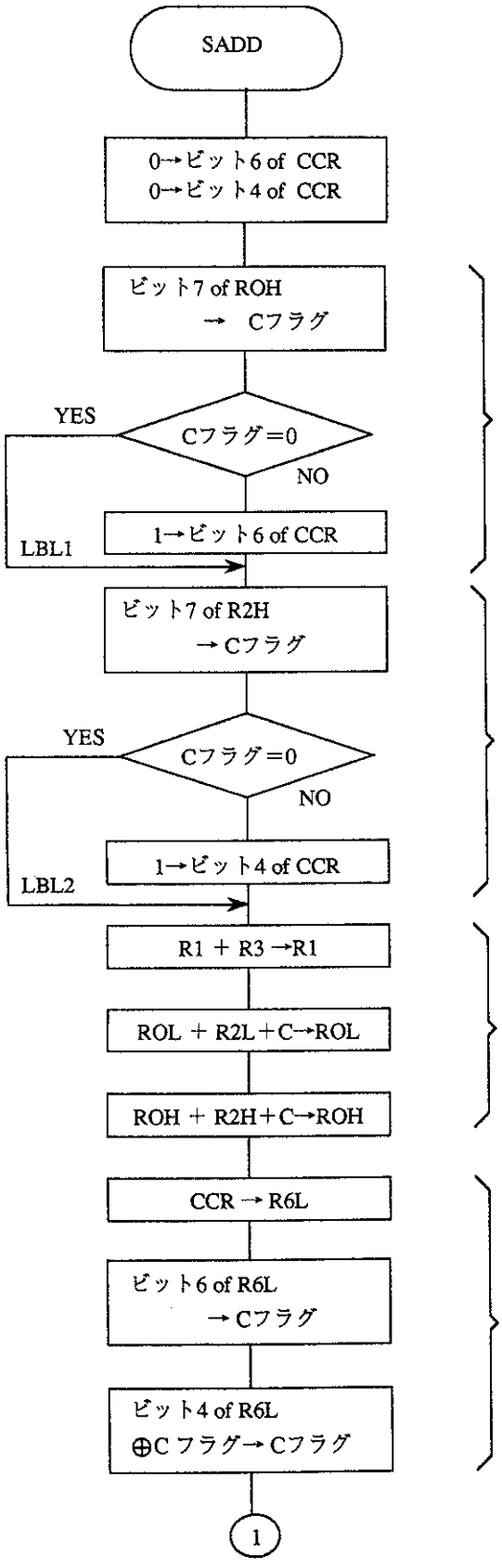
R0H + R2H+C → R0H

（v）次に符号ビット（CCRのユーザービット）を判定してVフラグを以下のように処理します。

〈符号ビット〉

ビット6 of CCR（被加数）	ビット4 of CCR（加数）	
0	0	→そのまま処理を続けます。
0	1	→Vフラグをクリアします。
1	0	
1	1	→そのまま処理を続けます。

フローチャート



CCRのユーザービット（ビット6,4）をクリアします。

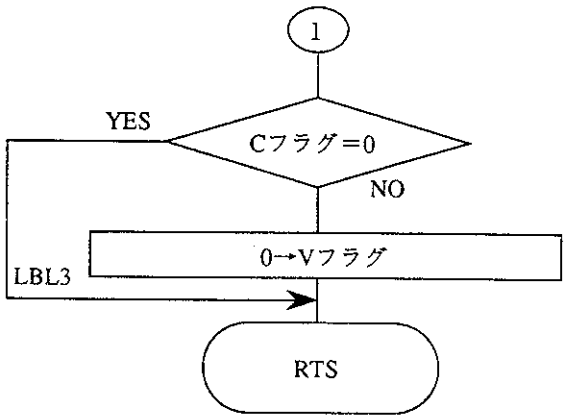
被加数の符号ビットをCCRのビット6に設定します。

加数の符号ビットをCCRのビット4に設定します。

被加数と加数を加算します。

被加数と加数の符号判定をします。

フローチャート



…… Vフラグをクリアします。

プログラムリスト

```
*** H8/300 ASSEMBLER                    VER 1.0B **   08/18/92 10:15:08
PROGRAM NAME =

    1                                   ;*****************************************************************
    2                                   ;*
    3                                   ;*   00 - NAME        :SIGNED 32 BIT BINARY ADDITION (SADD)
    4                                   ;*
    5                                   ;*****************************************************************
    6                                   ;*
    7                                   ;*   ENTRY            :R0     (UPPER WORD OF SUMMAND)
    8                                   ;*                     R1     (LOWER WORD OF SUMMAND)
    9                                   ;*                     R2     (UPPER WORD OF ADDEND)
   10                                   ;*                     R3     (LOWER WORD OF ADDEND)
   11                                   ;*
   12                                   ;*   RETURNS          :R0     (UPPER WORD OF RESULT)
   13                                   ;*                     R1     (LOWER WORD OF RESULT)
   14                                   ;*                     V FLAG OF CCR
   15                                   ;*                            (V=0;TRUE,V=1:OVERFLOW OR UNDERFLOW)
   16                                   ;*
   17                                   ;*****************************************************************
   18                                   ;
   19 SADD_cod C 0000                        .SECTION         SADD_code,CODE,ALIGN=2
   20                                        .EXPORT          SADD
   21                                   ;
   22 SADD_cod C       00000000      SADD .EQU     $                ;Entry point
   23 SADD_cod C 0000 06AD                  ANDC    #H'AD,CCR        ;Clear user bits and V flag of CCR
   24 SADD_cod C 0002 7770                  BLD     #7,R0H           ;Load sign bit of summand
   25 SADD_cod C 0004 4402                  BCC     LBL1             ;Branch if C=0
   26 SADD_cod C 0006 0440                  ORC.B   #H'40,CCR        ;Bit set user bit (bit 6 of CCR)
   27 SADD_cod C 0008                  LBL1
   28 SADD_cod C 0008 7772                  BLD     #7,R2H           ;Load sign bit of addend
   29 SADD_cod C 000A 4402                  BCC     LBL2             ;Branch if C=0
   30 SADD_cod C 000C 0410                  ORC.B   #H'10,CCR        ;Bit set user bit (bit 4 of CCR)
   31 SADD_cod C 000E                  LBL2
   32 SADD_cod C 000E 0931                  ADD.W   R3,R1            ;R3  + R1       -> R1
   33 SADD_cod C 0010 0EA8                  ADDX.B  R2L,R0L          ;R2L + R0L + C -> R0L
   34 SADD_cod C 0012 0E20                  ADDX.B  R2H,R0H          ;R2H + R0H + C -> R0H
   35 SADD_cod C 0014 020E                  STC     CCR,R6L          ;CCR -> R6L
   36 SADD_cod C 0016 776E                  BLD     #6,R6L           ;Bit load bit 4 of R6L
   37 SADD_cod C 0018 754E                  BXOR    #4,R6L           ;Bit exclusive OR sign bits
   38 SADD_cod C 001A 4402                  BCC     LBL3             ;Barnch if C=0
   39 SADD_cod C 001C 06FD                  ANDC.B  #H'FD,CCR        ;Clear V flag
   40 SADD_cod C 001E                  LBL3
   41 SADD_cod C 001E 5470                  RTS
   42                                   ;
   43                                        .END
*****TOTAL ERRORS       0
*****TOTAL WARNINGS     0
```

「ビットチェック＆ブランチ」（No.7）を使用

付録1　命令セット一覧表

表1　命令セット一覧（1）

ニーモニック		サイズ	アドレッシングモード／命令長（バイト）									オペレーション	コンディションコード						実行ステート数*
			#xx:8/16	Rn	@Rn	@(d:16, Rn)	@-Rn/@Rn+	@aa:8/16	@(d:8, PC)	@@aa	—		I	H	N	Z	V	C	
MOV	MOV.B #xx:8, Rd	B	2									#xx:8→Rd8	—	—	↕	↕	0	—	2
	MOV.B Rs, Rd	B		2								Rs8→Rd8	—	—	↕	↕	0	—	2
	MOV.B @Rs, Rd	B			2							@Rs16→Rd8	—	—	↕	↕	0	—	4
	MOV.B @(d:16, Rs), Rd	B				4						@(d:16, Rs16)→Rd8	—	—	↕	↕	0	—	6
	MOV.B @Rs+, Rd	B					2					@Rs16→Rd8 Rs16+1→Rs16	—	—	↕	↕	0	—	6
	MOV.B @aa:8, Rd	B						2				@aa:8→Rd8	—	—	↕	↕	0	—	4
	MOV.B @aa:16, Rd	B						4				@aa:16→Rd8	—	—	↕	↕	0	—	6
	MOV.B Rs, @Rd	B			2							Rs8→@Rd16	—	—	↕	↕	0	—	4
	MOV.B Rs, @(d:16, Rd)	B				4						Rs8→@(d:16, Rd16)	—	—	↕	↕	0	—	6
	MOV.B Rs, @-Rd	B					2					Rd16-1→Rd16 Rs8→@Rd16	—	—	↕	↕	0	—	6
	MOV.B Rs, @aa:8	B						2				Rs8→@aa:8	—	—	↕	↕	0	—	4
	MOV.B Rs, @aa:16	B						4				Rs8→@aa:16	—	—	↕	↕	0	—	6
	MOV.W #xx:16, Rd	W	4									#xx:16→Rd	—	—	↕	↕	0	—	4
	MOV.W Rs, Rd	W		2								Rs16→Rd16	—	—	↕	↕	0	—	2
	MOV.W @Rs, Rd	W			2							@Rs16→Rd16	—	—	↕	↕	0	—	4
	MOV.W @(d:16, Rs), Rd	W				4						@(d:16, Rs16)→Rd16	—	—	↕	↕	0	—	6
	MOV.W @Rs+, Rd	W					2					@Rs16→Rd16 Rs16+2→Rs16	—	—	↕	↕	0	—	6
	MOV.W @aa:16, Rd	W						4				@aa:16→Rd16	—	—	↕	↕	0	—	6
	MOV.W Rs, @Rd	W			2							Rs16→@Rd16	—	—	↕	↕	0	—	4
	MOV.W Rs, @(d:16, Rd)	W				4						Rs16→@(d:16, Rd16)	—	—	↕	↕	0	—	6

表1　命令セット一覧（2）

ニーモニック		サイズ	アドレッシングモード／命令長（バイト）									オペレーション	コンディションコード						実行ステート数*
			#xx:8/16	Rn	@Rn	@(d:16, Rn)	@-Rn/@Rn+	@aa:8/16	@(d:8, PC)	@@aa	—		I	H	N	Z	V	C	
MOV	MOV.W Rs, @-Rd	W					2					Rd16-2→Rd16 Rs16→@Rd16	—	—	↕	↕	0	—	6
	MOV.W Rs, @aa:16	W						4				Rs16→@aa:16	—	—	↕	↕	0	—	6
POP	POP Rd	W					2					@SP→Rd16 SP+2→SP	—	—	↕	↕	0	—	6
PUSH	PUSH Rs	W					2					SP-2→SP Rs16→@SP	—	—	↕	↕	0	—	6
MOVFPE	MOVFPE @aa:16, Rd	B						4				@aa:16→Rd（E同期）	—	—	↕	↕	0	—	⑤
MOVTPE	MOVTPE Rs, @aa:16	B						4				Rs→@aa:16（E同期）	—	—	↕	↕	0	—	⑤
ADD	ADD.B #xx:8, Rd	B	2									Rd8+#xx:8→Rd8	—	↕	↕	↕	↕	↕	2
	ADD.B Rs, Rd	B		2								Rd8+Rs8→Rd8	—	↕	↕	↕	↕	↕	2
	ADD.W Rs, Rd	W		2								Rd16+Rs16→Rd16	—	①	↕	↕	↕	↕	2
ADDX	ADDX.B #xx:8, Rd	B	2									Rd8+#xx:8+C→Rd8	—	↕	↕	②	↕	↕	2
	ADDX.B Rs, Rd	B		2								Rd8+Rs8+C→Rd8	—	↕	↕	②	↕	↕	2
ADDS	ADDS.W #1, Rd	W		2								Rd16+1→Rd16	—	—	—	—	—	—	2
	ADDS.W #2, Rd	W		2								Rd16+2→Rd16	—	—	—	—	—	—	2
INC	INC.B Rd	B		2								Rd8+1→Rd8	—	—	↕	↕	↕	—	2
DAA	DAA.B Rd	B		2								Rd8 10進補正→Rd8	—	*	↕	↕	*	③	2
SUB	SUB.B Rs, Rd	B		2								Rd8-Rs8→Rd8	—	↕	↕	↕	↕	↕	2
	SUB.W Rs, Rd	W		2								Rd16-Rs16→Rd16	—	①	↕	↕	↕	↕	2
SUBX	SUBX.B #xx:8, Rd	B	2									Rd8-#xx:8-C→Rd8	—	↕	↕	②	↕	↕	2
	SUBX.B Rs, Rd	B		2								Rd8-Rs8-C→Rd8	—	↕	↕	②	↕	↕	2

表1　命令セット一覧 (3)

ニーモニック		サイズ	アドレッシングモード／命令長 (バイト)									オペレーション	コンディションコード						実行ステート数*
			#xx:8/16	Rn	@Rn	@(d:16, Rn)	@-Rn/@Rn+	@aa:8/16	@(d:8, PC)	@@aa	—		I	H	N	Z	V	C	
SUBS	SUBS.W #1, Rd	W		2								Rd16-1→Rd16	—	—	—	—	—	—	2
	SUBS.W #2, Rd	W		2								Rd16-2→Rd16	—	—	—	—	—	—	2
DEC	DEC.B Rd	B		2								Rd8-1→Rd8	—	—	↕	↕	↕	—	2
DAS	DAS.B Rd	B		2								Rd8 10進補正→Rd8	—	*	↕	↕	*	—	2
NEG	NEG.B Rd	B		2								0-Rd→Rd	—	↕	↕	↕	↕	↕	2
CMP	CMP.B #xx:8, Rd	B	2									Rd8-#xx:8	—	↕	↕	↕	↕	↕	2
	CMP.B Rs, Rd	B		2								Rd8-Rs8	—	↕	↕	↕	↕	↕	2
	CMP.W Rs, Rd	W		2								Rd16-Rs16	—	①	↕	↕	↕	↕	2
MULXU	MULXU.B Rs, Rd	B		2								Rd8×Rs8→Rd16	—	—	—	—	—	—	14
DIVXU	DIVXU.B Rs, Rd	B		2								Rd16÷Rs8→Rd16 (RdH:余り, RdL:商)	—	—	⑥	⑦	—	—	14
AND	AND.B #xx:8, Rd	B	2									Rd8∧#xx:8→Rd8	—	—	↕	↕	0	—	2
	AND.B Rs, Rd	B		2								Rd8∧Rs8→Rd8	—	—	↕	↕	0	—	2
OR	OR.B #xx:8, Rd	B	2									Rd8∨#xx:8→Rd8	—	—	↕	↕	0	—	2
	OR.B Rs, Rd	B		2								Rd8∨Rs8→Rd8	—	—	↕	↕	0	—	2
XOR	XOR.B #xx:8, Rd	B	2									Rd8⊕#xx:8→Rd8	—	—	↕	↕	0	—	2
	XOR.B Rs, Rd	B		2								Rd8⊕Rs8→Rd8	—	—	↕	↕	0	—	2
NOT	NOT.B Rd	B		2								$\overline{Rd}$→Rd	—	—	↕	↕	0	—	2
SHAL	SHAL.B Rd	B		2									—	—	↕	↕	↕	↕	2

表1　命令セット一覧（4）

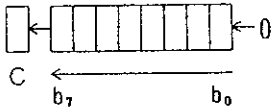
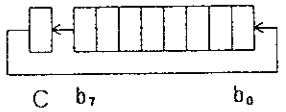
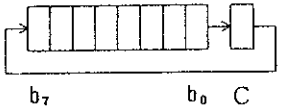
ニーモニック		サイズ	アドレッシングモード／命令長（バイト）									オペレーション	コンディションコード						実行ステート数*
			#xx:8/16	Rn	@Rn	@(d:16, Rn)	@-Rn/@Rn+	@aa:8/16	@(d:8, PC)	@@aa	—		I	H	N	Z	V	C	
SHAR	SHAR.B Rd	B		2									—	—	↕	↕	0	↕	2
SHLL	SHLL.B Rd	B		2									—	—	↕	↕	0	↕	2
SHLR	SHLR.B Rd	B		2									—	—	0	↕	0	↕	2
ROTXL	ROTXL.B Rd	B		2									—	—	↕	↕	0	↕	2
ROTXR	ROTXR.B Rd	B		2									—	—	↕	↕	0	↕	2
ROTL	ROTL.B Rd	B		2									—	—	↕	↕	0	↕	2
ROTR	ROTR.B Rd	B		2									—	—	↕	↕	0	↕	2
BSET	BSET #xx:3, Rd	B		2								(#xx:3 of Rd8)←1	—	—	—	—	—	—	2
	BSET #xx:3, @Rd	B			4							(#xx:3 of @Rd16)←1	—	—	—	—	—	—	8

表1　命令セット一覧 (5)

ニーモニック		サイズ	アドレッシングモード／命令長（バイト）									オペレーション	コンディションコード						実行ステート数*
			#xx:8/16	Rn	@Rn	@(d:16, Rn)	@-Rn/@Rn+	@aa:8/16	@(d:8, PC)	@@aa	—		I	H	N	Z	V	C	
BSET	BSET #xx:3, @aa:8	B						4				(#xx:3 of @aa:8)←1	–	–	–	–	–	–	8
	BSET Rn, Rd	B		2								(Rn8 of Rd8)←1	–	–	–	–	–	–	2
	BSET Rn, @Rd	B			4							(Rn8 of @Rd16)←1	–	–	–	–	–	–	8
	BSET Rn, @aa:8	B						4				(Rn8 of @aa:8)←1	–	–	–	–	–	–	8
BCLR	BCLR #xx:3, Rd	B		2								(#xx:3 of Rd8)←0	–	–	–	–	–	–	2
	BCLR #xx:3, @Rd	B			4							(#xx:3 of @Rd16)←0	–	–	–	–	–	–	8
	BCLR #xx:3, @aa:8	B						4				(#xx:3 of @aa:8)←0	–	–	–	–	–	–	8
	BCLR Rn, Rd	B		2								(Rn8 of Rd8)←0	–	–	–	–	–	–	2
	BCLR Rn, @Rd	B			4							(Rn8 of @Rd16)←0	–	–	–	–	–	–	8
	BCLR Rn, @aa:8	B						4				(Rn8 of @aa:8)←0	–	–	–	–	–	–	8
BNOT	BNOT #xx:3, Rd	B		2								(#xx:3 of Rd8)←$\overline{(\#xx:3 \text{ of } Rd8)}$	–	–	–	–	–	–	2
	BNOT #xx:3, @Rd	B			4							(#xx:3 of @Rd16)←$\overline{(\#xx:3 \text{ of } @Rd16)}$	–	–	–	–	–	–	8
	BNOT #xx:3, @aa:8	B						4				(#xx:3 of @aa:8)←$\overline{(\#xx:3 \text{ of } @aa:8)}$	–	–	–	–	–	–	8
	BNOT Rn, Rd	B		2								(Rn8 of Rd8)←$\overline{(Rn8 \text{ of } Rd8)}$	–	–	–	–	–	–	2
	BNOT Rn, @Rd	B			4							(Rn8 of @Rd16)←$\overline{(Rn8 \text{ of } @Rd16)}$	–	–	–	–	–	–	8
	BNOT Rn, @aa:8	B						4				(Rn8 of @aa:8)←$\overline{(Rn8 \text{ of } @aa:8)}$	–	–	–	–	–	–	8
BTST	BTST #xx:3, Rd	B		2								$\overline{(\#xx:3 \text{ of } Rd8)}$→Z	–	–	–	↕	–	–	2
	BTST #xx:3, @Rd	B			4							$\overline{(\#xx:3 \text{ of } @Rd16)}$→Z	–	–	–	↕	–	–	6
	BTST #xx:3, @aa:8	B						4				$\overline{(\#xx:3 \text{ of } @aa:8)}$→Z	–	–	–	↕	–	–	6
	BTST Rn, Rd	B		2								$\overline{(Rn8 \text{ of } Rd8)}$→Z	–	–	–	↕	–	–	2

表1　命令セット一覧（6）

ニーモニック		サイズ	アドレッシングモード／命令長（バイト）									オペレーション	コンディションコード						実行ステート数*
			#xx:8/16	Rn	@Rn	@(d:16, Rn)	@-Rn/@Rn+	@aa:8/16	@(d:8, PC)	@@aa	—		I	H	N	Z	V	C	
BTST	BTST Rn, @Rd	B			4							(Rn8 of @Rd16)→Z	—	—	—	↕	—	—	6
	BTST Rn, @aa:8	B						4				(Rn8 of @aa:8)→Z	—	—	—	↕	—	—	6
BLD	BLD #xx:3, Rd	B		2								(#xx:3 of Rd8)→C	—	—	—	—	—	↕	2
	BLD #xx:3, @Rd	B			4							(#xx:3 of @Rd16)→C	—	—	—	—	—	↕	6
	BLD #xx:3, @aa:8	B						4				(#xx:3 of @aa:8)→C	—	—	—	—	—	↕	6
BILD	BILD #xx:3, Rd	B		2								(#xx:3 of Rd8)→C	—	—	—	—	—	↕	2
	BILD #xx:3, @Rd	B			4							(#xx:3 of @Rd16)→C	—	—	—	—	—	↕	6
	BILD #xx:3, @aa:8	B						4				(#xx:3 of @aa:8)→C	—	—	—	—	—	↕	6
BST	BST #xx:3, Rd	B		2								C→(#xx:3 of Rd8)	—	—	—	—	—	—	2
	BST #xx:3, @Rd	B			4							C→(#xx:3 of @Rd16)	—	—	—	—	—	—	8
	BST #xx:3, @aa:8	B						4				C→(#xx:3 of @aa:8)	—	—	—	—	—	—	8
BIST	BIST #xx:3, Rd	B		2								$\bar{C}$→(#xx:3 of Rd8)	—	—	—	—	—	—	2
	BIST #xx:3, @Rd	B			4							$\bar{C}$→(#xx:3 of @Rd16)	—	—	—	—	—	—	8
	BIST #xx:3, @aa:8	B						4				$\bar{C}$→(#xx:3 of @aa:8)	—	—	—	—	—	—	8
BAND	BAND #xx:3, Rd	B		2								C∧(#xx:3 of Rd8)→C	—	—	—	—	—	↕	2
	BAND #xx:3, @Rd	B			4							C∧(#xx:3 of @Rd16)→C	—	—	—	—	—	↕	6
	BAND #xx:3, @aa:8	B						4				C∧(#xx:3 of @aa:8)→C	—	—	—	—	—	↕	6
BIAND	BIAND #xx:3, Rd	B		2								C∧($\overline{\text{#xx:3 of Rd8}}$)→C	—	—	—	—	—	↕	2
	BIAND #xx:3, @Rd	B			4							C∧($\overline{\text{#xx:3 of @Rd16}}$)→C	—	—	—	—	—	↕	6
	BIAND #xx:3, @aa:8	B						4				C∧($\overline{\text{#xx:3 of @aa:8}}$)→C	—	—	—	—	—	↕	6
BOR	BOR #xx:3, Rd	B		2								C∨(#xx:3 of Rd8)→C	—	—	—	—	—	↕	2
	BOR #xx:3, @Rd	B			4							C∨(#xx:3 of @Rd16)→C	—	—	—	—	—	↕	6
	BOR #xx:3, @aa:8	B						4				C∨(#xx:3 of @aa:8)→C	—	—	—	—	—	↕	6

表1　命令セット一覧（7）

ニーモニック		サイズ	アドレッシングモード／命令長（バイト）									オペレーション		コンディションコード						実行ステート数*
			#xx:8/16	Rn	@Rn	@(d:16, Rn)	@-Rn/@Rn+	@aa:8/16	@(d:8, PC)	@@aa	—		分岐条件	I	H	N	Z	V	C	
BIOR	BIOR #xx:3, Rd	B		2								C∨(#xx:3 of Rd8)→C		—	—	—	—	—	↕	2
	BIOR #xx:3, @Rd	B			4							C∨(#xx:3 of @Rd16)→C		—	—	—	—	—	↕	6
	BIOR #xx:3, @aa:8	B						4				C∨(#xx:3 of @aa:8)→C		—	—	—	—	—	↕	6
BXOR	BXOR #xx:3, Rd	B		2								C⊕(#xx:3 of Rd8)→C		—	—	—	—	—	↕	2
	BXOR #xx:3, @Rd	B			4							C⊕(#xx:3 of @Rd16)→C		—	—	—	—	—	↕	6
	BXOR #xx:3, @aa:8	B						4				C⊕(#xx:3 of @aa:8)→C		—	—	—	—	—	↕	6
BIXOR	BIXOR #xx:3, Rd	B		2								C⊕(#xx:3 of Rd8)→C		—	—	—	—	—	↕	2
	BIXOR #xx:3, @Rd	B			4							C⊕(#xx:3 of @Rd16)→C		—	—	—	—	—	↕	6
	BIXOR #xx:3, @aa:8	B						4				C⊕(#xx:3 of @aa:8)→C		—	—	—	—	—	↕	6
Bcc	BRA d:8 (BT d:8)	—							2			PC←PC+d:8		—	—	—	—	—	—	4
	BRN d:8 (BF d:8)	—							2			PC←PC+2		—	—	—	—	—	—	4
	BHI d:8	—							2			if condition	C∨Z=0	—	—	—	—	—	—	4
	BLS d:8	—							2			is true then	C∨Z=1	—	—	—	—	—	—	4
	BCC d:8 (BHS d:8)	—							2			PC←PC+d:8	C=0	—	—	—	—	—	—	4
	BCS d:8 (BLO d:8)	—							2			else next;	C=1	—	—	—	—	—	—	4
	BNE d:8	—							2				Z=0	—	—	—	—	—	—	4
	BEQ d:8	—							2				Z=1	—	—	—	—	—	—	4
	BVC d:8	—							2				V=0	—	—	—	—	—	—	4
	BVS d:8	—							2				V=1	—	—	—	—	—	—	4
	BPL d:8	—							2				N=0	—	—	—	—	—	—	4
	BMI d:8	—							2				N=1	—	—	—	—	—	—	4
	BGE d:8	—							2				N⊕V=0	—	—	—	—	—	—	4
	BLT d:8	—							2				N⊕V=1	—	—	—	—	—	—	4
	BGT d:8	—							2				Z∨(N⊕V)=0	—	—	—	—	—	—	4
	BLE d:8	—							2				Z∨(N⊕V)=1	—	—	—	—	—	—	4

表1　命令セット一覧（8）

ニーモニック		サイズ	アドレッシングモード／命令長（バイト）									オペレーション	コンディションコード						実行ステート数*
			#xx:8/16	Rn	@Rn	@(d:16, Rn)	@-Rn/@Rn+	@aa:8/16	@(d:8, PC)	@@aa	—		I	H	N	Z	V	C	
JMP	JMP @Rn	—			2							PC←Rn16	—	—	—	—	—	—	4
	JMP @aa:16	—						4				PC←aa:16	—	—	—	—	—	—	6
	JMP @@aa:8	—								2		PC←@aa:8	—	—	—	—	—	—	8
BSR	BSR d:8	—							2			SP-2→SP PC→@SP PC←PC+d:8	—	—	—	—	—	—	6
JSR	JSR @Rn	—			2							SP-2→SP PC→@SP PC←Rn16	—	—	—	—	—	—	6
	JSR @aa:16	—						4				SP-2→SP PC→@SP PC←aa:16	—	—	—	—	—	—	8
	JSR @@aa:8	—								2		SP-2→SP PC→@SP PC←@aa:8	—	—	—	—	—	—	8
RTS	RTS	—									2	PC←@SP SP+2→SP	—	—	—	—	—	—	8
RTE	RTE	—									2	CCR←@SP SP+2→SP PC←@SP SP+2→SP	↕	↕	↕	↕	↕	↕	10

表1　命令セット一覧 (9)

ニーモニック		サイズ	アドレッシングモード／命令長（バイト）									オペレーション	コンディションコード						実行ステート数*
			#xx:8/16	Rn	@Rn	@(d:16, Rn)	@-Rn/@Rn+	@aa:8/16	@(d:8, PC)	@@aa	—		I	H	N	Z	V	C	
SLEEP	SLEEP	—									2	低消費電力状態に遷移	—	—	—	—	—	—	2
LDC	LDC #xx:8, CCR	B	2									#xx:8→CCR	↕	↕	↕	↕	↕	↕	2
	LDC Rs, CCR	B		2								Rs8→CCR	↕	↕	↕	↕	↕	↕	2
STC	STC CCR, Rd	B		2								CCR→Rd8	—	—	—	—	—	—	2
ANDC	ANDC #xx:8, CCR	B	2									CCR∧#xx:8→CCR	↕	↕	↕	↕	↕	↕	2
ORC	ORC #xx:8, CCR	B	2									CCR∨#xx:8→CCR	↕	↕	↕	↕	↕	↕	2
XORC	XORC #xx:8, CCR	B	2									CCR⊕#xx:8→CCR	↕	↕	↕	↕	↕	↕	2
NOP	NOP	—									2	PC←PC+2	—	—	—	—	—	—	2
EEPMOV	EEPMOV	—									4	if R4L≠0 Repeat @R5→@R6 R5+1→R5 R6+1→R6 R4L-1→R4L Until R4L=0 else next;	—	—	—	—	—	—	④

【注】 *：実行ステート数は、オペコードおよびオペランドデータが内蔵メモリに存在する場合の値です。
①：ビット11から桁上がりまたはビット11へ桁下がりが発生したとき“1”にセットされ、それ以外のとき“0”にクリアされます。
②：演算結果がゼロのとき、演算前の値を保持し、それ以外のとき“0”にクリアされます。
③：補正結果に桁上がりが発生したとき“1”にセットされ、それ以外のとき演算前の値を保持します。
④：実行ステート数は、R4Lの設定値がnのとき4n+9となります。
⑤：Eクロック同期転送命令の実行ステート数は一定ではありません。
⑥：除数が負のとき“1”にセットされ、それ以外のとき“0”にクリアされます。
⑦：除数がゼロのとき“1”にセットされ、それ以外のとき“0”にクリアされます。

H8/300シリーズ, H8/300Lシリーズ プログラミングガイド

発行年月 平成5年3月 第1版
発　行 株式会社 日立製作所
半導体事業本部統括営業本部
編　集 株式会社 超LSIメディア
技術ドキュメントグループ

H8/300 シリーズ、H8/300L シリーズ
プログラミングガイド

ルネサス エレクトロニクス株式会社
神奈川県川崎市中原区下沼部1753 〒211-8668

ADJ-502-037